ESCOLA NAVAL

talant de bien faire

JOSÉ DIOGO CANDEIAS DE MAGALHÃES

REINFORCEMENT LEARNING: The application to Autonomous Biomimetic Underwater Vehicles control

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval – Ramo de Armas e eletrónica

Alfeite
2018

ESCOLA NAVAL

talantde bien faire

JOSÉ DIOGO CANDEIAS DE MAGALHÃES

REINFORCEMENT LEARNING: The application to Autonomous Biomimetic Underwater Vehicles control

Dissertação para obtenção do grau de Mestre em Ciências Militares Navais, na especialidade de Engenharia Naval – Ramo de Armas e eletrónica

Orientação de: Professor Doutor Bruno Duarte Damas

O Aluno Mestrando

O Orientador

_____

ASPOF EN-AEL José Diogo Candeias de Magalhães

_____

Professor Doutor Bruno Duarte Damas

Alfeite
2018

"You can never cross the ocean until you have the courage to lose sight of the shore."

- Christopher Columbus

Acknowledgments

The process of making a thesis is not easy and it would be impossible if it would be done alone. So, I would like to thank all the people that directly or indirectly helped me finish this road.

First of all, to all my family, that watched me grow for all these years and did all they could to keep my head held high and faced in the right direction.

To all my friends, that raised me off the ground so many times and were always available to talk and keeping my mind sane.

And last but not least, to my tutor Prof. Bruno Damas that were always there if I needed help and showed me different types of solutions to my problems when I didn't had any.

Abstract

In the last few years, the curiosity for studying the nature and its processes has been increasing in such a way, that the Human being has been trying to mimic its behaviors to try and solve everyday problems, and in relation to conventional methods solutions, some interesting results have been achieved by utilizing this new approach.

The international project SABUVIS, the project responsible for the idea of the content studied in this work, main objective is to use this new approach for intelligence, surveillance and reconnaissance of the surface and underwater picture, by using vehicles that mimic the movement of animals like fish and seals. Controlling this types of vehicles it's not trivial: due to its complex kinematics and dynamics making it hard to analytically derive controllers that can efficiently perform a given task, such as reaching a given position in a minimum time.

The objective of this document is to evaluate the results of the application of Artificial Intelligence (AI) in automatic biomimetic underwater vehicles (BUVs), by using a Reinforcement Learning algorithm (Q-learning), so that this type of vehicles are capable of reaching a desired position in a efficient way, providing a new way to control this new type of vehicles in which the algorithm is in constant learning.

Keywords: BUV, Artificial Intelligence, Reinforcement Learning, Q-learning.

Resumo

Nos últimos anos, a curiosidade pelo estudo da natureza e os seus processos tem vindo a aumentar, ao ponto de o ser Humano tentar imitar os seus comportamentos para tentar resolver os problemas que são enfrentados no dia a dia. Com este novo processo, têm surgido alguns resultados interessantes em relação às soluções obtidas pelos métodos convencionais.

O conteúdo deste trabalho insere-se no projeto internacional SABUVIS que tem como objetivo principal a utilização deste processo para inteligência e reconhecimento de panorâmicas de superfície e sub-superfície, através de veículos subaquáticos que imitam a locomoção de animais como o peixe e a foca. No entanto, o controlo do movimento destes veículos não é trivial: devido às suas complexas equações de cinemática e dinâmica é complicado derivar controladores que sejam eficientes para executarem uma tarefa simples, tal como deslocar-se até a um ponto num mínimo tempo possível.

Este documento tem como objetivo a avaliação de resultados da aplicação de Inteligência Artificial em veículos biomiméticos automáticos de subsuperfície (BUVs), através de um algoritmo de aprendizagem por reforço (Q-learning), para que este tipo de veículos sejam capaz de se deslocar até um ponto alvo de uma forma eficaz providênciando, assim, uma nova forma de locomução para BUVs que permite que o algoritmo se encontre em constante aprendizagem.

Palavras chave: BUV, Inteligência Artificial, Aprendizagem por Reforço, Q-learning.

Contents

Glossary

AI Artificial Intelligence

APA American Psychological Association

AUV Autonomous Underwater Vehicle

BUV Biomimetic Underwater Vehicle

CINAV Centro de Investigação Naval

CN3 Communication/Navigation Network Nodes

DP Dynamic Programming

DMS3 Destacamento de Mergulhadores Sapadores nº3

FEUP Faculdade de Engenharias da Universidade do Porto

ID Inspection/Identification

IMC Inter-Module Communication

IOP Institute of Physics

ISR Intelligence, Surveillance and Reconnaissance

LSTS Laboratório de Sistemas e Tecnologia Subaquática

MCM Mine Counter Measures

MDP Markov Decision Process

NRL Naval Research Lab

NUWC Naval Undersea Weapons Center

ROV Remotely Operated Vehicle

RL Reinforcement Learning

SABUVIS Swarm of Biomimetic Underwater Vehicles for Underwater Intelligence, Surveillance and Reconnaissance

SMPD Semi-Markov Decision Process

TCS Time Critical Strike

UAV Unmanned Aerial Vehicle

US United States

UUV Unmanned Underwater Vehicle

Definitions

Automation Process or procedure performed without human assistance.

Autonomy The time it takes for a battery to completely deplete.

1 Introduction

Biomimetics is a field that has been gaining an exponential popularity throughout the years (Lepora, Verschure, & Prescott, 2013). The study of biomimetic vehicles has been a point of interest in the military context due to its furtive capabilities: the ability of camouflage within the environment and they take even better interest for the Navy, because when comparing to conventional vehicles, they have a unique acoustic signature that will be much harder to identify by other acoustic sensors.

However, biomimetic vehicles come with one big problem: it isn't a trivial task to control such complex systems. Let's compare the process of implementing the controllers in a conventional vehicle, e.g. a car, and a biomimetic vehicle, e.g. a dog, by trying to execute the simple task: "move forward".

For the car-like vehicle, such task shouldn't be a problem, the only inputs needed would be a frequency and direction of rotation for the motors in each of the four wheels, that operate independently from each other, and the car would start moving forward.

The same task isn't that simple anymore when trying to control four legs for the dog-like vehicle. First of all, every leg would be composed of joints, which by itself will increase the number of inputs per leg. Secondly, if the legs operate independently from each other, the dog would most likely fall over, because the legs depend on one another, therefore, they need to cooperate with each other in order for the vehicle to execute the task given. This cooperation makes it much harder to map situations into actions for the controllers.

The objective of this work is to solve this type of problem via Reinforcement Learning, an AI algorithm that is capable of mapping states into actions, only by evaluating its interactions with the environment, without any previous knowledge of its surroundings. This eases the work done by humans, because there is no need to worry about programming every action to perform every possible state, leaving the algorithm learning from itself, by trial and error, to find an efficient group of actions that fulfills the objective given.

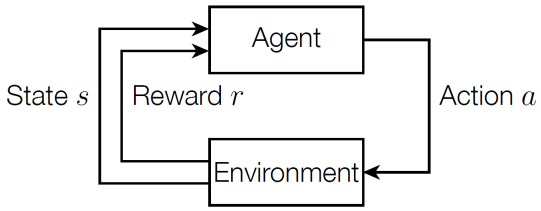


Fig. 1: The agent-environment interaction

My master thesis is the motivation of this written study, from a Weapons and Electronics Engineering course in the Portuguese Naval Academy, coordinated by Centro de Investigação Naval (CINAV), which is the organism responsible for the development and investigation in the Naval Academy.

Also, this study is part of an international project "Swarm of Biomimetic Underwater Vehicles for Underwater Intelligence, Surveillance and Reconnaissance (ISR) (SABUVIS)", whose main objective, as mentioned by its name, is to use Biomimetic Underwater Vehicles (BUVs) in missions for stealth data collection and surveillance.

This project will also measure the acoustic signature and the electrical efficiency of the biomimetic vehicle, in order to compare it to convectional vehicles. There is no proof that this robots produce less noise than conventional vehicles, thus one more reason for this project to be implemented.

This work will be divided in three main points. First, a brief survey where it will be presented the relevant moments for the history of biomimetics and the technologies already existing in the area.

The main part consists in the description of the processes that the simulator, a model of the kinematics and dynamics of the BUV in a software program explained in the Definition and problem analysis section, and the algorithm have been through until they reached their final form. In this section will be presented some chosen graphics that show the implementation and evolution of the algorithm.

And last, but not least, the evaluation of the results obtained while testing the algorithm. Were we will try to reach an answer to see if this type of Artificial Intelligence

is viable to control the movement of BUVs. It will also be explained the positive and negative aspects encountered during the making of this study.

In this work the references system used will be the American Psychological Association (APA) system.

2 Related Work

2.1 Unmanned Underwater Vehicles

Earth surface is covered by more than two-thirds of water, and yet, we know so litle about what misteries these vast seas hide. It is only natural for Human curiosity to kick in, and make us think about ways to explore this unknown part of the earth. This leads to Autonomous Underwater Vehicles (AUVs) that have been evolving in the last years, in terms of computing power, allowing for even more complex missions, and the amount of energy stored on board for longer missions, with the benefit of not having to risk human lives in the process (Yuh, 2000).

Unmanned Underwater Vehicle (UUV) is the designation given to any vehicle that operates underwater without human occupants. UUVs are divided in two categories: Remotely Operated Vehicles (ROVs), remotely controlled by a human (Christ & Wernli Sr, 2013), and AUVs, controlled without any direct human input (Paull, Saeedi, Seto, & Li, 2014). Due to AUVs relying on autonomous control they are more complex and more expensive compared to ROVs (Siciliano & Khatib, 2016).

Since water does not allow radio-frequency transmission and has an insufficient bandwidth by acoustic transmission for direct control, using UUVs provides some challenges like position uncertainty and noisy communication. Another challenge caused by water is the non linearity of the mapping between thruster command and generated force.

Besides all this challenges UUVs are very versatile and can offer capabilities in many areas especially when the mission may threaten to risk human life. Therefore, they are well suited for military operations where the human life risk is eminent.

2.1.1 UUVs in the Navy

The US Navy identifies five major benefits to using modern unmanned vehicles in maritime surface and sub-surface applications:

- Unmanned vehicles are far less expensive to operate and maintain than manned vehicles;
- Automated sensors are able to maintain near-constant awareness and coverage of an environment;
- Near-constant surveillance means persistence in data collection, enabling a better understanding of long-term behavior patterns and trends;
- Unmanned platforms also promise to improve productivity, as they allow manned platforms to pursue tasks elsewhere;
- Unmanned platforms keep human sailors and expensive manned platforms away from danger.

UUVs can be used in different applications and missions such as:

- Intelligence Surveillance and Reconnaissance (ISR) - (e.g. SHRIMP) (Muljowidodo, Adi, Budiyono, Prayogo, et al., 2009);
- Mine Countermeasures (MCM) - (e.g. REMUS) (Stokey et al., 2001);
- Anti-Submarine Warfare (Nicolai, 2002);
- Inspection/Identification (ID) - (e.g. DRIP) (Miller, 1996);
- Oceanography/Hydrography (Evans, Smith, Martin, & Wong, 1999);
- Communication/Navigation Network Nodes (CN3) (Barron, 1998);
- Payload Delivery (Brown & Clark, 2010);
- Time Critical Strike (TCS) (Lala & Harper, 1994).

In the Portuguese Navy there are the SEACon UUVs, developed by Laboratório de Sistemas e Tecnologia Subaquática (LSTS) of Faculdade de Engenharia da Universidade do Porto (FEUP). In the ambit of the project SEACon, three vehicles were delivered to the Portuguese Navy. This vehicles are operated by the Destacamento de Mergulhadores Sapadores nº3 (DMS3). The DSM3 has the responsibility of the operation of the UUVs due to its missions, such as:

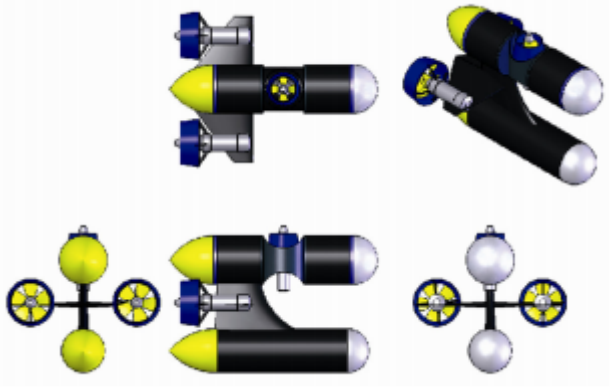


Fig. 2: SHRIMP ROV

Fig. 3: REMUS AUV

- Recognize and disarm explosives;
- Coordinate, conduct and execute sea rescue operations;
- Participate in operations of castaways rescue and recuperation of small crafts;
- Support Civil Protection services in case of a disaster;
- Cooperate with the responsible entities in the economic activities surveillance, related to sea exploration;
- Cooperate with the responsible entities in the suppression of illegal activities of narcotics traffic;
- Cooperate with the responsible entities in the scientific study of the aquatic environment;
- Perform searches, analysis and cleaning of waterways with access to disembark locations.

Fig. 4: DRIP UUV

Fig. 5: SEACon vehicle

Where the SEACon are prepared to be deployed in MCM, search and rescue, surveillance of objects in the bottom of the sea and scientific study missions (Silva, 2017).

The SEACon are cylindrical vehicles, with a torpedo shape, divided into three sections: front, mid and tail section. The three sections can be separated and it is possible to install different type of sensors, depending on the type of mission.

2.2 Biommimetics

The definition of Biomimetics is finding solutions for problems by mimicking nature systems, models and elements. For many years, humans have been studying nature behaviors to surpass their own challenges, like building a flying vehicle by having as a

model flying birds. Although the final products (airplanes and helicopters) aren't that similar to the initial study model, due to nature complex systems, robust, autonomous, and efficient solutions (minimizing the cost for maximum gain), that are completely adapted to the environment.

Biomimetics is multidisciplinary and it needs a good cooperation between all its fields to create a good end result that performs a given task.

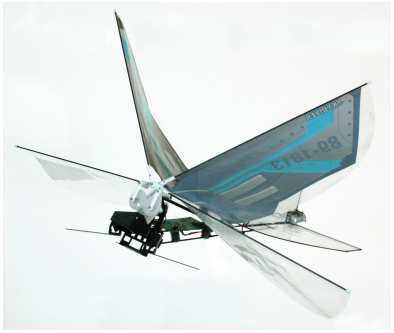
An explosive growth in the research of biomimetics has been happening in the last few years, the number of papers that are published has been doubling every 3 years. It started around 1950s with less than 100 each year, reaching currently around 3000 publications per year. Besides, the studies in this area are still increasing and it is expected that its growth continues in the future (Lepora, Verschure, & Prescott, 2013).

One of the early examples of a biomimetic implementation is the material Velcro, invented by the Belgian named Georges de Mestral, that examined and studied the morphology of tiny plants burrs, that after walk with his dog, were adhered to its fur. There are various types of biomimetics robots being studied today, one of those examples is the study of shape-shifting robots, studied by the Professor Maarja Krussma (Lepora, Mura, Krapp, Verschure, & Prescott, 2013).

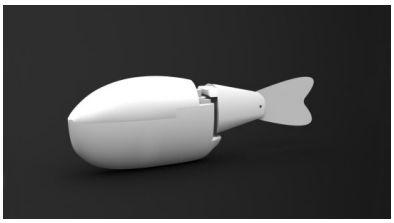
For AUVs, one of the crucial aspects is the movement, that is divided into three types of environments: air, ground and water. On ground, there are various types of mechanisms like legged (Nelson & Quinn, 1999) and worm-like crawling (Gonzalez-Gomez, Aguayo, & Boemo, 2005) and snake robots (Shugen, 2001), flapping-wing flight movement (Kim, Song, & Ahn, 2012) to travel through air and swimming (Szymak, Morawski, & Malec, 2012) to journey through the water.

The US Department of Defense has initiated some of the first efforts to the understanding of natural biological systems as a base model to future engineering systems, being the the earliest work made by the Navy dating back to late 1950s (Siochi et al., 2002).

(a) Ground movement mechanism (legged crawling, worm-like crawling and snake-like, respectively)



(b) Air movement mechanism (flapping-wing)



(c) Water movement mechanism (swimming)

Fig. 6: Biomimetic inspired robots

US Navy researches revolves around surface ships, the development of carbon nanotubes and organic composites for electronic and structural applications. There are two main research labs in the Navy working on Biomimetics: Naval Research Lab (NRL) working on biosensors (Thompson, 2005), self-assembly (Zhang, Marini, Hwang, & Santoso, 2002), and molecular engineering (Shchukin, Sukhorukov, Price, & Lvov, 2005); Naval Undersea Weapons Center (NUWC) that its interested in novel biomimetic propulsion techniques and hydrodynamic flow control, small-scale, semi-autonomous undersea probes.

Controlling this types of vehicles it's not trivial: due to its complex kinematics and dynamics making it hard to analytically derive controllers that can efficiently perform a given task, such as reaching a given position target in a minimum time. Hence, the need of an AI that learns by itself by its interactions with the environment, in order to easy the work of human programmers that had to do the controllers, of this type of vehicles, hand made.

2.3 Project SABUVIS

The work present in this document is part of an international project SABUVIS, whose main objective is to use BUVs in missions for stealth data collection and surveillance, where Portugal, Poland and Germany are the countries part of the consortium. In Portugal the collaborators are: OceanScan, LSTS and CINAV.

OceanScan is a leading international equipment company providing technology to the oil and gas, defense, petrochemical, renewables and nuclear industries and it can also supply personnel, one person or complete teams of surveyors to offshore survey and ROV markets. OceanScan is the one responsible to build the robot for the project SABUVIS.

Laboratório de Sistemas e Tecnologia Subaquática (LSTS) is an interdisciplinary research laboratory in Faculdade de Engenharia da Universidade do Porto (FEUP), established in 1997 and it is specialized on the design, construction, and operation of unmanned underwater, surface and air vehicles and on the development of tools and

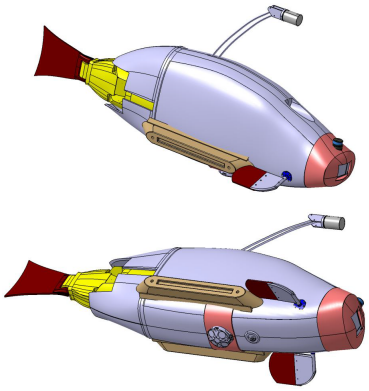


Fig. 7: Fish-like BUV 3D model

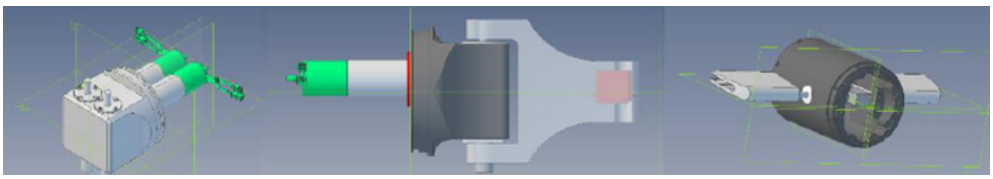


Fig. 8: Models for the controllers of a seal-like tail, fish-like tail and fins, respectively

technologies for the deployment of networked vehicle systems. LSTS is responsible for developing the software that will run on vehicle of the project.

Centro de Investigação Naval (CINAV) belongs to Escola Naval and has the mission of promoting the research, development and innovation in areas of great importance to the Portuguese Navy. It is responsible of developing the low level controllers and the Artificial Intelligence for the robot.

The SABUVIS project is divided into three types of vehicles:

- BUV 1 - A remotely controlled underwater vehicle mimicking a seal. The seal-like vehicle is composed by two lateral fins and a tail that is composed by two smaller flippers, trying to simulate a breaststroke or "frog"stroke style of swimming. It is being developed by the Poland Navy School;

- BUV 2 - It is also a remotely controlled vehicle, but this time mimicking a fish. The fish-like vehicle is integrated by two lateral fins and by a tail, composed by only one flipper sectioned into two parts to give more fluidity to the movement. It is being developed by Krakow university;

- BUV 3 - This underwater vehicle as similar shape as the above two, however its tail will be interchangeable between a fish-like and a seal-like tail. And while the the first two BUVs will be remotely controlled, some AI for the controllers will be implemented on BUV 3 for study purposes. It is being developed by OceanScan, LSTS and the Portugal Navy School.

These different mechanical structures inspired by biological systems present a much more complex kinematic structure that makes the task of controlling the motion of the vehicle a non trivial one. For a fish-like BUV, for example, it is not straightforward to develop controllers that actuate in the fins and tail of the vehicle in order to make it follow a desired trajectory in an efficient way. The purpose of this work is to develop adaptive controllers for these kind of vehicles based on RL techniques.

3 Reinforcement Learning

Reinforcement Learning (RL) is an area of machine learning that studies how agents take actions in order to maximize their reward.

The beginning of RL happens when two main independent threads intertwine. One concerning the learning experience by trial and error that was studied in the psychology of animal learning, the other one concerning the use of value functions and dynamic programming to solve the problem of optimal control. There is even a third thread, not so independent from the others, concerning temporal-difference methods. The fusion of this three threads produced the actual field of reinforcement learning.

RL is used in a large number of domains: continuous-time discounted algorithms were employed for elevator scheduling, preventive maintenance problems are usually semi-Markov decision problems (SMDPs) with an undiscounted objective function, a SMDP with a discount factor was employed for a cell phone network-management problem, and used in other domains (Gosavi, 2009).

The agent is the learner of the problem and everything that it can control directly, it's him that is going to choose its actions in order to complete the objective given, while the environment is everything that the agent can't control directly, although it is going to be influenced by the agent actions. This changes in the environment are going to be observed by the agent and will be the cause of its learning process.

In RL, an agent is supposed to choose its actions in a way to maximize an external reward that it gets from its interactions with the environment: a positive reward is given when it fulfills certain conditions defined by the programmer, and a lower reward is obtained when that condition is not met or partially met. In an episodic setting, where the task is restarted after each end of an episode, the objective is to maximize the total reward per episode, which is what happens in this experiment. This reward is tightly linked to the desired behavior for the agent and should be chosen carefully as the agent, will solely learn based on the rewards it gets. For instance, when learning to navigate from a position to other the agent should get higher value rewards as it gets closer to the desired position.

The agent and the environment may be modeled, while in a state $s \in S$ and perform actions $a \in A$, which can be multi-dimensional and be either discrete or continuous. A state s has the important information of the situation to predict future states, in a chess game a state could be the places of the pieces on the board. An action a is used to change the state that the agent is in, for example moving the king piece to one of it adjacent spaces in the example above. Following an action a there will be a new state *s'*.

Each time an action is taken, the agent will receive a reward R, a scalar value based on the state and new state. In the chess game: the agent receives 1 if the agent wins the game, -1 if it loses and 0 for all the other states, for example.

The agent always tries to maximize the cumulative reward in the long run. If we give the same importance to the reward expected throughout time, the function that the agent tries to maximize, at the time step t (Sutton, Barto, et al., 1998), is given by:

$$G_t = R_{t+1} + R_{t+2} + R_{t+3} + \cdots + R_T, \tag{1}$$

where T is a final time step. This function is used when there is a notion of final step, when the interactions between the agent and the environment can be separated into subsequences, called episodes.

When it isn't possible to break the task into episodes, like on-going task, where the final step is $T = \infty$, this could easily turn the function that we are trying to maximize to be infinite. This introduces the concept of discounting, defined by the γ parameter, $0 \leq \gamma \leq 1$, called the discounted rate, which determines the present value of future rewards:

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + \cdots + \sum_{k=0}^{\infty} \gamma^k R_{t+k+1}, \tag{2}$$

A policy is a mapping from states to probabilities of selecting each possible action, where $\pi(a|s)$ is the probability of taking action $a \in A$ if the agent is following the policy π and is in the state $s \in S$.

The state-value function for policy π, v_π, gives us the value of a state $s \in S$ under a policy π. While the action-value function for policy π, q_π, gives the value of taking action a in state s under that policy. Both this functions can be estimated from experience.

We can express the relationship between the value state and the value of its next states with the Bellman equation for v_π:

$$v_\pi = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S, \tag{3}$$

Most of the RL algorithms tend to estimate value functions (i.e. functions of states or state-actions pairs) that estimate how good it is for the agent to be in a state or to perform an action in a given state.

For MDPs, the state-value function (v_π) for policy π is defined by:

$$v_\pi(s) = E_\pi[G_t|S_t = s] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_t + k + 1|S_t = s], \forall s \in S, \tag{4}$$

where E_π is the expected value of a random variable given that the agent follows policy π.

And the action-value function (q_π) for policy π is defined by:

$$q_\pi(s, a) = E_\pi[G_t|S_t = s, A_t = a] = E_\pi[\sum_{k=0}^{\infty} \gamma^k R_t + k + 1|S_t = s, A_t = a] \tag{5}$$

Knowing the perfect model of the environment, an algorithm that can be used to compute optimal policies is part of a collection defined by the term: dynamic programming (DP).

The dynamic programming algorithms break the problem into a series of overlapping sub-problems, and by combining solutions to those smaller sub-problems, it can find the solution to bigger sub-problems.

Taking into consideration the property of value functions in dynamic programming, that satisfy recursive relationships similar to (2):

$$v_\pi(s) = \sum_a \pi(a|s) \sum_{s',r} p(s', r|s, a)[r + \gamma v_\pi(s')], \forall s \in S, \tag{6}$$

There are various methods to solve this problem but the one being studied in this work is the Q-learning algorithm.

In Reinforcement Learning, the agent tries to maximize its return in an unknown environment by performing actions and getting rewards, trying to understand how future rewards are affected by action it takes.

Markov Decision Process (MDP), or stochastic dynamic programs, are models for sequential decision making when outcomes are uncertain (Puterman, 2014). MDP is a good way to model this task:

- The agent has a finite set S of states that it can perceive and a finite set A of actions that it can perform;
- The agent perceives the current state and chooses an action to execute;
- The environment responds with a return (good or bad) and a new state;
- The agent may not have access to what will happen when it chooses an action in that state, but the only important information to decide is the action choice in the current state.

If the probability distribution of future states is only dependable of the present state, and not the on the ones that lead up to that state, it is said that the process has the Markov property.

There are a lot of ways to implement the learning focus in this work we will only focus on the Q-learning algorithm, a Reinforcement Learning algorithm that the agent learns to assign values to state-action pairs.

3.1 Q-Learning

Q-Learning is a Reinforcement Learning algorithm that estimates, from its interaction with the environment, the utility (calculate the maximum expected future reward) of performing a given action in a particular state, given by Q(S, A) [6]. This policy can be learned from this interaction, without the need to have a model for this environment, using the update rule.

$$Q(S,A) = Q(S,A) + \alpha[R + \gamma \max_a Q(S',a) - Q(S,A)] \ , \tag{7}$$

where Q denotes the expected accumulated future reward obtained if action A is performed in state S. S' is the observed next state after executing action A and R the corresponding reward. $0< \alpha \leq 1$ is a step-size parameter and $0< \gamma \leq 1$ denotes the discount-rate factor: the lower this value the lesser the importance given to distant future rewards, i.e, the more myopic the agent is regarding future rewards.

The learned action-value function, Q, directly approximates q_*, the optimal action-value function, independent of the policy being followed, which leads to early convergence proofs. This was the criteria of selection of the RL algorithm, due to the limited time provided for this project.

Although the Q-learning algorithm is a tabular method working with discrete values of states and actions, there are other methods that work with continuous states, like function approximation with tabular methods, and methods that work with both continuous states and actions, such as: policy, improvement, PI^2, PoWer, Reinforce, that will not be studied here.

4 Definition and problem analysis

This section serves as an introduction to the practical work done in this project, defining the main objective and problem. While it also briefly explains explain the tools utilized during the process.

4.1 Objective

Has mention in section 2.2, controlling the movement of biomimetic vehicles is not an easy task, due to the complex kinematics and dynamics of the nature mechanisms that are being mimicked. So, instead of mapping every single state into an action, what if the robot learned by itself to do so?

The objective of this work is to evaluate the performance of Reinforcement Learning methods in a BUV, built by the national consortium, in order to make it possible for the robot to follow a trajectory, defined by a various points, swimming in sequence, from one point to the next.

4.2 Problem definition

In this work the objective is to make a fish-like vehicle learn from itself through a RL algorithm. The vehicle is composed by two lateral fins and a simple fish-like tail sectioned in two parts.

In order to solve the problem above, the Q-learning algorithm will be used, and to do so is necessary to define how we are going to approach the problem.

4.2.1 State S

First, we need to determine the most relevant information at every moment, that follows a Markov property, this information is called the state of the vehicle. State being a possible situation that the robot can find itself into, from a group of possible situations, this group is defined by be user.

It is important that this information is as simple as possible, but at the same time, tries to summarize every past event that has future consequences for the task given, for example, in a tic-tac-toe game it is only necessary to know how the board is filled (which spaces are filled in with "X"or "O"or if they are empty) and which player's turn is it, the order in which the board was filled is not relevant for future plays.

There is another problem to take into consideration: the curse of dimensionality. In machine learning, when trying to learn how to map states into actions in a high-dimensional space, although it leads to a refined result, a vast amount of time is needed to train that data, and grows in an exponential way.

The factors that are going to describe the state to this problem will be: the distance to the goal and the angle in relation to the goal.

4.2.2 Action A

Defining the actions is the second step. An action is how the vehicle is going to change from one state to another, and it suffers from the same curse of dimesionality as the states.

The actions will be defined by the degrees of freedom that describe the movement for each fin and tail section. These parameters are: the frequency, the mean value of oscillation and the amplitude of oscillation, which are the same parameters that are used to represent a waveform.

4.2.3 Reward R

At last, the robot needs to know if it is getting closer to fulfill the task given, it is possible to know that by defining the reward function.

Measuring the euclidean distance to the goal seems to be the best reward function. It would be also important to take into consideration the roll angle, to try and maintain the vehicle in a straight up position as much as possible.

4.3 Simulator

Has mentioned, the derivation of the kinematics and dynamics for BUVs is not an easy task. A study for this type of equations was made by Polish Commander Piotr, due to the SABUVIS project, for a fish-like vehicle with two lateral fins and a tail with only one degree of freedom (not sectioned)(Szymak, 2016).

$$M\dot{v} + D(v)v + g(\eta) = \tau \ , \tag{8}$$

- v - vector of linear and angular velocities in the movable system;
- η - vector of vehicle position coordinates and its Euler angles in the immovable system;
- M - matrix of inertia (the sum of the matrices of the rigid body and the accompanying masses);
- $D(v)$ - hydrodynamic damping matrix;
- $g(\eta)$ - vector of restoring forces and moments of forces of gravity and buoyancy;
- τ - vector of control signals (the sum of vector of forces and moments of force generated by propulsion system τ_p and by environmental disturbances τ_d)

In this simulator the fish-like vehicle is compared to a point particle in a space where the gravity and buoyancy forces are applied to the vehicle, and it will determine the position and velocity of the vehicle when exterior forces, of the tail and fins behavior, are applied to it.

Although the real vehicle has two sections, the main objective of this work is to study the viability of using a RL algorithm to solve this problem. It will be as, if the second section of the tail would follow the movement of the first section, by not having a frequency and the mean value of oscillation being 0.

Commander Piotr transferred those dynamics and kinematics to a software platform MATLAB, where it receives the following inputs:

- F - the frequency of oscillation for the tail fin [rpm];
- K - medium oscillation value of the tail;
- F1 - the frequency of oscillation for the left fin [rpm];
- K1 - medium oscillation value of the left fin;
- F2 - the frequency of oscillation for the right fin [rpm];
- K2 - medium oscillation value of the right fin;
- T - simulation time;
- n - the number of iterations.

And were the output was a state vector, with the following variables:

- x - position in the x axis;
- y - position in the y axis;
- z - position in the z axis;
- u - velocity in the x axis;
- v - velocity in the y axis;
- w - velocity in the z axis;
- psi - yaw angle;
- theta - pitch angle;
- phi - roll angle;

In the simulation, due to the fish-like robot being compared to a point particle, it was considered that no trimming was done in terms of pitch and roll angle. This parameters will only by influenced by the behavior of the tails and fins.

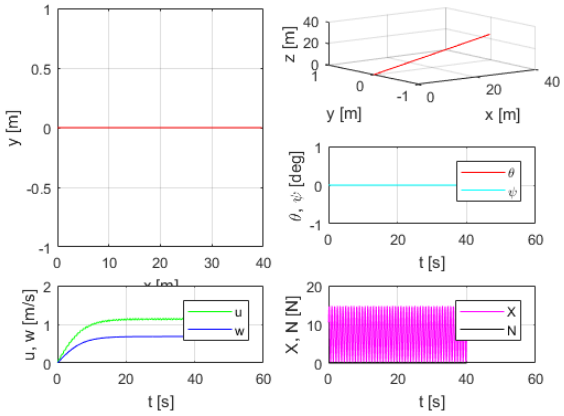


Fig. 9: Simulation run with default input values

By running the simulator for the first time, with the default input parameters (F = 1000, K = 0, F1 = 1000, K1 = 0, F2 = 1000 and K2 = 0), we can observe that, with this parameters, the robot navigates in a straight line raising its altitude (in the z axis) due to its buoyancy. 9.

4.4 LSTS toolchain

With the code up and running and tested in MATLAB, it is necessary to translate it to the language that will run on the software that runs on the vehicle.

LSTS created the software toolchain Neptus-IMC-DUNE that will be implemented in the project SABUVIS. During the realization of this thesis, it was given one week with LSTS to learn their toolchain, and it was dived in three main parts: DUNE, IMC and Neptus (Pinto et al., 2013).

DUNE is the on-board software running on the vehicle, it has a C++ programming environment and it is responsible for navigation, code for control and access to sensors and actuators. It is CPU architecture and operating system independent. Due to its versatility, it also runs in Manta communication gateways. It also has the advantage of running in small memories (16 megabytes).

Inter-Module Communication (IMC) protocol is a message oriented protocol to build interconnect systems of vehicles, sensors and human operators that pursue com-

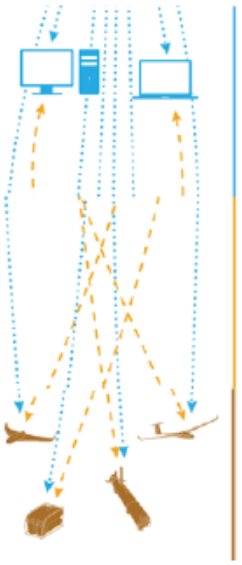


Fig. 10: LSTS toolchain, Neptus-IMC-Neptus

mon goals. It provides shared set of messages, abstracting from hardware and communication heterogeneity, that can be serialized and transferred over different means. It allows different tasks, from sensor drivers to guidance controllers, that run independently from each other on separate threads or processes, to exchange data using the message bus mechanism.

Neptus is a command, control, communications and intelligence framework for operations with vehicles, systems and human operators. It provides a coherent visual interface to command all the classes of autonomous vehicles and sensors.

Different types of geographical data can be used while planning, including S57 charts, tiled raster images from various sources and user-defined features. Plans can be simulated and validated before execution according to vehicle capabilities (battery endurance, maneuver support, sensors, etc). It is also able to visualize real-time data from multiple vehicles and after the mission it is possible to compile its results or from individual plan execution for review and analysis, for future adjustments.

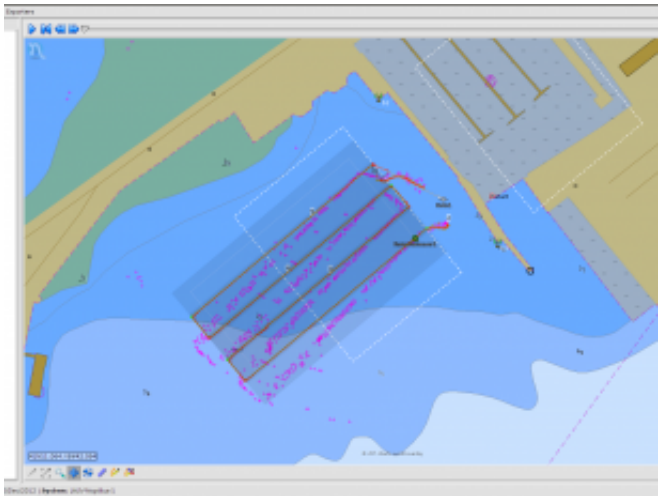


Fig. 11: Neptus human operator interface

5 Implementation

In this chapter, will try and consist on a critical analysis of the results obtained by the simulator. This is not a trivial problem, because there are to many open parameters that will influence the convergence of the algorithm, like the actualization time, simulation time per episode, step-sized parameter, discounted factor and the probability of exploration (ϵ-greedy).

The line of thought of this work was to begin using a very similar structure, of inputs and out puts, of the Commander Piotr work, and as it begins to show some good results, start changing the structure to make it more user friendly and limiting some variables in order to try and simulate the most real environment possible.

The performance measure that is going to be utilized to evaluate this results is the accumulative reward, which is a sum of all the rewards obtained in each iteration throughout each episode.

5.1 First approach

The tabular nature of the vanilla Q-Learning algorithm requires a discretized set of actions and states, there are modified version of this algorithm that accept continuous states and actions. Taking advantage of the inputs defined by Commander Piotr in his simulator (where the maximum frequency for the tail and fins was 3000 rpm and minimum 0 rpm and by giving an arbitrary numbers to the deviation, because it was not defined in Commander Piotr work the deviation interval), with respect to actions we define A as the variables whose columns contain the discretized action values for each component of the action vector

$$A = \begin{bmatrix} 3000 & 500 & 3000 & 500 & 3000 & 500 \\ 2250 & 250 & 2250 & 250 & 2250 & 250 \\ 1500 & 0 & 1500 & 0 & 1500 & 0 \\ 750 & -250 & 750 & -250 & 750 & -250 \\ 0 & -500 & 0 & -500 & 0 & -500 \end{bmatrix}$$

where each column represents F, K, F1, K1, F2 and K2 respectively. This way, we have a set of 5^6 different actions from where to choose at each iteration.

We consider the state to be the position in each axis, in meters, and the vehicle angle in the xy plane (in radians). The state variables S, that contain that discrete values, is then defined as

$$S = \begin{bmatrix} 1 & 1 & 1 & 0.6 \\ 2 & 2 & 2 & 1.2 \\ \vdots & \vdots & \vdots & \vdots \\ 10 & 10 & 10 & 6 \end{bmatrix},$$

where each column contains the possible values for x, y, z, in steps of 1, and the yaw angle, in steps of 0.6, respectively. This makes 10^4 distinct states.

The objective of this simulation is to reach a goal point and stay there. To fulfill it, the reward function is defined as

$$R = -\sqrt{x^2 + y^2 + z^2} \tag{9}$$

so that the reward will be higher the closer it gets to the goal.

The reward had to be made negative due to Q, that is initialized with zeros. By making it negative, it is guaranteed that each action is selected at least one time for each state, in the infinite number of episodes.

The simulation is divided by episodes. Each episode starts the vehicle in the same position with a, approximately, 4 meter distance to goal and with the vehicle pointed at it, each episode it has 200 iterations, each one representing 0.1 seconds, and it end when 20 seconds have passed, starting a new episode. This time was based when running the simulator given by Commander Piotr results, we took into consideration the time it took to travel a set distance and added a more time so that is possible for the algorithm to learn from its mistakes.

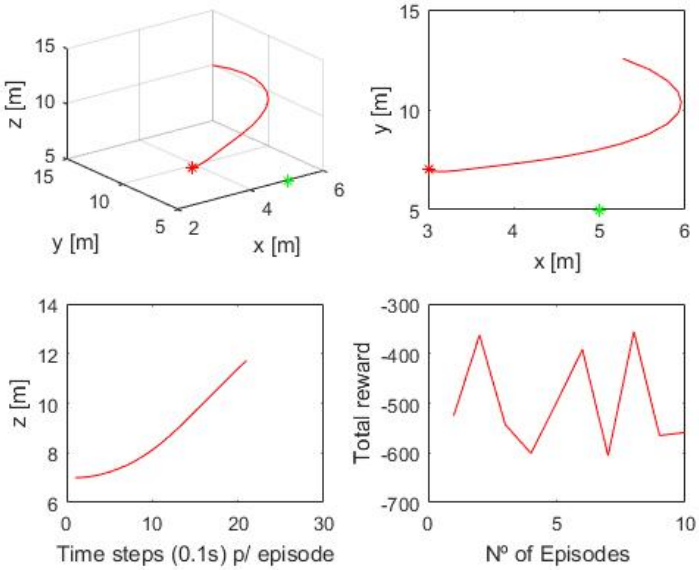


Fig. 12: First 10 episodes of the program

The graphics above show one of the trajectories of the vehicle, in a 3D dimension and 2D x0y axis, its altitude during the episode and its accumulative reward in the past episodes. The red star being the starting position and the green star the goal.

It is noted that the accumulative reward has already started to converge, which means that is starting to reach a satisfactory solution, but it is taking a long time to do it because of the dimension of the state and action variables and also some MATLAB functions (e.g. interp1) that took 0.5 seconds per episode to run. This algorithm run for 5 days.

This first approach was made by trying to keep it simple. This way, we ended up using the inputs and outputs already defined in the simulator provided by Commander Piotr and working our way based on that. It was a test to see if this type of algorithms was capable of controlling BUVs.

As shown in the Figure 12, in the top down view, the "fish"is swimming away from the goal, and in the altitude graphic its proven that the robot doesn't know how to try and maintain its altitude or try to submerge when passed its desired z value, it is just floating away at the moment. But in the last graphic of the figure 3 (accumulative reward), the robot is still doing very random movements trying every possible action, at

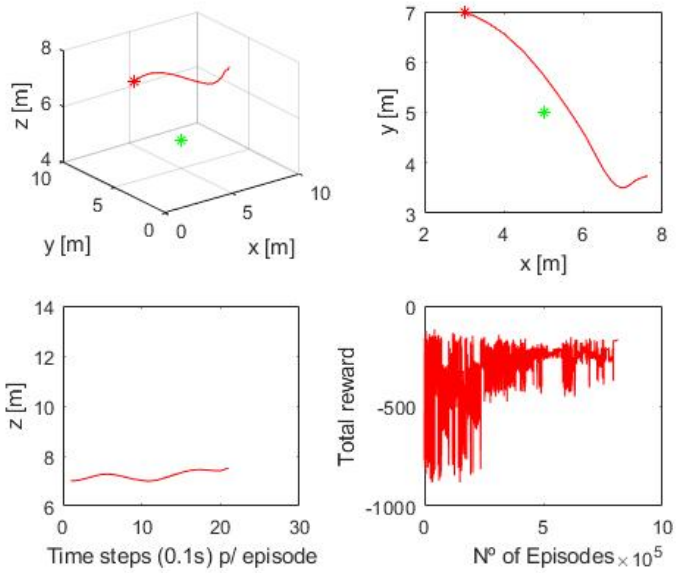


Fig. 13: 80000th episode

least once, for every state. The algorithm hasn't started "learning", because it doesn't have enough data to know which are the best actions, or the least worse actions, to take for a state.

In Figure 13, the "fish"is already trying to reach the goal as fast as possible in is top down view, however due to its momentum, it ends up passing it, realizing at the end of the episode that it isn't receiving the best reward that it could get at the moment, and if it keeps getting way of the goal it will only get worse, which can be proven by the almost 90 degree left turn at the end of the trajectory, that means that the robot is trying to return to goal. We can observe some good learning progresses as well in the altitude graphic, because it is already maintaining its altitude near the goal's.

The proof of success of this work is shown in the accumulative reward graphic, it is very evident that it is converging and reaching very satisfactory solutions.

With this results it is proven that is possible to use this type of algorithm to the control of the BUVs, but it is done in a very simple method. In the second implementation was used a different notation for the action variables that although is a more complex one to implement, it is more intuitive to Humans to understand: a sine wave.

5.2 Change of variables

Defining the the states by a 3D axis and the vehicle yaw compared to the goal doesn't take advantage of the space symmetry, for example, in a 2D axis if the goal was in the position x = 5 and y = 5, and the vehicle was in the state x = 3 and y = 4, by the previous state variables (and that the vehicle was always pointed at the goal) it would be a different state than x = 6 and y = 2, while in distance to the goal it would be same state, due to its symmetry.

To take advantage of the symmetry factor, lets define the new state variables as the discrete distance to the goal in 0 to 10 meters, and by the discrete relative yaw and pitch angles relative to the goal position.

$$S = \begin{bmatrix} -90 & -90 & 0 \\ -60 & -60 & 1 \\ -30 & -30 & 2 \\ -10 & -10 & 3 \\ -5 & -5 & 4 \\ 0 & 0 & 5 \\ 5 & 5 & 6 \\ 10 & 10 & 7 \\ 30 & 30 & 8 \\ 60 & 60 & 9 \\ 90 & 90 & 10 \end{bmatrix},$$

This way we were able to reduce to size of the state variables to 11^3 in comparison to the last definition of state variables.

Action variables defined in frequency and deviation are not intuitive for the human being to understand. So for an easy understanding of the output signal to the controllers, it was decided to transform frequency and deviation into a a waveform signal, i.e. amplitude, frequency and offset.

To apply this concept it was necessary to change the internal simulation code by removing MATLAB functions that were generating amplitude values based on frequency and deviation.

Being A defined by the following table:

750			1500			3000							**Frequency**
0			10			-10			30		-30		**Deviation**
0	2.5	5	0	2.5	5	0	2.5	5	0	2.5	0	2.5	**Amplitude**

Table 1: New action variables

In this table for each represented frequency we are able to select one deviation value that can assume two or three values of amplitude. We have now 39 different actions for each fin and tail, having a total of 39^3 sets of actions possible.

It was not necessary to utilize all the previous action variables options, for example, by having the option to select an amplitude of 0 it is redundant to choose a frequency of 0.

By applying this change we actually increase the number of action sets relative to the first definition of A, but since the number of states were drastically decreased, the total combination of the new states and actions variables ($11^3 \times 39^3$) is still lower than the first implementation ($10^4 \times 5^6$).

With a lower number of the state and action pair, the speed of convergence will increase. And by changing the simulation code, removing the *interp1* functions, that were one of the causes for long time necessary for the program to run, it will also contribute to the increase of the speed of convergence.

Where in the left and right fin oscillation, the left fin is represented by the green line and the right fin by the red one.

Although the new changes of variables are able to reach a satisfactory solution faster than the first approach, there is a problem with using a sine wave as an action input.

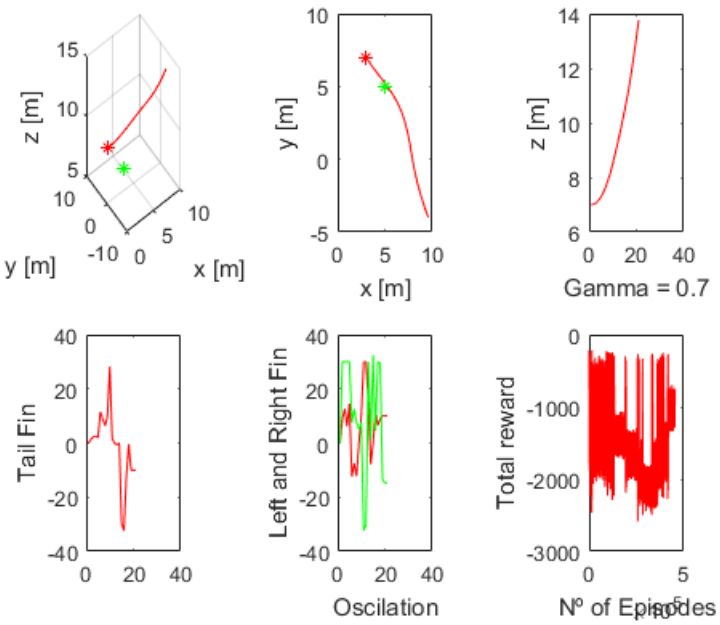


Fig. 14: Test with the new state and action variables

Between to different actions there might be a continuity problem, because the position where the tail or fins were left is not necessarily the point were they are meant to start in the next action, and can lead to situations illustrated in the figure below.

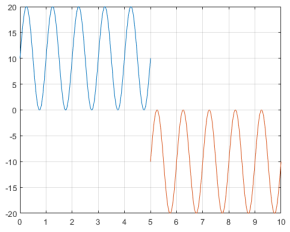


Fig. 15: Continuity problem

To try and solve this kind of problem we are to find a transitory state that can link the end position of one action to the starting position of the other. The solution that was brought to light, was using a sine wave with a fixed frequency (the maximum frequency value presented by Commander Piotr in his simulation, 3000 mHz), fast enough so that this state doesn't take too long, no off-set and an amplitude and capable of reaching of every possible position (an amplitude of 35, for example) that the fins and tails can take. Resulting the sine wave:

$$y = 35sin(2\pi 3) \tag{10}$$

By knowing the last point of the last action, it is possible to find the phase of the transitory sine wave that has the same value. After that the transitory sine wave will proceed normally until it value is the same as the starting value of the second action to take. An example is shown in the figure below:

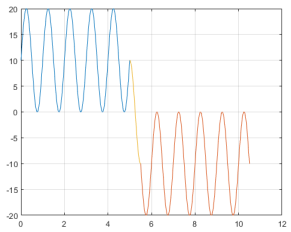


Fig. 16: Solution for the continuity problem

The change of variable was a success, it proved that after all the changes the algorithm still converges, making inputs more user friendly and it is easy to identify the behavior of the tail and fins. Although it is starting to converge at the same time as the previous variables, it is giving a lower reward to worst action-sate pairs, while taking advantage of the symmetry of the space, turning some positions relative to goal ambiguous.

5.3 Decision Parameters

In this subsection, the results of some tests will be shown in order to try and decide a base value for the open parameters described previously: actualization time, simulation time per episode, step-sized parameter, discounted factor and the probability of exploration.

The actualization time will have a preset value of 0.1 seconds. Although some test were realized it was not possible to obtain representative images, due to the total time required for the accumulative reward to start converging.

5.3.1 Step-sized parameter

The step-sized parameter or learning rate (α) is a value between $0 < \alpha \leq 1$, and as the name describes is the factor that determines the rate at which the functions learns.

In this tests we gave the values $\alpha = 0.5$ and $\alpha = 0.9$.

The results are as follows:

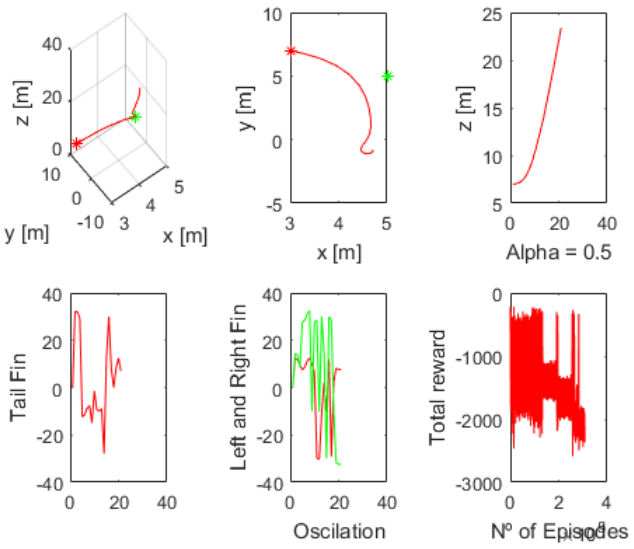


Fig. 17: Test with Alpha = 0.5

We can observe that in the beginning of each test the accumulative reward as a similar behavior, because it is the phase that the algorithm is testing every possible action at least once in each state to initialize the utility number of each action-state pair for future comparison.

We can see that $\alpha = 0.9$ is converging faster than $\alpha = 0.5$. The higher the step-sized parameter the higher negative value will be given to the worst actions, therefore it will take a longer time for the algorithm to visit them again.

This way at the end of the tests, the $\alpha = 0.9$ is already taking one of the best set of actions to achieve its goal while $\alpha = 0.5$ is still visiting some worst actions.

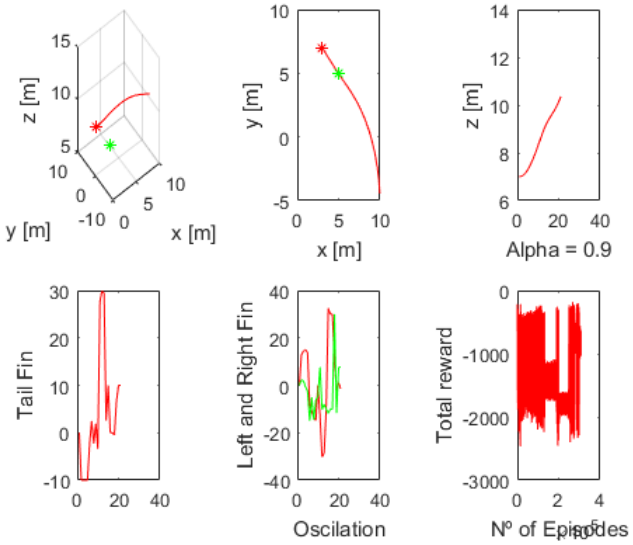


Fig. 18: Test with Alpha = 0.9

With this we have chosen the step-sized parameter $\alpha = 0.8$, although 0.9 is supposed to give better accumulative reward lets not forget that in the real environment the utility value of the action-state pairs have little fluctuations, so we must not take the first worst action has guaranteed to be one bad action forever.

5.3.2 Discounted factor parameter

The discount factor (γ) is a parameter, with a value between $0 < \gamma \leq 1$, which represents how much future events lose their value according to how far away in time they are.

A discount factor of 0 would mean that only the immediate rewards were taken into consideration. The higher your discount factor, the farther the rewards will propagate through time.

In this implementation we are only taking into consideration one action into the future and, for the tests, we gave the values $\gamma = 0.3$ and $\gamma = 0.7$.

The program run for a day and the results are as follows:

As said in the previous evaluation the accumulative reward starts by being similar, but at around 300000 episodes we can see that $\gamma = 0.7$, the one gives a higher factor

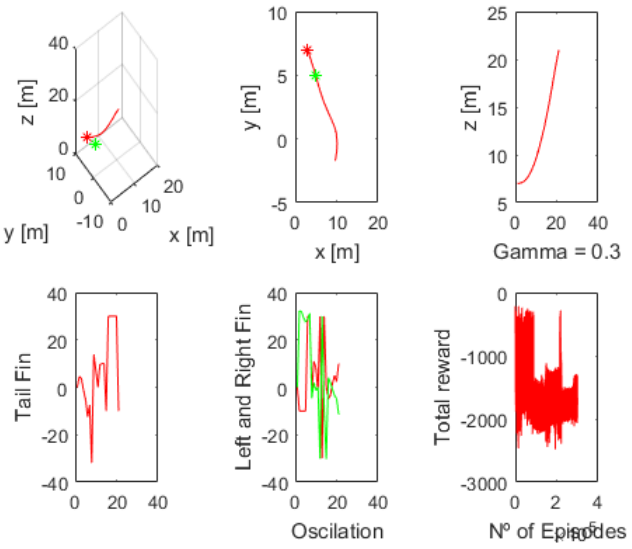


Fig. 19: Test with Gamma = 0.3

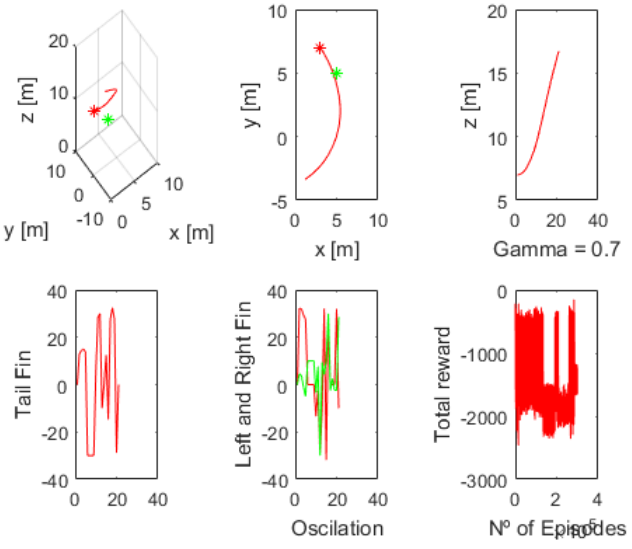


Fig. 20: Test with Gamma = 0.7

to future actions than $\gamma = 0.3$, is already starting to converge to a reasonable solution, but it is still a bit way from a satisfactory solution, while the one that gives an higher importance to the actions taken in the moment is still searching and updating in the low rewards (worst actions) of the action-state pairs.

Although $\gamma = 0.7$ presented better results, the water is not a stable environment so the future action may not have the utility that was predicted. In this case, we will give the same importance to the actions taken in moment and predicted actions in the future, be using the value $\gamma = 0.5$.

5.3.3 ϵ-greedy implementation

Is it the best to the agent always choose the action with the maximum return in a certain state? Is one of the questions that were made in Reinforcement Learning, and the answer is no (Sutton and Barto, 1998, Bertsekas and Tsitsiklis, 1996).

The studies done in this area prove that the algorithm takes a little bit longer to start converging but converges faster to the final return value and it has a better accumulative reward in the end, if it has a low probability of choosing a random action in each time step.

This introduces two new concepts: exploitation, when the agent chooses the action with the highest return value, and exploitation, when the agent picks the action randomly.

One of the great challenges of the Reinforcement Learning is to find a balance between exploration and exploitation.

In this ϵ-greedy implementation, where ϵ is the probability of choosing a random action, we will try the same algorithm with two different values of ϵ, 0.05 and 0.15.

The program run for three days and the results are the following:

As we it is possible to observe, no convergence of the accumulative reward started to occur in both situations. It was expected for the accumulative reward to take longer

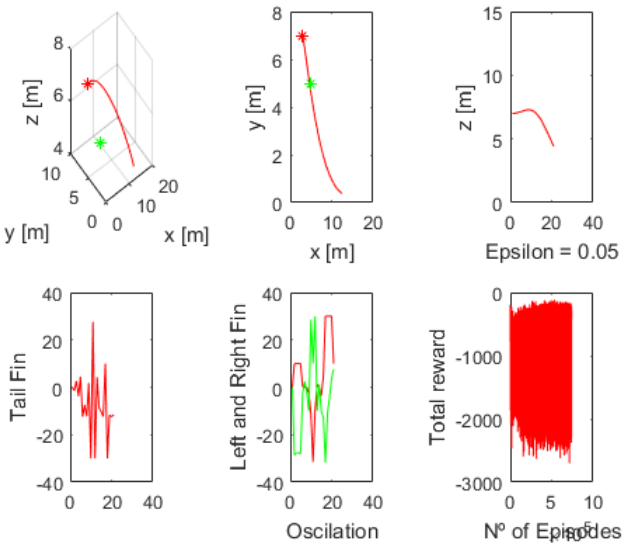


Fig. 21: Test with Epsilon = 0.05

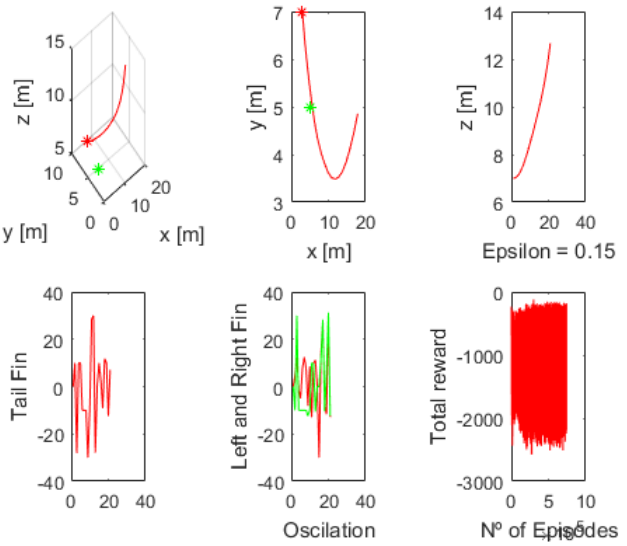


Fig. 22: Test with Epsilon = 0.15

to start converging at the beginning, but not reaching roughly 700000 episodes without doing so.

With this results, it was decided for the Q-learning algorithm to be completely greedy, leaving behind the exploration.

5.4 Final implementation

In this final implementation, we took into consideration the results in the previous step-sized, discounted factor and ϵ-greedy tests. Reaching the final values of $\alpha = 0.8$, $\gamma = 0.5$ and $\epsilon = 0.0$.

It was also taken into consideration that the vehicle will have antennas in the upper part of its model, so it is necessary for it to maintain a stable roll value. With this an extra roll column, in degrees, was added to the state variables and a penalty in terms of roll (θ) was added to the reward function.

Being the new S and R defined by:

$$S = \begin{bmatrix} -90 & -90 & -90 & 0 \\ -60 & -60 & -60 & 1 \\ -30 & -30 & -30 & 2 \\ -10 & -10 & -10 & 3 \\ -5 & -5 & -5 & 4 \\ 0 & 0 & 0 & 5 \\ 5 & 5 & 5 & 6 \\ 10 & 10 & 10 & 7 \\ 30 & 30 & 30 & 8 \\ 60 & 60 & 60 & 9 \\ 90 & 90 & 90 & 10 \end{bmatrix},$$

$$R = -|\theta|/11 * \sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2}^2 \tag{11}$$

For the final implementation, instead of having a fixed starting position, the starting position was generated randomly within five meters of the goal.

With this we made the program run for a whole week and the results are as followed:

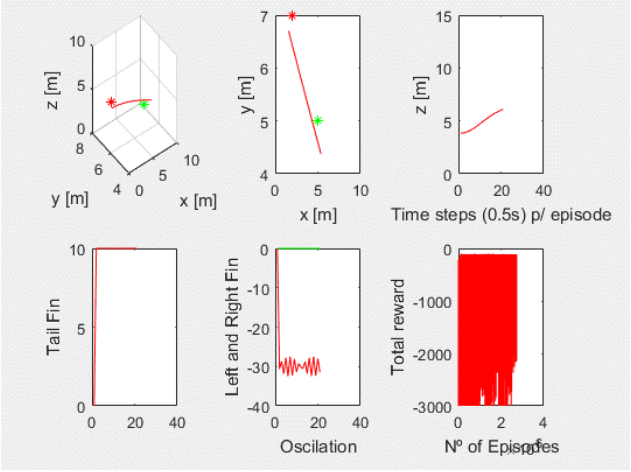


Fig. 23: Final results

For the final evaluation we can confirm by the trajectories (3D, top-down view and altitude) that the algorithm is reaching a very good solution of locomotion, heading directly to the goal but not so fast that it passes it, and its maintaining its altitude in the same one as the goal, providing with the best rewards, although it isn't very perceptible in the accumulative reward graphic, however we can observe that it started to converge in the end.

The only variable that is not possible to read here is the roll, which has a minor importance to reach the objective in the simulation, but it will be a great help in the real environment to keep the antennas straight up and available for communications.

With this final evaluation the vehicle fulfilled its objective of reaching the goal in 20 seconds, figuring out how to move its tail and fins to move in the direction of the goal, but in a cautious way in order to remain near it the longest time possible for a better reward, which is a really good solution.

6 Future work

Portugal joined the SABUVIS project one year after it started, due to some political problems. Therefore, the construction of the real model was also delayed and it isn't completed yet, which prevented from experimenting with the algorithm in the real vehicle and study its behavior.

One of the objectives of this work was to see how would the algorithm adapt from a simulation environment to a real one and trying to adapt to a tail with two degrees of liberty, which was not possible. Maybe, in a near future it will be.

And it is the last final step missing in the SABUVIS.

As we all know, the space is described by continuous variables. The discretization of the action and state variables is the biggest problem in the Q-learning algorithm, that can lead to some approximation errors. To surpass this downside, there will be a future study in different types of Reinforcement Learning algorithm based on policy improvement or function approximation, such as PI^2 or PoWER.

Another important aspect of this work is the time it takes to learn. By the experiences described in previous chapters, the algorithm took a considerable amount of time to start to converge to a satisfactory solution, from one to seven days depending on the number of states and actions.

In this work, the existence of obstacles was not taken into consideration. So if the vehicle finds some kind of wall between his starting position and his goal, if it was already trained in simulation without any kind of obstacle, it will take a long time to overcome it. To try and solve this problem, it may be necessary to train the algorithm to detect and surpass this kind of trials as well or , by simply detecting an obstacle generate a new goal in order to surpass it.

It is important to take into consideration that the vehicle will have to surface every once in a while, to communicate via Wi-Fi or satellite, to correct its position by GPS, after a long period of inertial navigation, or receive new orders to abort his mission or change the destination goal.

The algorithm didn't learn this type of functions that are established in the software of other UUV's made by LSTS.

Conclusion

With this work it was proved that is possible to control Biomimetic Underwater Vehicles with Reinforcement Learning algorithms.

In one hand, it has its limitations, like the long time it takes to learn in the simulation phase and in the real phase if an obstacle appears it will take a long time to overcome it, besides this situation will have an effect in its learning experience.

One the other hand, it is a much easier way to control biomimetic vehicles than by doing it remotely or by programming handmade controllers, due to the robot learning from itself, therefore there is no need to programming the controllers that are not intuitive.

The SABUVIS project is at its end due date and there is still some vehicle models to finish and to test the algorithm in the real environment, to see if there are any adjustments to be made.

This project had a lot of problems because the algorithm tests were too time consuming. For each parameter test the algorithm took one or more days to run and for the final implementation run it took a whole week, due to MATLAB heavy processing and the dimension of the state and action variables, all of tests had to be run separately. And with each time a bug was detected in the code, all of the tests had to be run all over again, leaving the state of this project idle during that time.

This work was an introduction to the world of robotics, an area that is gaining more and more influence in the daily life and had it is a subject that I want to study in the future.

It opened my vision to the scientific world, by publishing an article in the Institute of Physics (IOP) magazine and presenting it in the opening ceremony at the Seaconf realized in the Romanian Naval Academy.

References

Barron, T. D. (1998, May 5). *Apparatus for interconnecting an underwater vehicle and a free floating communications pod.* Google Patents. (US Patent 5,748,102)

Brown, C., & Clark, R. P. (2010). Using a novel vehicle conceptual design utility to evaluate a long-range, large payload uuv. In *Oceans 2010* (pp. 1–10).

Christ, R. D., & Wernli Sr, R. L. (2013). *The rov manual: a user guide for remotely operated vehicles.* Butterworth-Heinemann.

Evans, J., Smith, J., Martin, P., & Wong, Y. (1999). Beach and near-shore crawling uuv for oceanographic measurements. In *Oceans'99 mts/ieee. riding the crest into the 21st century* (Vol. 3, pp. 1300–1306).

Gonzalez-Gomez, J., Aguayo, E., & Boemo, E. (2005). Locomotion of a modular worm-like robot using a fpga-based embedded microblaze soft-processor. In *Climbing and walking robots* (pp. 869–878). Springer.

Gosavi, A. (2009). Reinforcement learning: A tutorial survey and recent advances. *INFORMS Journal on Computing*, *21*(2), 178–192.

Kim, H.-J., Song, S.-H., & Ahn, S.-H. (2012). A turtle-like swimming robot using a smart soft composite (ssc) structure. *Smart Materials and Structures*, *22*(1), 014007.

Lala, J. H., & Harper, R. E. (1994). Architectural principles for safety-critical real-time applications. *Proceedings of the IEEE*, *82*(1), 25–40.

Lepora, N. F., Mura, A., Krapp, H. G., Verschure, P. F., & Prescott, T. J. (2013). *Biomimetic and biohybrid systems: Second international conference, living machines 2013, london, uk, july 29–august 2, 2013, proceedings* (Vol. 8064). Springer.

Lepora, N. F., Verschure, P., & Prescott, T. J. (2013). The state of the art in biomimetics. *Bioinspiration & biomimetics*, *8*(1), 013001.

Miller, D. P. (1996). Design of a small, cheap uuv for under-ship inspection and salvage. In *Autonomous underwater vehicle technology, 1996. auv'96., proceedings of the 1996 symposium on* (pp. 18–20).

Muljowidodo, K., Adi, N., Budiyono, A., Prayogo, N., et al. (2009). Design of shrimp rov for surveillance and mine sweeper.

Nelson, G. M., & Quinn, R. D. (1999). Posture control of a cockroach-like robot. *IEEE Control Systems*, *19*(2), 9–14.

Nicolai, L. M. (2002, June 25). *Anti-submarine warfare uav and method of use thereof.* Google Patents. (US Patent 6,409,122)

Paull, L., Saeedi, S., Seto, M., & Li, H. (2014). Auv navigation and localization: A review. *IEEE Journal of Oceanic Engineering*, *39*(1), 131–149.

Pinto, J., Dias, P. S., Martins, R., Fortuna, J., Marques, E., & Sousa, J. (2013). The lsts toolchain for networked vehicle systems. In *Oceans-bergen, 2013 mts/ieee* (pp. 1–9).

Puterman, M. L. (2014). *Markov decision processes: discrete stochastic dynamic programming.* John Wiley & Sons.

Shchukin, D. G., Sukhorukov, G. B., Price, R. R., & Lvov, Y. M. (2005). Halloysite nanotubes as biomimetic nanoreactors. *Small*, *1*(5), 510–513.

Shugen. (2001). Analysis of creeping locomotion of a snake-like robot. *Advanced Robotics*, *15*(2), 205–224.

Siciliano, B., & Khatib, O. (2016). *Springer handbook of robotics.* Springer.

Silva, H. V. d. (2017). *Desenvolvimento de um sistema de lançamento e recolha dos auvs seacon através dos submarinos da classe tridente* (Unpublished doctoral dissertation).

Siochi, E. J., Anders Jr, J. B., Cox, D. E., Jegley, D. C., Fox, R. L., & Katzberg, S. J. (2002). Biomimetics for nasa langley research center: year 2000 report of findings from a six-month survey.

Stokey, R., Austin, T., Allen, B., Forrester, N., Gifford, E., Goldsborough, R., ... von Alt, C. (2001). Very shallow water mine countermeasures using the remus auv: a practical approach yielding accurate results. In *Oceans, 2001. mts/ieee conference and exhibition* (Vol. 1, pp. 149–156).

Sutton, R. S., Barto, A. G., et al. (1998). *Reinforcement learning: An introduction.* MIT press.

Szymak, P. (2016). Mathematical model of underwater vehicle with undulating pro-

pulsion. In *Mathematics and computers in sciences and in industry (mcsi), 2016 third international conference on* (pp. 269–274).

Szymak, P., Morawski, M., & Malec, M. (2012). Conception of research on bionic underwater vehicle with undulating propulsion. In *Solid state phenomena* (Vol. 180, pp. 160–167).

Thompson, R. B. (2005). *Fluorescence sensors and biosensors.* CRC Press.

Yuh, J. (2000). Design and control of autonomous underwater robots: A survey. *Autonomous Robots*, *8*(1), 7–24.

Zhang, S., Marini, D. M., Hwang, W., & Santoso, S. (2002). Design of nanostructured biological materials through self-assembly of peptides and proteins. *Current opinion in chemical biology*, *6*(6), 865–871.

Annexes

Annexe A - Submitted Article for the IOP magazine

Annexe B - Q-learning MATLAB code

Reinforcement Learning: *The Application to Autonomous Biomimetic Underwater Vehicles Control*

J Magalhães[1], B Damas[1] and V Lobo[1]

[1] CINAV, Portuguese Navy Research Center, Almada, Portugal

E-mail: candeias.magalhaes@marinha.pt

Abstract. The Autonomous Biomimetic Vehicles have been increasing in popularity in the past few years. Controlling such type of vehicles is not trivial: due to its complex dynamics and kinematics, it is complex to analytically derive controllers that can efficiently perform a given task, such as reaching a given position target in a minimum time. In this paper we will evaluate the results of the implementation of a reinforcement algorithm in autonomous biomimetic underwater vehicles, providing a new way to control this type of vehicles in which the algorithm is in constant learning.

1. Introduction and motivation

Biomimetics is a field where various principles are applied to mimic biological processes, such as the humanoids robotics field where human-like robots are developed [1]. In the context of underwater vehicles the kinematics and dynamics of the vehicles try to emulate the motion of different sea creatures.

The study of autonomous Biomimetic Underwater Vehicles (BUV) have been increasing in the past few years in a military context due to its furtive locomotion capabilities: they have the ability to camouflage within the environment and a silent undulating propulsion that generates a very different acoustic signature when compared to conventional Unmanned Underwater Vehicles (UUV) [2].

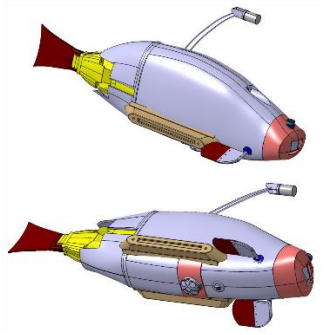


Figure 1. 3D model of fish-like vehicle

This work is part of the international project "Swarm of Biomimetic Underwater Vehicles for Underwater Intelligence, Surveillance and Reconnaissance (ISR)" (SABUVIS), whose main objective is to use BUVs in missions for stealth data collection and surveillance, where Portugal, Poland and Germany and the countries part of the consortium. Where the Portugal collaborators are Escola Naval, LSTS and OceanScan.

Under this project two different vehicles are being built, one mimicking a fish (Figure 1) and the other replicating a seal. Both vehicles have the same basics characteristics, similar form, two

side fins and a tail, the major differences being their size and tail. The seal-like tail is composed by two smaller tails that mirror the movement of each other, trying to mimic a breaststroke or “frog” stroke style of swimming, while the fish-like tail is a simple tail sectioned in two parts to give more fluidity.

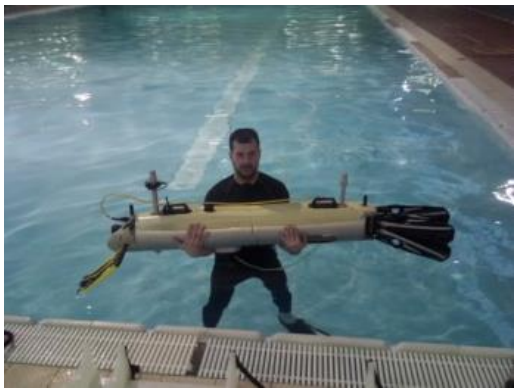


Figure 2. Seal-like vehicle during tests

Under this project the generated acoustic noise developed by the BUVs, as well as their autonomy capabilities, will be measured and compared to conventional UUVs.

These different mechanical structures that are inspired in biological systems present a much more complex kinematic structure that makes the task of controlling the motion of the vehicle a non trivial one. For a fish-like BUV [3], for example, it is not straightforward to develop controllers that actuate the fins and the tail of the vehicle in order to make it follow a desired trajectory in an efficient way [4][5].

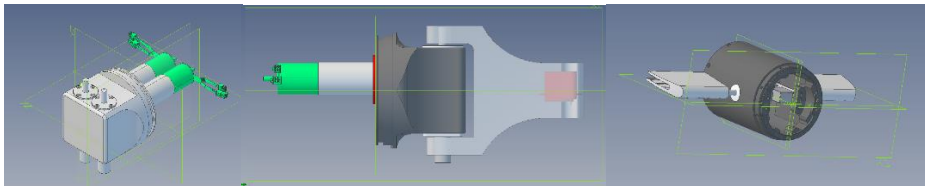


Figure 3. Model of the seal-like tail, fish-like tail and fins models, respectively

The purpose of this work is to develop adaptive controllers for these kind of vehicles based on Reinforcement Learning (RL) techniques. Reinforcement learning is an Artificial Intelligence technique, also biologically inspired, that revolves around "trial and error” theory. In RL an agent is supposed to choose its actions in a way to maximize an external reward that it gets from its interaction with the environment: a positive reward is given when it fulfills certain conditions defined by the programmer, and a lower reward is obtained when that condition is not met. This reward is tightly linked to the desired behaviour for the agent and should be chosen carefully as the agent, or robot, will solely learn based on the rewards it gets. For instance, when learning to navigate from a position to other the agent should get higher value rewards as it gets closer to the desired position.

To learn to perform a given task the agent has an internal policy that maps states to actions. Given the history of visited states, associated performed actions and received rewards the agent continuously adapts its policy in order to maximize its expected accumulated reward in the long run, this way progressively starting to exploit the actions that get the higher positive rewards, until it reaches the best way to perform the task.

There is no need to worry about programming every action to perform in every possible state when using a RL scheme, as these actions are learned from trial and error; it is even possible to the algorithm to adapt to unexpected situations, because the agent will teach itself in a continuous process of adaptation. This is a major advantage of using RL schemes when compared to traditional controllers, where a good knowledge of the process to control is typically needed.

2. Reinforcement learning in underwater biomimetic vehicles control

To implement a RL controller in the BUVs the work will be divided in two different stages. In the first stage a simulator will be developed and the RL algorithms will be tested and applied in a simulated environment. As the BUV will learn how to control its movements from scratch, the learning phase will start in such a simulated environment to prevent actuator wear, collisions and other damage to the physical robot that would inevitable result from the initial exploratory random movements it would perform in this phase. Afterwards, when the simulated robot can already control its movement in a satisfactory way, the learned model will be transferred to the real vehicle, where it will undergo a second learning phase. Such adaptation is required to take into account the differences between the simulated and the real environment.

2.1. BUV Q-learning simulation

Q-Learning is a Reinforcement Learning algorithm that estimates, from its interaction with the environment, the utility of performing a given action in a particular state, given by Q(S, A) [6]. This policy can be learned from this interaction, without the need to have a model for this environment, using the update rule.

$$Q(S,A) = Q(S,A) + \alpha[R + \gamma max_a Q(S',a) - Q(S,A)], \qquad (1)$$

where Q denotes the expected accumulated future reward obtained if action A is performed in state S. S' is the observed next state after executing action A and R the corresponding reward. $0 < \alpha \leq 1$ is a step-size parameter and $0 < \gamma \leq 1$ denotes the discount-rate factor: the lower this value the lesser the importance given to distant future rewards, *i.e*, the more myopic the agent is regarding future rewards.

The vehicle in training is a fish-like composed by two lateral fins, one on each side, and one tail. Each motor is actuated by a sinusoidal signal with variable frequency (F) [mHz] and deflection (K), the average value in which the tail or fins cycle around, where F and K denote the tail variables, F1 and K1 the left fin and F2 and K2 for right fin. Together these 6 controlled variables define the action vector [7].

The tabular nature of the vanilla Q-Learning algorithm requires a discretized set of actions and states. With respect to actions we define A as the variables whose columns contain the discretized action values for each component of the action vector

$$\mathrm{A} = \begin{bmatrix} 3000 & 500 & 3000 & 500 & 3000 & 500 \\ 2250 & 250 & 2250 & 250 & 2250 & 250 \\ 1500 & 0 & 1500 & 0 & 1500 & 0 \\ 750 & -250 & 750 & -250 & 750 & -250 \\ 0 & -500 & 0 & -500 & 0 & -500 \end{bmatrix},$$

where each column represents F, K, F1, K1, F2 and K2 respectively and where we set F = F1 = F2, in steps of 750, and K = K1 = K2, in steps of 250. This way, we have a set of 5^6 different actions from where to choose at each iteration.

We consider the state to be the position in each axis, in meters, and the vehicle angle in the xy plane (in radians). The state variables S, that contain that discrete values, is then defined as

$$S = \begin{bmatrix} 1 & 1 & 1 & 0.6 \\ 2 & 2 & 2 & 1.2 \\ \vdots & \vdots & \vdots & \vdots \\ 9 & 9 & 9 & 5.4 \\ 10 & 10 & 10 & 6 \end{bmatrix},$$

where each column contains the possible values for x, y, z, in steps of 1, and ψ, in steps of 0.6, respectively. This makes 10^4 distinct states.

The objective of this simulation is to reach a goal point and stay there. To fulfill it, the reward function is defined as

$$R = -\sqrt{\Delta x^2 + \Delta y^2 + \Delta z^2},$$

so that the reward will be higher the closest it gets to the goal.

The reward had to be made negative due to Q, that is initialized with zeros, selecting the action with the maximum value to the present and future state. By making it negative, it is guaranteed that each action is selected at least one time for each state, in the infinite number of episodes.

The simulation is divided by episodes. Each episode starts the vehicle in a random position with a 4 meter distance to goal and with the vehicle pointed at it, each episode it has 200 iterations, each one representing 0.1 seconds, and it end when 20 seconds have passed, starting a new episode.

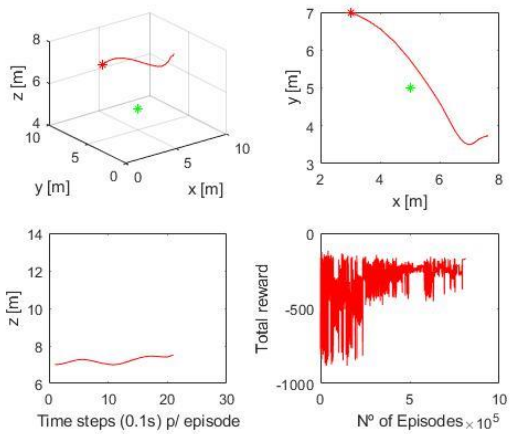


Figure 4. Graphics of early results of the Q-learning algorithm

The graphics above show one of the trajectories of the vehicle, in a 3D dimension and 2D x0y axis, its altitude during the episode and its accumulative reward in the past episodes. The red star being the starting position and the green star the goal.

It is noted that the accumulative reward has already started to converge, which means that is starting to reach a satisfactory solution, but it is taking a long time to do it.

2.2. LSTS

The Laboratório de Sistemas e Tecnologia Subaquática (LSTS) is an interdisciplinary research laboratory part of Faculdade de Engenharia da Universidade do Porto (FEUP) that specializes on the design, construction and operation of unmanned underwater, surface and air vehicles and on the development of tools and technologies for the deployment of networked vehicle systems.

LSTS created the software toolchain Neptus-IMC-DUNE that will be present in the project SABUVIS. DUNE is the system for vehicle on-board software, it has a C++ programming environment and it is responsible for navigation, code for control and access to sensors and actuators; IMC is the communication protocol; Neptus is a command, control, communications and intelligence framework for operations with vehicles, systems and human operators.

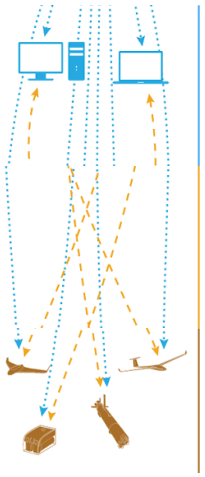


Figure 5. LSTS toolchain, Neptus-IMC-DUNE

In the project SABUVIS, LSTS is responsible for the integration of software and sensors on the vehicle.

3. Conclusions and future work

The convergence of the Q-learning algorithm shows us that we are headed in the right direction with these results, we can already see that the simulation is trying to reach the goal as soon as possible, also it is already starting to learn to turn back and try to return to the goal to obtain a better reward. There are still some problems like needing an adjust in the state variables to decrease the convergence time and the main problem being the discretization of the values. A different approach may resort to function approximation or algorithms based on policy improvement {*e.g.*, PI^2 or POWER), in order to circumvent the discretization of the state and action space. We also expect to implement the algorithm in the physical BUV, using the software toolchain that runs on the vehicle developed by FEUP.

References

[1] BAR-COHEN, Yoseph, "Biomometics: Biological Inspired Technologies", California, Jet Propulsion Laboratory (JPL), California Institute of Technology, 2006

[2] LISTEWNIK, K., "Sound Silencing Problem of Underwater Vehicles", in Solid State Phenomena, Vol. 196, Trans Tech Publications, 2013, pp. 212-219

[3] LOW, K. H., "Modelling and parametric study of modular undulating fin rays for fish robots", in Mechanism and Machine Theory, Vol. 44, 2009, pp. 615–632

[4] COLGATE, J. E. and LYNCH, K. M., "Mechanics and Control of Swimming: A Review", in IEEE Journal of Oceanic Engineering, Vol. 29, No. 3, 2004, pp. 660-673

[5] MALEC, M. et al., "Analysis of Parameters of Traveling Wave Impact on the Speed of Biomimetic Underwater Vehicle", in Solid State Phenomena, Vol. 210, Trans Tech Publications, 2014, pp. 273-279

[6] BARTO, Andrew G. and SUTTON, Richard S., Reinforcement Learning: An introduction, 1st edition, Cambridge, MIT, 1998

[7] SZYMAK Piotr, "Mathematical model of underwater vehicle with undulating propulsion", Poland, Polish Naval Academy, Institute of Electrical Engineering and Automatics

Annexe B - Q-learning MATLAB code

```
function tau = Fins(ogon_czest,ogon_kat,OX_ampl,p_lewa_czest,p_lewa_kat,
    PLX_ampl,p_prawa_czest,p_prawa_kat,PPX_ampl,i,dt,Vx);

%measured rotation velocity for tail fin [Hz]
ocz_pom=[3000
         1500
         1200
         900
         600
         300
         0];
%measured average thrust of tail fin [N]
oxsr_pom=[18
       12.8
            6.4
            2.4
            1.2
            0.4
            0];
%measured amplitude of tail fin oscillation [N]
oxampl_pom=[14.2
       8.2
            6.2
            3.2
            1.2
            0.6
            0];

%measured rotation velocity for side fins [Hz]
pcz_pom=[3000
         1500
         1200
         900
         600
         300
         0];
```

```
36
37 %measured average thrust of side fin [N]
38 pxsr_pom=[8
39         6.2
40             3.2
41             1.2
42             0.5
43             0.15
44             0];
45 %measured amplitude of side fins oscillation [N]
46 pxampl_pom=[5.2
47         3.2
48             2.2
49             1.2
50             0.4
51             0.2
52             0];
53
54 %constants
55 przekladnia_ogon = 12; przekladnia_pletwa = 12;
56 ogoncz = ogon_czest/60/przekladnia_ogon;
57 prawacz = p_prawa_czest/60/przekladnia_pletwa;
58 lewacz = p_lewa_czest/60/przekladnia_pletwa;
59 r1 = 0.1; r2 = 0.4; %distances from origin to ... etc
60 r3 = 0.3; r4 = 0.13;
61 xG=0;
62 %thrust generated by side and tail fins
63 OX_sr = interp1(ocz_pom,oxsr_pom,ogon_czest,'linear');
64 %OX_ampl = interp1(ocz_pom,oxampl_pom,ogon_czest,'linear');
65 OX = OX_sr + OX_ampl * sind(i/dt*ogoncz);
66 PLX_sr = interp1(pcz_pom,pxsr_pom,p_lewa_czest,'linear');
67 %PLX_ampl = interp1(pcz_pom,pxampl_pom,p_lewa_czest,'linear');
68 PLX = PLX_sr + PLX_ampl * sind(i/dt*lewacz);
69 PPX_sr = interp1(pcz_pom,pxsr_pom,p_prawa_czest,'linear');
70 %PPX_ampl = interp1(pcz_pom,pxampl_pom,p_prawa_czest,'linear');
71 PPX = PPX_sr + PPX_ampl * sind(i/dt*prawacz);
72 %X, Y, N generated by tail fin
```

```
73  ogonX = OX * cosd(ogon_kat);
74  ogonY = OX * sind(ogon_kat);
75  promien1 = sqrt(((r1+(r2*cosd(ogon_kat)))^2)+((sind(ogon_kat)*r2)^2));
76  beta = atand((sind(ogon_kat)*r2)/(r1+(cosd(ogon_kat)*r2)));
77  ogonN = promien1 * ogonY * cosd(beta) * (-1);
78  %X, Z, N, M generated by side fins
79  pletwaLX = PLX * cosd(p_lewa_kat);
80  pletwaLZ = PLX * sind(p_lewa_kat);
81  pletwaLN = r4 * pletwaLX;
82  pletwaLM = r3 * pletwaLZ * (-1);
83  pletwaPX = PPX * cosd(p_prawa_kat);
84  pletwaPZ = PPX * sind(p_prawa_kat);
85  pletwaPN = r4 * pletwaPX;
86  pletwaPM = r3 * pletwaPZ * (-1);
87  %side fin as a rudder
88  xvx=0.014; xvxx=0.91;
89  Opor_pletwa = 0; D = 0;
90  beta = 90-(atand(r4/r3));
91  r = sqrt((r3^2)+(r4^2));
92  if ((p_prawa_czest==0)&&(p_prawa_kat==90)),
93      D = xvx + (xvxx*abs(Vx));
94      Opor_pletwa = cosd(beta)*D*r;
95  end %right side fin
96  if ((p_lewa_czest==0)&&(p_lewa_kat==90)),
97      D = xvx + (xvxx*abs(Vx));
98      Opor_pletwa = cosd(beta)*D*r*(-1);
99  end %left side fin
100
101 tau=[(ogonX+pletwaLX+pletwaPX-D),ogonY,(pletwaLZ+pletwaPZ),0,(pletwaLM+
        pletwaPM),(ogonN-pletwaPN+pletwaLN-Opor_pletwa)]'; %
```

```
1  function ni = Integration(nider, ni0, dt)
2
3  ni = ni0 + dt*nider;
```

```
1  function nider = Dynamics(ni, tau, dt)
2
3     u = ni(1);
```

```
v = ni(2);
w = ni(3);
p = ni(4);
q = ni(5);
r = ni(6);
x = ni(7);
y = ni(8);
z = ni(9);
phi = ni(10);
theta =ni(11);
psi =ni(12);
vel=[u v w p q r]';

m = 45; Ix = 0.73; Iy = 7.72; Iz = 7.72;
Xup = 4.5; Yup = 59; Zup = 59; Kup = 0; Mup = 11.2; Nup = 11.2;
MRB = [
       m                0                0                0                0
                                0
       0                m                0                0                0
                                0
       0                0                m                0                0
                                0
       0                0                0                Ix               0
                                0
       0                0                0                0
           Iy                    0
       0                0                0                0                0
                                Iz];
MA = [
       Xup      0                0                0                0
                        0
       0                Yup      0                0                0
                        0
       0                0                Zup      0                0
                        0
       0                0                0                Kup      0
                        0
```

```
0 0 0 0 Mup 0
0 0 0 0 0 Nup];
M = MRB + MA;
invM = inv(M);

Xu = 0.0894; Xuu = 5.72; Yv = 1.9; Yvv = 18.58; Zw = 1.9; Zww = 18.58;
Kp = 0; Kpp = 0; Mq = 0.8; Mqq = 11; Nr = 0.7; Nrr = 10;
D11=Xu+(Xuu*abs(u));
D22=Yv+(Yvv*abs(v));
D33=Zw+(Zww*abs(w));
D44=Kp+(Kpp*abs(p));
D55=Mq+(Mqq*abs(q));
D66=Nr+(Nrr*abs(r));
D = [
    D11 0 0 0 0 0
    0 D22 0 0 0 0
    0 0 D33 0 0 0
    0 0 0 D44 0 0
    0 0 0 0 D55 0
    0 0 0 0 0 D66];

x0 = 0; y0 = 0; z0 = 0; P = 450; B = 460;
xB = 0; yB = 0; zB = 0; xG = 0; yG = 0; zG = 0;
G = [
        (P-B)*sin(theta)
        (P-B)*cos(theta)*sin(phi)
        -(P-B)*cos(theta)*cos(phi)
         0
        -(zG*P+zB*B)*sin(theta)*cos(phi) + (xG*P+xB*B)*cos(theta)
```

```
                    *cos(phi)
60              0];
61
62    nider = invM*(tau+G-D*vel);
```

```
1   function vel_global = Transformation(vel_local, ni)
2   u = ni(1);
3   v = ni(2);
4   w = ni(3);
5   p = ni(4);
6   q = ni(5);
7   r = ni(6);
8   phi = ni(10);
9   theta =ni(11);
10  psi =ni(12);
11
12  c1=cos(phi);              s1=sin(phi);
13  c2=cos(theta);            s2=sin(theta);          t2=tan(theta);
14  c3=cos(psi);              s3=sin(psi);
15
16  LG=[
17      c3*c2                 c3*s2*s1-s3*c1          s3*s1+c3*c1*s2          0
                                    0                       0
18      s3*c2                 c3*c1+s3*s1*s2          s2*s3*c1-c3*s1          0
                                    0                       0
19      -s2           c2*s1                                   c2*c1
                          0                   0                           0
20      0                     0                                             0
                                          1                   s1*t2
        c1*t2
21      0                     0                                             0
                                          0                   c1
                          -s1
22      0                     0                                             0
                                          0                   s1/c2
                  c1/c2];
23
24  vel_global=LG*vel_local;
```

```
function ni = SIM_BUV(ni,F,K,A,F1,K1,A1,F2,K2,A2)

warning off;

rad = 180/pi;
T = 0.5;                              %simulation time
%InitData=Initial;
T_global = 0;
dt = 1/18;
%ni = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];      %state vector
                                     %ni=[u v w p q r x y z phi theta psi]
nider = [0; 0; 0; 0; 0; 0];                %derivatives of state vector
tau = [0; 0; 0; 0; 0; 0];                  %vector of forces and moments tau
    = [X Y Z K M N]
N = ceil(T/dt);                                  %number of simulation
   steps
Vp = 0.0;                            %sea current [m/s]
AlfaP = 0 * pi/180;                  % [deg]

%simulation
for i=1:(N+1)
    %parameters for fins oscillation
    %F-frequency of tail fin oscillation, K-tail fin deflection
    %F1-frequency of left side fin oscillation, K1-left side fin
        deflection
    %F2-frequency of right side fin oscillation, K2-right side fin
        deflection
    %F = 1000; K = 0; F1 = 1000; K1 = 0; F2 = 1000; K2 = 0;

    tau = Fins(F,K,A,F1,K1,A1,F2,K2,A2,i,dt,ni(1));
    nider = Dynamics(ni,tau,dt);
    vel_0 = ni(1:6);
    vel_local = Integration(nider, vel_0, dt);
    %sea current
    uP = Vp*cos(AlfaP-ni(12)-pi);
    vP = Vp*sin(AlfaP-ni(12)-pi);
    vel_local(1) = vel_local(1)+uP;
```

```
    vel_local(2) = vel_local(2)+vP;
    vel_global = Transformation(vel_local, ni);                    %velocity
        transformation to global coordinate system
    coord_0 = ni(7:12);
    coord_global = Integration(vel_global, coord_0, dt);
    ni=[        vel_local

                coord_global];
    T_global=T_global+dt;
    %data for figures
    %W(i,:) = [time    u     w     x      y     z      theta       psi
                 X     Y      Z      N    ];
    W(i,:) = [(i*dt),ni(1),ni(3),ni(7),ni(8),ni(9),(ni(11)*rad),(ni(12)*
        rad),tau(1),tau(2),tau(3),tau(6)];
end

%save 'data.txt' W -ascii;  %data for figures
%BUV = figure;              %plot of figures
%subplot(3,2,[1 3]); plot(W(:,4),W(:,6),'r'); xlabel('x [m]');  ylabel('z
    [m]'); grid;
%subplot(3,2,2); plot3(W(:,4),W(:,5),W(:,6),'r'); xlabel('x [m]');
    ylabel('y [m]'); zlabel('z [m]'); grid;
%subplot(3,2,5); plot(W(:,1),W(:,2),'g', W(:,1),W(:,3),'b');  xlabel('t [
    s]'); ylabel('u, w [m/s]'); grid; legend('u','w');
%subplot(3,2,4); plot(W(:,1),W(:,7),'r', W(:,1),W(:,8),'c'); xlabel('t [s
    ]'); ylabel('\theta, \psi [deg]'); grid; legend('\theta', '\psi');
%subplot(3,2,6); plot(W(:,1),W(:,9),'m', W(:,1),W(:,12),'k'); xlabel('t [
    s]'); ylabel('X, N [N]'); grid; legend('X', 'N');
```

```
clc; close all;
clear;

% state matrix
%S  = [1:10;1:10;1:10;0.6:0.6:6];

% state variables
S = [0:10; -90 -60 -30 -10 -5 0 5 10 30 60 90; -90 -60 -30 -10 -5 0 5 10
    30 60 90; -90 -60 -30 -10 -5 0 5 10 30 60 90];
```

```
9
10 vpsi = [0.6:0.6:6];
11
12 % action matrix
13 %A =
       [0:750:3000;-500:250:500;0:750:3000;-500:250:500;0:750:3000;-500:250:500];
14
15 % action variables
16 Aa = [750 1500 3000];
17 Ab = [0 0 0 10 10 10 -10 -10 -10 30 30 -30 -30];
18 Ac = [0 2.5 5 0 2.5 5 0 2.5 5 0 2.5 0 2.5];
19
20 % build a state action matrix by finding all valid states
21 %Q = zeros(10000,15625);
22
23 %new Q
24 Q = zeros(1331,59319);
25
26 %ni= [u v w p q r x y z phi theta psi]
27     ni = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0];  %state vector
28
29 % learning rate settings
30 alpha = 0.8;
31 gamma = 0.5;
32
33 GOAL = [5,5,5];
34 xg = 5;
35 yg = 5;
36 zg = 5;
37 NUM_ITERATIONS = 100000;
38 T = 0;
39 flag = 0;
40 i = 1;
41 rad = 180/pi;
42
43 %for i=1:NUM_ITERATIONS
```

```
44 while flag == 0
45
46     %ni= [u  v  w  p  q  r  x  y  z phi theta psi]
47     ni = [0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0; 0]; %state vector
48
49     % starting position
50     X = randn(3,1);
51     Y = X / sqrt(X'*X) * 4;
52     ni(7:9) = Y + 5;
53     x = round(ni(7)); sx = x;
54     y = round(ni(8)); sy = y;
55     z = round(ni(9)); sz = z;
56     T = 0;
57     r = 0;
58
59
60 x = 3; sx = 3; ni(7) = 3;
61 y = 7; sy = 7; ni(8) = 7;
62 z = 7; sz = 7; ni(9) = 7;
63
64     j = 1;
65     W(j,1) = ni(7);
66     W(j,2) = ni(8);
67     W(j,3) = ni(9);
68     Roll = ni(10);
69     psi = ni(12);
70     W(j,4) = 0;
71     W(j,5) = 0;
72     W(j,6) = 0;
73
74     D = sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2);
75
76
77     % start pointed to goal
78     if x - xg > 0
79         ni(12) = pi + atan((y-yg)/(x-xg));
80     elseif x - xg < 0
```

```
81          ni(12) = atan((y-yg)/(x - xg));
82      elseif x - xg == 0 && y - yg > 0
83          ni(12) = 3*pi/4;
84      else
85          ni(12) = pi/2;
86      end
87  %      psi = ni(12);
88  %%      ni(12) = - pi/4; %rad for 135 degrees
89  %
90  %          dpsi = abs(vpsi - ni(12));
91  %          minpsi = min(dpsi);
92  %          for a = 1:10
93  %              if dpsi(a) == minpsi psi = vpsi(a); end
94  %          end
95  %          psi = round(psi/0.6);
96
97      % transform values to indices
98      % D
99      D = sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2);
100     iD = round(D);
101     if iD > 10
102         iD = 10;
103     end
104
105     %PSI
106     dx = x - xg;
107     dy = y - yg;
108     d = sqrt(dx^2 + dy^2);
109     if dx <= 0 && dy <= 0
110         psig = asin(abs(dx)/d);
111     elseif dx <= 0 && dy > 0
112         psig = asin(abs(dy)/d) + pi/2;
113     elseif dx > 0 && dy > 0
114         psig = asin(abs(dx)/d) + pi;
115     else
116         psig = asin(abs(dy)/d) + 3*pi/2;
117     end
```

```
118     Az = psig - ni(12);
119     mAz = [-90 -60 -30 -10 -5 0 5 10 30 60 90];
120     [Az, iAz] = min(abs(mAz-Az*rad));
121
122     %PITCH
123     dz = z - zg;
124     if dz >= 0
125         thetag = - asin(abs(dz)/D);
126     else
127         thetag = asin(abs(dz)/D);
128     end
129     Pch = thetag - ni(11);
130     mPch = [-90 -60 -30 -10 -5 0 5 10 30 60 90];
131     [Pch, iPch] = min(abs(mPch-Pch*rad));
132
133     %ROLL
134     mRoll = [-90 -60 -30 -10 -5 0 5 10 30 60 90];
135     [Roll, iRoll] = min(abs(mRoll-Roll*rad));
136
137     % transform matriz x, y, z
138     %iQ = (x + y * 10 + z * 10*10 + psi*10*10*10) - 1110;
139
140     % new transformation matriz D, Az, Pch, ?Roll?
141     iQ = (iD + iAz * 11 + iPch *11*11) - 132;
142
143     status = [x,y,z];
144     countActions = 0;
145
146     while T < 10
147         % record the current state of the robot for use later
148 %         prvx = x; prvy = y; prvz = z; prvpsi = psi;
149         prvD = D; prvyAz = Az; prvPch = Pch; prvRoll = Roll;
150         prviQ = iQ;
151
152         % select an action value
153         % which has the maximum value of Q in it
154         % if more than one actions has same value then select randomly
```

```
            from them
155         [val, index] = max(Q(iQ,:));
156         [xx,yy] = find(Q(iQ,:) == val);
157         if size(yy,1) > 1
158             index = 1+round(rand*(size(yy,1)-1));
159             action = yy(index,1);
160         else
161             action = index;
162         end
163
164 %         % transform actions
165 %         d = action + 3905;
166 %         A1 = rem(d, 5);
167 %         if A1 == 0
168 %             A1 = 5;
169 %         end
170 %         d = d - A1; F = A(1,A1);
171 %         A2 = rem(d, 25)/5;
172 %         if A2 == 0
173 %             A2 = 5;
174 %         end
175 %         d = d - A2*5; K = A(2,A2);
176 %         A3 = rem(d, 125)/25;
177 %         if A3 == 0
178 %             A3 = 5;
179 %         end
180 %         d = d - A3*25; F1 = A(3,A3);
181 %         A4 = rem(d, 625)/125;
182 %         if A4 == 0
183 %             A4 = 5;
184 %         end
185 %         d = d - A4*125; K1 = A(4,A4);
186 %         A5 = rem(d, 3125)/625;
187 %         if A5 == 0
188 %             A5 = 5;
189 %         end
190 %         d = d - A5*625; F2 = A(5,A5);
```

```
191 %           A6 = rem(d, 15625)/3125;
192 %           if A6 == 0
193 %               A6 = 5;
194 %           end
195 %           K2 = A(6,A6);
196
197           p = action;
198           %Associate values to the ones in the S variables
199           iF = rem(p,3);
200           p = p - iF;
201           if iF == 0 iF = 3; end
202           F = Aa(iF);
203
204           iKA = rem(p,39)/3;
205           p = p - iKA*3;
206           if iKA == 0 iKA = 13; end
207           K = Ab(iKA);
208           A = Ac(iKA);
209
210           iF = rem(p,117)/39;
211           p = p - iF*39;
212           if iF == 0 iF = 3; end
213           F1 = Aa(iF);
214
215           iKA = rem(p,1521)/117;
216           p = p - iKA*117;
217           if iKA == 0 iKA = 13; end
218           K1 = Ab(iKA);
219           A1 = Ac(iKA);
220
221           iF = rem(p,4563)/1521;
222           p = p - iF*1521;
223           if iF == 0 iF = 3; end
224           F2 = Aa(iF);
225
226           iKA = rem(p,59319)/4563;
227           if iKA == 0 iKA = 13; end
```

```
228         K2 = Ab(iKA);
229         A2 = Ac(iKA);
230
231
232         % do the selected action
233         j = j + 1;
234         ni = SIM_BUV(ni,F,K,A,F1,K1,A1,F2,K2,A2);
235         T = T + 0.5;
236
237         % adquire position
238         x = ni(7); W(j,1) = ni(7);
239         y = ni(8); W(j,2) = ni(8);
240         z = ni(9); W(j,3) = ni(9);
241         Roll = ni(10);
242         W(j,4) = A*sind(2*pi*F*T)+K;
243         Test = A*sind(2*pi*F*T)+K;
244         W(j,5) = A1*sind(2*pi*F1*T)+K1;
245         W(j,6) = A2*sind(2*pi*F2*T)+K2;
246
247
248 %          dpsi = abs(vpsi - ni(12));
249 %          minpsi = min(dpsi);
250 %          for a = 1:10
251 %              if dpsi(a) == minpsi psi = vpsi(a); end
252 %          end
253 %          psi = round(psi/0.6);
254
255
256         status = [x,y,z];
257
258         %set(lnh,'X[m]',W(:,1),'Y[m]',W(:,2),'Z[m]',W(:,3)) % change the
                line data
259
260         % transform matriz x, y, z
261         % xyz = (x + y * 10 + z * 10*10 + psi*10*10*10 ) - 1110;
262
263         % new transformation matriz D, Az, Pch, ?Roll?
```

```
264         iQ = (iD + iAz * 11 + iPch *11*11) - 132;
265
266         % count the actions required to reach the goal
267         countActions = countActions + 1;
268
269         % reward function
270         % R = - (sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2))^2;
271         R = - (sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2))^2;
272         if R == 0 %|| flag == 1
273             flag = 1;
274             %R = 0;
275         end
276         r = r + R;
277
278         % update information for robot in Q for later use
279         Q(prviQ,action) = Q(prviQ,action) + alpha*(R+gamma*max(Q(iQ,:)) -
                Q(prviQ,action));
280     end
281
282 %     if r < -3000
283 %         Re(i) = -3000;
284 %     else
285         Re(i) = r;
286 %     end
287
288     if rem(i,10) == 0
289         i
290     end
291
292     if rem(i,100) == 0 || i == 10 %|| i == 100 || i == 1000
293     close;
294     BUV = figure;
295         subplot(2,3,1);plot3(W(:,1),W(:,2),W(:,3),'r'); xlabel('x [m]');
                ylabel('y [m]'); zlabel('z [m]'); grid;
296         hold on; plot3(5,5,5,'g*'); hold on; plot3(sx,sy,sz,'r*');
297         subplot(2,3,2);plot(W(:,1),W(:,2),'r');xlabel('x [m]');  ylabel('
                y [m]');
```

```
298         hold on; plot(5,5,'g*'); hold on; plot(sx,sy,'r*');
299         subplot(2,3,4);plot(W(:,4),'r'); ylabel('Tail Fin');
300         subplot(2,3,5);plot(W(:,5),'r'); xlabel('Oscilation'); ylabel('
                Left and Right Fin'); hold on; plot(W(:,6),'g');
301         subplot(2,3,3);plot(W(:,3),'r'); xlabel('Time steps (0.5s) p/
                episode');  ylabel('z [m]');hold on; plot(14,'g');
302         subplot(2,3,6);plot(Re(:),'r'); xlabel('N of Episodes'); ylabel('
                Total reward');
303         drawnow
304     end
305
306     if flag == 1
307         close;
308         META = figure;
309         subplot(2,3,1);plot3(W(:,1),W(:,2),W(:,3),'r'); xlabel('x [m]');
                ylabel('y [m]'); zlabel('z [m]'); grid;
310         hold on; plot3(5,5,5,'g*'); hold on; plot3(sx,sy,sz,'r*');
311         subplot(2,3,2);plot(W(:,1),W(:,2),'r');xlabel('x [m]');  ylabel('
                y [m]');
312         hold on; plot(5,5,'g*'); hold on; plot(sx,sy,'r*');
313         subplot(2,3,4);plot(W(:,4),'r'); ylabel('Tail Fin');
314         subplot(2,3,5);plot(W(:,5),'r'); xlabel('Oscilation'); ylabel('
                Left and Right Fin'); hold on; plot(W(:,6),'g');
315         subplot(2,3,3);plot(W(:,3),'r'); xlabel('Time steps (0.5s) p/
                episode');  ylabel('z [m]');hold on; plot(14,'g');
316         subplot(2,3,6);plot(Re(:),'r'); xlabel('N of Episodes'); ylabel('
                Total reward');
317         drawnow
318     end
319
320     iterationCount(i,1) = countActions;
321     i = i + 1;
322     flag = 0;
323
324 end
```

Annexe C - Q-learning C++ code

```
1
2  #define _USE_MATH_DEFINES
3
4  #include "stdafx.h"
5  #include <iostream>
6  #include <Eigen/Dense>
7  #include <math.h>
8
9  using namespace Eigen;
10 using namespace std;
11
12
13 MatrixXd Integration (MatrixXd nider, MatrixXd ni_0, float dt)
14 {
15         MatrixXd vel_local(6, 1);
16         vel_local = ni_0 + dt * nider;
17         return vel_local;
18 }
19
20 MatrixXd Dynamics(MatrixXd ni, MatrixXd tau, float dt)
21 {
22         float u, v, w, p, q, r, x, y, z, phi, theta, psi;
23         u = ni(1, 1);
24         v = ni(2, 1);
25         w = ni(3, 1);
26         p = ni(4, 1);
27         q = ni(5, 1);
28         r = ni(6, 1);
29         x = ni(7, 1);
30         y = ni(8, 1);
31         z = ni(9, 1);
32         phi = ni(10, 1);
33         theta = ni(11, 1);
34         psi = ni(12, 1);
35         MatrixXd vel(1, 6);
36         vel << u, v, w, p, q, r;
```

```
37          vel = vel.transpose();
38
39          float m = 45; float Ix = 0.73; float Iy = 7.72;
40          float Iz = 7.72; float Xup = 4.5; float Yup = 59;
41          float Zup = 59; float Kup = 0;
42          float Mup = 11.2; float Nup = 11.2;
43          MatrixXd MRB(6, 6);
44          MRB <<  m, 0, 0, 0, 0, 0,
45                  0, m, 0, 0, 0, 0,
46                  0, 0, m, 0, 0, 0,
47                  0, 0, 0, Ix, 0, 0,
48                  0, 0, 0, 0, Iy, 0,
49                  0, 0, 0, 0, 0, Iz;
50          MatrixXd MA(6, 6);
51          MA <<   Xup, 0, 0, 0, 0, 0,
52                  0, Yup, 0, 0, 0, 0,
53                  0, 0, Zup, 0, 0, 0,
54                  0, 0, 0, Kup, 0, 0,
55                  0, 0, 0, 0, Mup, 0,
56                  0, 0, 0, 0, 0, Nup;
57          MatrixXd M(6, 6);
58          M = MRB + MA;
59          MatrixXd invM;
60          invM = M.inverse();
61
62          float Xu = 0.0894; float Xuu = 5.72; float Yv = 1.9;
63          float Yvv = 18.58; float Zw = 1.9; float Zww = 18.58;
64          float Kp = 0; float Kpp = 0; float Mq = 0.8;
65          float Mqq = 11;
66          float Nr = 0.7; float Nrr = 10;
67          float D11 = Xu + (Xuu*abs(u));
68          float D22 = Yv + (Yvv*abs(v));
69          float D33 = Zw + (Zww*abs(w));
70          float D44 = Kp + (Kpp*abs(p));
71          float D55 = Mq + (Mqq*abs(q));
72          float D66 = Nr + (Nrr*abs(r));
73          MatrixXd D(6, 6);
```

```
74          D <<     D11, 0, 0, 0, 0, 0,
75                               0, D22, 0, 0, 0, 0,
76                               0, 0, D33, 0, 0, 0,
77                               0, 0, 0, D44, 0, 0,
78                               0, 0, 0, 0, D55, 0,
79                               0, 0, 0, 0, 0, D66;
80
81          float x0 = 0; float y0 = 0; float z0 = 0; float P = 450;
82          float B = 460; float xB = 0; float yB = 0; float zB = 0;
83          float xG = 0; float yG = 0; float zG = 0;
84          MatrixXd G(6, 1);
85          G <<     (P - B)*sin(theta),
86                               (P - B)*cos(theta)*sin(phi),
87                               -(P - B)*cos(theta)*cos(phi),
88                               0,
89                               -(zG*P + zB * B)*sin(theta)*cos(phi)
90  + (xG*P + xB * B)*cos(theta)*cos(phi),
91                               0;
92
93          MatrixXd nider(6, 1);
94          nider = invM * (tau + G - D * vel);
95          return nider;
96  }
97
98  MatrixXd Transformation(MatrixXd vel_local, MatrixXd ni)
99  {
100         float u, v, w, p, q, r, x, y, z, phi, theta, psi;
101         u = ni(1, 1);
102         v = ni(2, 1);
103         w = ni(3, 1);
104         p = ni(4, 1);
105         q = ni(5, 1);
106         r = ni(6, 1);
107         x = ni(7, 1);
108         y = ni(8, 1);
109         z = ni(9, 1);
110         phi = ni(10, 1);
```

```
111         theta = ni(11, 1);
112         psi = ni(12, 1);
113 
114         float c1 = cos(phi);    float s1 = sin(phi);
115         float c2 = cos(theta);  float s2 = sin(theta);
116         float t2 = tan(theta);
117         float c3 = cos(psi);    float s3 = sin(psi);
118 
119         MatrixXd LG(6, 6);
120         LG <<   c3 * c2, c3*s2*s1 - s3 * c1, s3*s1 + c3 * c1*s2,
121 0, 0, 0,
122                         s3*c2, c3*c1 + s3 * s1*s2,
123 s2*s3*c1 - c3 * s1, 0, 0, 0,
124                         -s2, c2*s1, c2*c1, 0, 0, 0,
125                         0, 0, 0, 1, s1*t2, c1*t2,
126                         0, 0, 0, 0, c1, -s1,
127                         0, 0, 0, 0, s1 / c2, c1 / c2;
128 
129         MatrixXd vel_global(6, 1);
130         vel_global = LG * vel_local;
131         return vel_global;
132 }
133 
134 MatrixXd Fins(float ogon_czest, float ogon_kat, float OX_ampl,
135 float p_lewa_czest, float p_lewa_kat, float PLX_ampl,
136 float p_prawa_czest, float p_prawa_kat,
137 float PPX_ampl, float i, float dt, float Vx)
138 {
139         MatrixXd oxsr_pom(3, 1);
140         oxsr_pom << 18,
141                                 12.8,
142                                 1.8;
143         MatrixXd pxsr_pom(3, 1);
144         pxsr_pom << 8,
145                                 6.2,
146                                 0.9;
147         float tc = ogon_czest / 750;
```

```
148         float tl = p_lewa_czest / 750;
149         float tr = p_prawa_czest / 750;
150 
151         float przekladnia_ogon = 12;
152         float przekladnia_pletwa = 12;
153         float ogoncz = ogon_czest / 60 / przekladnia_ogon;
154         float prawacz = p_prawa_czest / 60 / przekladnia_pletwa;
155         float lewacz = p_lewa_czest / 60 / przekladnia_pletwa;
156         float r1 = 0.1; float r2 = 0.4;
157         float r3 = 0.3; float r4 = 0.13;
158         float xG = 0;
159 
160         float OX_sr = oxsr_pom(tc, 1);
161         float OX = OX_sr + OX_ampl
162 * sin(i / dt * ogoncz * M_PI/180);
163         float PLX_sr = pxsr_pom(tl, 1);
164         float PLX = PLX_sr + PLX_ampl
165 * sin(i / dt * lewacz * M_PI/180);
166         float PPX_sr = pxsr_pom(tr, 1);
167         float PPX = PPX_sr + PPX_ampl
168 * sin(i / dt * prawacz * M_PI / 180);
169 
170         float ogonX = OX * cos(ogon_kat * M_PI / 180);
171         float ogonY = OX * sin(ogon_kat * M_PI / 180);
172         float promien1 = sqrt (pow((r1
173 + (r2*cos(ogon_kat * M_PI / 180))), 2)
174 + pow((sin(ogon_kat * M_PI / 180)*r2 ), 2));
175         float beta = atan((sin(ogon_kat * M_PI / 180)*r2)
176 / (r1 + (cos(ogon_kat * M_PI / 180)*r2))* M_PI / 180);
177         float ogonN = promien1 * ogonY * cos(beta * M_PI / 180)
178 * (-1);
179 
180         float pletwaLX = PLX * cos(p_lewa_kat * M_PI / 180);
181         float pletwaLZ = PLX * sin(p_lewa_kat * M_PI / 180);
182         float pletwaLN = r4 * pletwaLX;
183         float pletwaLM = r3 * pletwaLZ * (-1);
184         float pletwaPX = PPX * cos(p_prawa_kat * M_PI / 180);
```

```
185         float pletwaPZ = PPX * sin(p_prawa_kat * M_PI / 180);
186         float pletwaPN = r4 * pletwaPX;
187         float pletwaPM = r3 * pletwaPZ * (-1);
188 
189         float xvx = 0.014; float xvxx = 0.91;
190         float Opor_pletwa = 0; float D = 0;
191         float beta = 90 - (atan(r4 / r3 * M_PI / 180));
192         float r = sqrt (pow(r3, 2) + pow(r4, 2));
193 
194         if ((p_prawa_czest == 0) && (p_prawa_kat == 90))
195         {
196                 D = xvx + (xvxx*abs(Vx));
197                 Opor_pletwa = cos(beta * M_PI / 180)*D*r;
198         }
199         if ((p_lewa_czest == 0) && (p_lewa_kat == 90))
200         {
201                 D = xvx + (xvxx*abs(Vx));
202                 Opor_pletwa = cos(beta * M_PI / 180)*D*r*(-1);
203         }
204 
205         MatrixXd tau(6, 1);
206         tau << (ogonX + pletwaLX + pletwaPX - D),
207                         ogonY,
208                         (pletwaLZ + pletwaPZ),
209                         0,
210                         (pletwaLM + pletwaPM),
211                         (ogonN - pletwaPN + pletwaLN
212 - Opor_pletwa);
213 
214         return tau;
215 }
216 
217 MatrixXd SIM_BUV(MatrixXd ni, float F, float K, float A,
218 float F1, float K1, float A1, float F2, float K2, float A2)
219 {
220         float rad = 180 / M_PI;
221         float T = 0.5;
```

```
222
223         float T_global = 0;
224         float dt = 1 / 18.0;
225
226         MatrixXd nider(6, 1);
227         nider <<         0,
228                          0,
229                          0,
230                          0,
231                          0,
232                          0;
233
234         MatrixXd tau(6, 1);
235         tau <<  0,
236                         0,
237                         0,
238                         0,
239                         0,
240                         0;
241
242         float N = ceil(T / dt);
243         float Vp = 0.0;
244         float AlfaP = 0 * M_PI / 180;
245
246         for (int i = 1; i <= N + 1; i++)
247         {
248                 tau = Fins(F, K, A, F1, K1, A1, F2, K2, A2, i,
249 dt, ni(1, 1));
250                 nider = Dynamics(ni, tau, dt);
251                 MatrixXd vel_0(6, 1);
252                 vel_0(1, 1) = ni(1, 1);
253                 vel_0(2, 1) = ni(2, 1);
254                 vel_0(3, 1) = ni(3, 1);
255                 vel_0(4, 1) = ni(4, 1);
256                 vel_0(5, 1) = ni(5, 1);
257                 vel_0(6, 1) = ni(6, 1);
258                 MatrixXd vel_local(6, 1);
```

```
259                 vel_local = Integration(nider, vel_0, dt);
260
261                 float uP = Vp * cos(AlfaP - ni(12, 1) - M_PI);
262                 float vP = Vp * sin(AlfaP - ni(12, 1) - M_PI);
263                 vel_local(1, 1) = vel_local(1, 1) + uP;
264                 vel_local(2, 1) = vel_local(2, 1) + vP;
265                 MatrixXd vel_global(6, 1);
266                 vel_global = Transformation(vel_local, ni);
267                 MatrixXd coord_0(6, 1);
268                 coord_0(1, 1) = ni(7, 1);
269                 coord_0(2, 1) = ni(8, 1);
270                 coord_0(3, 1) = ni(9, 1);
271                 coord_0(4, 1) = ni(10, 1);
272                 coord_0(5, 1) = ni(11, 1);
273                 coord_0(6, 1) = ni(12, 1);
274                 MatrixXd coord_global(6, 1);
275                 coord_global = Integration(vel_global,
276 coord_0, dt);
277                 ni(1, 1) = vel_local(1, 1);
278                 ni(2, 1) = vel_local(2, 1);
279                 ni(3, 1) = vel_local(3, 1);
280                 ni(4, 1) = vel_local(4, 1);
281                 ni(5, 1) = vel_local(5, 1);
282                 ni(6, 1) = vel_local(6, 1);
283                 ni(7, 1) = coord_global(1, 1);
284                 ni(8, 1) = coord_global(2, 1);
285                 ni(9, 1) = coord_global(3, 1);
286                 ni(10, 1) = coord_global(4, 1);
287                 ni(11, 1) = coord_global(5, 1);
288                 ni(12, 1) = coord_global(6, 1);
289                 T_global = T_global + dt;
290         }
291
292         return ni;
293
294 }
295
```

```
296 int main()
297 {
298         MatrixXd Aa(3, 1);
299         Aa << 750,
300                                 1500,
301                                 3000;
302         MatrixXd Ab(13, 1);
303         ni << 0,
304                                 0,
305                                 0,
306                                 10,
307                                 10,
308                                 10,
309                                 -10,
310                                 -10,
311                                 -10,
312                                 30,
313                                 30,
314                                 -30,
315                                 -30;
316         MatrixXd Ac(13, 1);
317         Ac << 0,
318                                 2.5,
319                                 5,
320                                 0,
321                                 2.5,
322                                 5,
323                                 0,
324                                 2.5,
325                                 5,
326                                 0,
327                                 2.5,
328                                 0,
329                                 2.5;
330
331         MatrixXd Q(14641, 59319);
332         Q << 0;
```

```
333         MatrixXd ni(12, 1);
334         ni << 0,
335                                 0,
336                                 0,
337                                 0,
338                                 0,
339                                 0,
340                                 0,
341                                 0,
342                                 0,
343                                 0,
344                                 0,
345                                 0;
346         float alpha = 0.8;
347         float gamma = 0.5;
348         float E = 0.15;
349
350         T = 0;
351         float r = 0;
352
353         int xg = 5;
354         int yg = 5;
355         int zg = 5;
356
357         float T = 0;
358         int flag = 0;
359         int i = 0;
360         float = 180/3.14;
361
362         while (flag == 0)
363         do
364         {
365                 ni << 0,
366                                 0,
367                                 0,
368                                 0,
369                                 0,
```

```
370                                 0,
371                                 0,
372                                 0,
373                                 0,
374                                 0,
375                                 0,
376                                 0;
377
378             int x = 3; int sx = 3; ni(7, 1) = 3;
379             int y = 7; int sy = 7; ni(8, 1) = 7;
380             int z = 7; int sz = 7; ni(9, 1) = 7;
381
382             int j = 1;
383
384             float D = sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2);
385             int iD = round(D);
386             if (iD > 10)
387             {
388                     iD = 10;
389             }
390
391             float dx = x - xg;
392             float dy = y - yg;
393             float psig;
394             if ((dx <= 0) && (dy <= 0))
395             {
396                     psig = asin(abs(dx)/d);
397             }
398             else if ((dx <= 0) && (dy > 0))
399             {
400                     psig = asin(abs(dy)/d) + 3.14/2;
401             }
402             else if ((dx > 0) && (dy > 0))
403             {
404                     psig = asin(abs(dx)/d) + 3.14;
405             }
406             else
```

```
407                 {
408                         psig = asin(abs(dy)/d) + 3*3.14/2;
409                 }
410                 float pAz = psig - ni(12, 1);
411                 MatrixXd mAz(11, 1);
412                 mAz << -90,
413                                 -60,
414                                 -30,
415                                 -10,
416                                 -5,
417                                 0,
418                                 5,
419                                 10,
420                                 30,
421                                 60,
422                                 90;
423
424                 float Az = 0;
425                 float dAz = 360;
426                 int iAz = 1;
427                 for (int g = 1; g <= 11; g++)
428                 {
429                         if (abs(mAz(g, 1) - pAz * rad) <= dAz)
430                         {
431                                 dAz = abs(mAz(g, 1) - pAz * rad);
432                                 Az = mAz(g, 1);
433                                 iAz = g;
434                         }
435                 }
436
437                 float dz = z - zg;
438                 float thetag;
439                 if (dz >= 0)
440                 {
441                         thetag = - asin(abs(dz)/D);
442                 }
443                 else
```

```
444                 {
445                         thetag = asin(abs(dz)/D);
446                 }
447                 float pPch = thetag - ni(11, 1);
448                 MatrixXd mPch(11, 1);
449                 mPch << -90,
450                                 -60,
451                                 -30,
452                                 -10,
453                                 -5,
454                                 0,
455                                 5,
456                                 10,
457                                 30,
458                                 60,
459                                 90;
460
461                 float Pch = 0;
462                 float dPch = 360;
463                 int iPch = 1;
464                 for (int g = 1; g <= 11; g++)
465                 {
466                         if (abs(mPch(g, 1) - pPch * rad) <= dPch)
467                         {
468                                 dPch = abs(mPch(g, 1)
469 - pPch * rad);
470                                 Pch = mPch(g, 1);
471                                 iPch = g;
472                         }
473                 }
474
475                 MatrixXd mRoll(11, 1);
476                 mRoll << -90,
477                                 -60,
478                                 -30,
479                                 -10,
480                                 -5,
```

```
481                                 0,
482                                 5,
483                                 10,
484                                 30,
485                                 60,
486                                 90;
487
488                 float pRoll = ni(10, 1);
489                 float Roll = 0;
490                 float dRoll = 360;
491                 int iRoll = 1;
492                 for (int g = 1; g <= 11; g++)
493                 {
494                         if (abs(mRoll(g, 1) - pRoll * rad)
495 <= dRoll)
496                         {
497                                 dRoll = abs(mRoll(g, 1)
498 - pRoll * rad);
499                                 Roll = mRoll(g, 1);
500                                 iRoll = g;
501                         }
502                 }
503
504                 int iQ = (iD + iAz * 11 + iPch *11*11
505 + iRoll *11*11*11)- 1463;
506
507                 while (T < 10)
508                 do
509                 {
510                         int priviQ = iQ;
511
512                         float maxQ = 0;
513                         int action = 0;
514
515                         float rnd = (rand() % 101) / 100.0;
516
517                         if (rnd < E)
```

```
518                     {
519                             action = rand() % 58319 + 1;
520                     }
521                     else
522                     {
523                             for (int g = 1; g <= 59319; g++)
524                             {
525                                     if (Q(iQ, g) >= maxQ)
526                                     {
527                                             maxQ = Q(iQ, g);
528                                             action = g;
529                                     }
530                             }
531                     }
532
533                     int p = action;
534
535                     float iF = p%3;
536                     p = p - iF;
537                     if (iF == 0) {iF = 3;}
538                     float F = Aa(iF, 1);
539
540                     float iKA = p%39/3;
541                     p = p - iKA*3;
542                     if (iKA == 0) {iKA = 13;}
543                     float K = Ab(iKA, 1);
544                     float A = Ac(iKA, 1);
545
546                     iF = p%117/39;
547                     p = p - iF*117;
548                     if (iF == 0) {iF = 3;}
549                     float F1 = Aa(iF, 1);
550
551                     iKA = p%1521/117;
552                     p = p - iKA*117;
553                     if (iKA == 0) {iKA = 13;}
554                     float K1 = Ab(iKA, 1);
```

```
555                         float A1 = Ac(iKA, 1);
556
557                         iF = p%4563/1521;
558                         p = p - iF*1521;
559                         if (iF == 0) {iF = 3;}
560                         float F2 = Aa(iF, 1);
561
562                         iKA = p%59319/4563;
563                         p = p - iKA*4563;
564                         if (iKA == 0) {iKA = 13;}
565                         float K2 = Ab(iKA, 1);
566                         float A2 = Ac(iKA, 1);
567
568                         j = j + 1;
569                         ni = SIM_BUV(ni, F, K, A, F1, K1, A1,
570 F2, K2, A2);
571                         T = T + 0.5;
572
573                         float D = sqrt((x-xg)^2 + (y-yg)^2
574 + (z-zg)^2);
575                         int iD = round(D);
576                         if (iD > 10)
577                         {
578                                 iD = 10;
579                         }
580
581                         float dx = x - xg;
582                         float dy = y - yg;
583                         float psig;
584                         if ((dx <= 0) && (dy <= 0))
585                         {
586                                 psig = asin(abs(dx)/d);
587                         }
588                         else if ((dx <= 0) && (dy > 0))
589                         {
590                                 psig = asin(abs(dy)/d) + 3.14/2;
591                         }
```

```
592                         else if ((dx > 0) && (dy > 0))
593                         {
594                                 psig = asin(abs(dx)/d) + 3.14;
595                         }
596                         else
597                         {
598                                 psig = asin(abs(dy)/d)
599 + 3*3.14/2;
600                         }
601                         float pAz = psig - ni(12, 1);
602                         float Az = 0;
603                         float dAz = 360;
604                         int iAz = 1;
605                         for (int g = 1; g <= 11; g++)
606                         {
607                                 if (abs(mAz(g, 1) - pAz * rad)
608 <= dAz)
609                                 {
610                                         dAz = abs(mAz(g, 1)
611 - pAz * rad);
612                                         Az = mAz(g, 1);
613                                         iAz = g;
614                                 }
615                         }
616
617                         float dz = z - zg;
618                         float thetag;
619                         if (dz >= 0)
620                         {
621                                 thetag = - asin(abs(dz)/D);
622                         }
623                         else
624                         {
625                                 thetag = asin(abs(dz)/D);
626                         }
627                         float pPch = thetag - ni(11, 1);
628                         float Pch = 0;
```

```
629                         float dPch = 360;
630                         int iPch = 1;
631                         for (int g = 1; g <= 11; g++)
632                         {
633                                 if (abs(mPch(g, 1) - pPch * rad)
634 <= dPch)
635                                 {
636                                         dPch = abs(mPch(g, 1)
637 - pPch * rad);
638                                         Pch = mPch(g, 1);
639                                         iPch = g;
640                                 }
641                         }
642
643                         MatrixXd mRoll(11, 1);
644                         mRoll << -90,
645                                         -60,
646                                         -30,
647                                         -10,
648                                         -5,
649                                         0,
650                                         5,
651                                         10,
652                                         30,
653                                         60,
654                                         90;
655
656                         float pRoll = ni(10, 1);
657                         float Roll = 0;
658                         float dRoll = 360;
659                         int iRoll = 1;
660                         for (int g = 1; g <= 11; g++)
661                         {
662                                 if (abs(mRoll(g, 1)
663 - pRoll * rad) <= dRoll)
664                                 {
665                                         dRoll = abs(mRoll(g, 1)
```

```
666 - pRoll * rad);
667                                             Roll = mRoll(g, 1);
668                                             iRoll = g;
669                                         }
670                                 }
671
672                                 iQ = (iD + iAz * 11 + iPch *11*11
673 + iRoll *11*11*11) - 1463;
674
675                                 float R = - (abs(Roll)/10+1)*
676 (sqrt((x-xg)^2 + (y-yg)^2 + (z-zg)^2))^2;
677                                 r = r + R;
678
679                                 Q(prviQ,action) = Q(prviQ,action)
680 + alpha*(R + gamma*maxQ - Q(prviQ,action));
681                             }
682                 }
683     return 0;
684 }
```