学校代码	10699
分 类 号	V448.2
密　　级	公开
学　　号	2019280213

题目 Research on Autonomous Planning of Robotic in-Space Assembly Using Reinforcement Learning

作者 Jorand Gallou

学 科、专 业 航空宇航科学与技术

指 导 教 师 王明明

申请学位日期 2021 年 06 月

西 北 工 业 大 学

硕 士 学 位 论 文

(学位研究生)

题目 :Research on Autonomous Planning of Robotic in-Space Assembly Using Reinforcement Learning

作　　者 : Jorand Gallou

学科、专业 : 航空宇航科学与技术

指 导 教 师 : 王明明

2021 年 06 月

Research on Autonomous Planning of Robotic in-Space Assembly Using Reinforcement Learning

By

Under the Supervision of Professor
Wang Ming Ming

A Dissertation Submitted to
Northwestern Polytechnical University

In Partial Fulfillment of the Requirement
for the Degree of
Master of Flight Vehicle Design

Xi'an P. R. China
June 2021

Abstract

从人类进行太空探索以来，几乎所有的航天器都是在地面上制造和组装，然后集成到运载火箭中送入轨道。这种方法对有效载荷的尺寸、体积和设计造成了很大的限制。此外，望远镜和天线的尺寸与它们的性能密切相关。因此，对空间装配进行改进就很有必要，以摆脱这些限制。通过在轨组装，发射的飞行器只需要搭载这些更大更复杂的结构所需的模块部件，然后通过机器人进行建造。在建造坚固且轻便的结构方面，桁架组装发挥了重要作用。桁架装配方案可以用一个简单的结构构建复杂的结构，大大减少了运载火箭的空间和重量。

求解运动规划是一个找到将物体从起点移动到终点的机械臂有效构型序列的计算问题。其目标是为机器人在其工作空间中找到运动的最佳路径。在过去的几十年里，强化学习（RL，reinforcement learning）作为机器人编程的最佳方式之一，提供了更好的自主性和可靠性。机器学习可以被认为是最优控制理论的一个新分支。本论文的主要研究内容涵盖了不同的领域，以开发一种新的技术来实现望远镜等结构的自主机器人空间组装。

首先，本文的研究重点是新型的桁杆设计，以实现更加高效的桁架装配。桁杆间的连接需要先进的机器人技术，涉及双机械臂协调操作的问题。本文提出的使用磁铁进行桁杆节点连接的方法，大大降低了机器人装配操作的难度。同时，在望远镜组装的实例中介绍了两种设计方案。一是带槽杆件的设计，望远镜镜面可以在槽内滑动；二是在杆件和镜子背面附加磁铁，使其固定在结构上。第二种技术以”+” 的形式出现，更加轻巧和方便组装，更具说服力。

其次，本文研究了自主路径规划方法。首先介绍了研究中采用的 UR10 机械臂的运动学，分别对正向和逆向运动学进行了研究。这一部分对于理解机械臂的工作原理是非常必要的。然后介绍了有助于理解 RL 的工具，如马尔科夫决策过程和贝尔曼方程，在学习过程中用于更新状态。此外，此部分还对 Q-Learning 和 Deep-Q-Learning（DQL）进行了解释。为了更好地理解 DQL，详细介绍了神经网络。

最后，详细介绍了仿真与实验的过程，并对结果作了深入的分析。本文选择 Pybullet 作为仿真与实验的软件，对 Q-Learning 和 DQL 的可到达环境进行了比较。这种环境包括在最小的步骤内到达目标点的集合内。结果显示最有效的方法是 DQL，它是更适合于类似于机器人连续任务的策略。最后一部分是关于挑选和放置环境。它是在一个位置拣选一个磁化球，然后把它放到桌子上的另一个位置。该任务任务非常接近于组装，并给出了满意的结果。由于采取了经验回放和目标网络方法，在训练过程中没有出现过拟合现象。

Key words : 运动规划，神经网络，在轨装配，机械臂，强化学习，桁架.

Since the start of the space conquest, almost all spacecraft have been manufactured and assembled on the ground, then integrated into a launch vehicle for delivery into orbit. This approach imposes significant limitations on the size, volume, and design of payloads. In addition, the size of the telescopes and antennas is intimately linked to their performance. Therefore there is a need for improvement of the space assembly to get rid of these limitations. With the on-orbit assembly, the launched vehicle only embarks the modular components required for bigger and more complex structures which are then build via a robot. Truss assembly plays an important role when one needs to build strong and light structures. Truss allows constructing complex structures from a simple one. It considerably reduces the amount of space and weight in the launch vehicle.

Motion planning is a computational problem to find a sequence of valid configurations that moves the object from the start to the goal. The idea is to find the optimal path for a robot in its environment. In the last few decades reinforcement learning (RL) has appeared to be one of the best ways to program robots, it provides better autonomy and reliability. Machine learning could be considered as a new branch of optimal control theory. The main research contents of this thesis cover different areas for the development of a new technique to realize autonomous robotic in-space assembly for structures such as telescopes.

Firstly, the research focuses on a new design of bars to build more efficient truss assembly. The joint nodes require techniques that involve the most advanced techniques in robotic and usually require two robotic arms. The concept exposed here uses magnets for the node connection which considerably reduces the difficulty of the assembly for the robot. Then two techniques are presented in the example of telescope assembly, one with grooved bars on which the mirrors are slide and a second one which incorporates magnets to the bars and the back of the mirror to fix it to the structures. The second technique with the bars in a form of ”+” has been more conclusive because easier to assemble and lighter.

Secondly, autonomous path planning is studied. First of all the kinematic of the UR10 is presented which is the robot manipulator used during the research. This part is necessary to understand how the robot manipulator works. The forward and inverse kinematic are studied. Then tools to understand RL are exposed such as Markov Decision Process and the Bellman equation which is used to update the states during the learning process. Also, Q-Learning and Deep-Q-Learning (DQL) are explained in this part. Neural Networks are detailed for a better understanding of DQL.

Finally, the simulation and the experiments are detailed. In this chapter, the evolution of the experiments is shown. Pybullet is the final software that was chosen to conduct the experiments. The $Reach$ environment is compared for Q-Learning and DQL. This environment

consists of reaching a target point within the minimum steps. The most efficient technique is DQL, it is the more suitable policy for continuous tasks such as robotic. The last section is about the $Pick\ and\ Place$ environment. It is picking a magnetized ball at one position and placing it in another location on the table. This last task is very close to an assembly and gave promising results. Overfitting hasn't occurred during the training thanks to the use of experience replay and target network.

Key words : Motion Planning, Neural Network, Orbital Assembly, Robotic arm, Reinforcement Learning, Truss.

Table of Contents

Acronyms

DDPG - Deep Deterministic Policy Gradient ;
DNN - Deep Neural Network ;
DOF - Degree Of Freedom ;
DQL - Deep Q Learning ;
DRL - Deep Reinforcement Learning ;
GPU - Graphical Process Unit ;
GUI - Graphical Unit Interface ;
HST - Hubble Space Telescope ;
ISA - In Space Assembly ;
ISS - International Space Station ;
MDP - Markov Decision Process ;
NN - Neural Network ;
RL - Reinforcement Learning ;
RRT - Rapidly exploring Random Tree ;
URDF - Unified Robot Description File

List of Tables

List of Figures

1 Introduction

1.1 Background

The assembly of spacecraft on the ground and their integration into a launch vehicle places many constraints (mass, volume, and load) on the capabilities that can be deployed in space [1], including adding to the cost of launch. The sizes of the systems such as telescope are limited by the launch vehicle capabilities [2, 3]. Another issue related to this approach is the technology obsolescence due to the long-term ground construction and verification process [4]. In contrast, on-orbit assembly offers a pathway to address such limitations in a variety of ways [5-7].

As the following figures highlight Figure 1-1 and Figure 1-2, over time the size of the structures in space are increasing to gain in performance.

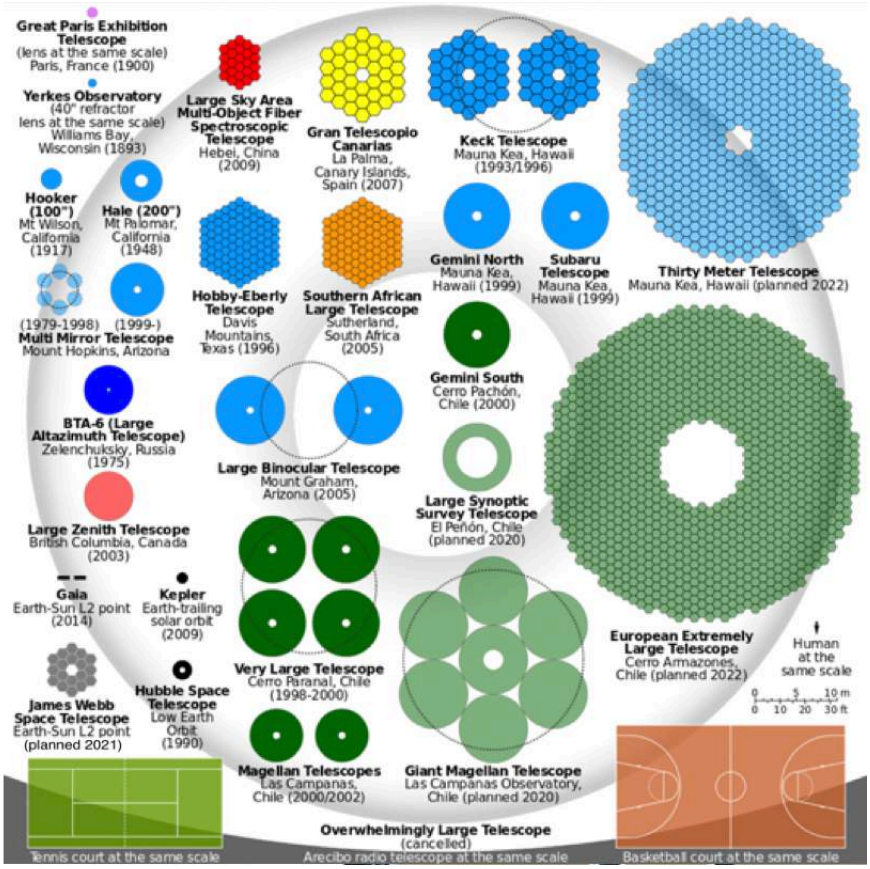


Fig. 1-1 Evolution of the size of the telescope from [8]

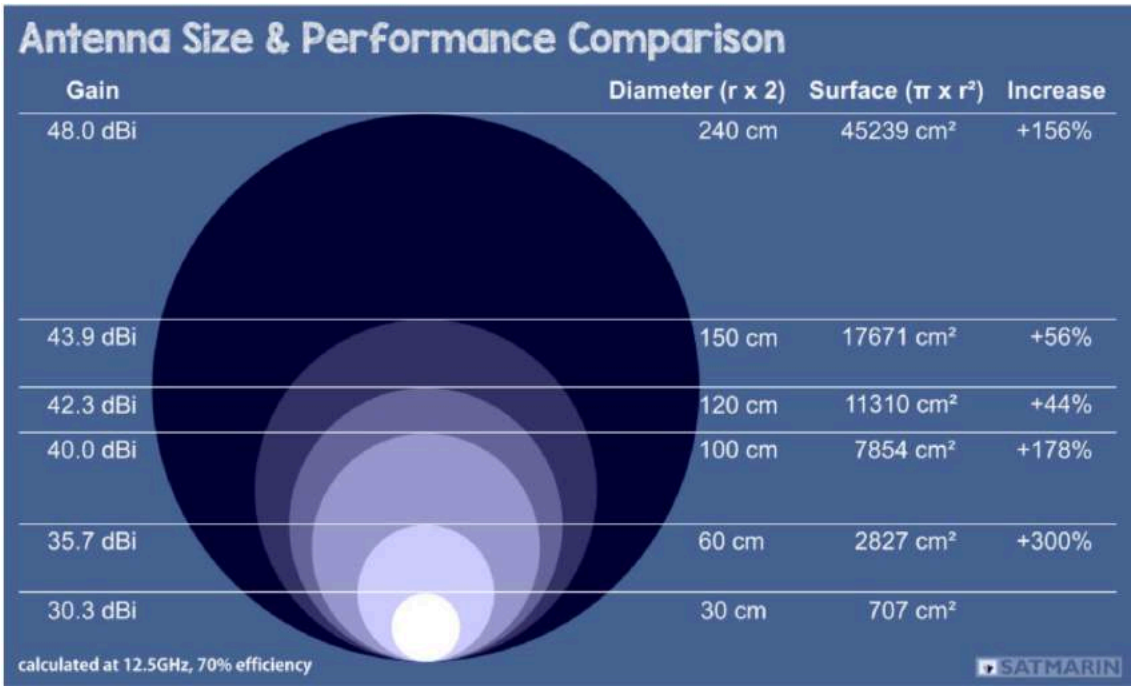


Fig. 1-2 Relationship between performance and antenna size from Satmarin (2017)

Different techniques already exist for large structure assembly. The main examples of In Space Assembly are the Hubble Space Telescope (HST) [9] and the International Space Station (ISS) [10].

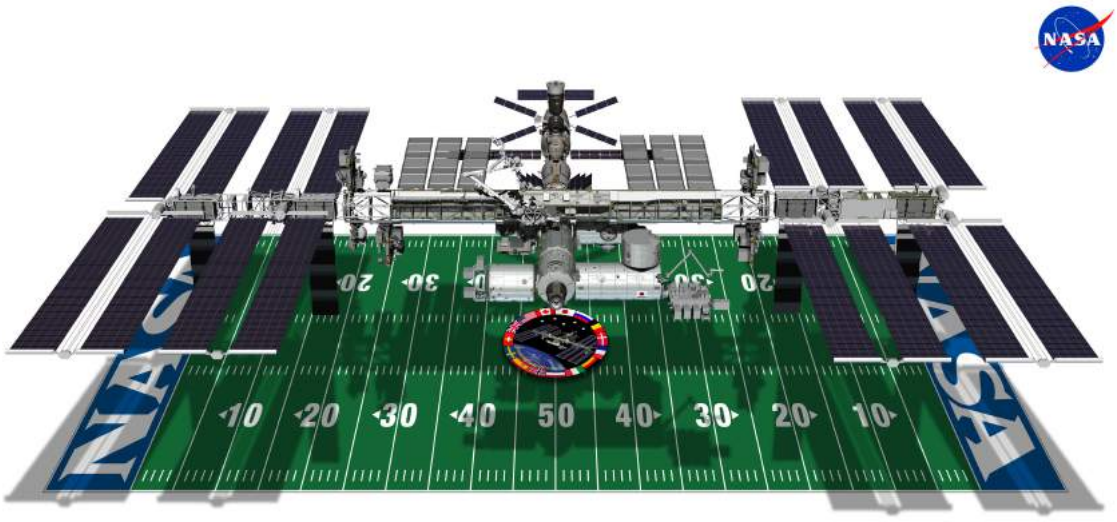


Fig. 1-3 International Space Station

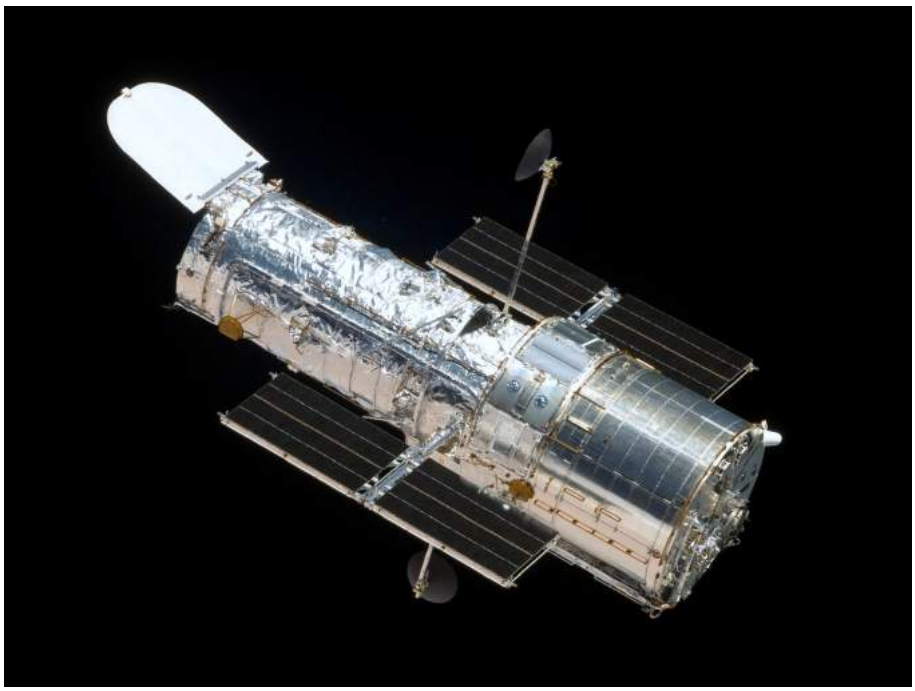


Fig. 1-4 Hubble telescope

Hubble was launched in 1990 and built with on-orbit servicing in mind. Astronauts were trained in the intricacies of the systems and the modularity of the parts. After launch, a servicing mission ensued to repair the mirror and blurry optics. Five servicing missions over the next 12 years followed, lengthening the lifespan of the telescope and improving its capabilities [11]. HST still produces valuable science today.

The assembly of the ISS involved over 160 spacewalks spanning 1,061 hours. With assembly now complete, the station is the size of a football field as illustrated in Figure 1-3. The ISS encompasses over 900 cubic meters of pressurized volume and has been home to over 200 people representing 15 countries.

The frequent servicing missions for the HST and the ISS and the Columbia accident motivated Northrop Grumman to designed a Hubble Robotic Servicing Vehicle (Lillie 2006), complete with two robotic arms having seven degrees of freedom and a 23-foot total arm span called Dextre [12] developed by Macdonald, Dettweiler, and Associates (MDA) of Canada. MDA has extensive experience in robotic, human-in-the-loop servicing, having developed Canadarm [13] for the Space Shuttle and Canadarm2 and Dextre for the International Space Station.

Since the 1970s, many international scientific research institutions have begun to study techniques for constructing large structures in space [14-16]. Studies revealed that trusses are relative structural simplicity with high packaging efficiency [17] they can be assembled piece-by-piece on-orbit. They form the primary support structure in many missions including aero-

brakes [18], telescopes [19], and solar array fields [20]. The elements are designed to be inserted and removed singly in "random access", thus eliminating some assembly order constraints and enabling a physical realization of the construction process.

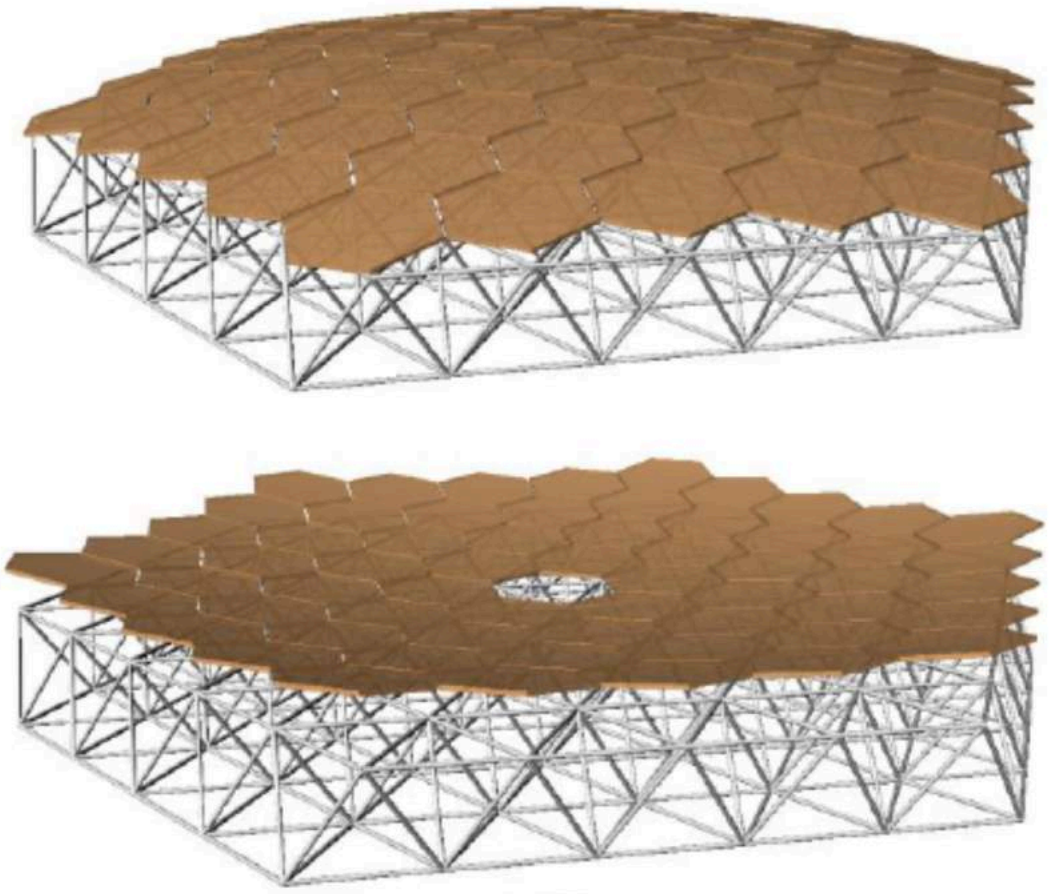


Fig. 1-5 Large space structures. (Top) Aerobrake, (Bottom) Precision segmented reflector from [18]

With the development of In Space Assembly (ISA), in addition to space, remote control robots [21] researchers began to work on fully autonomous space robot systems for autonomous assembly [22]. Autonomous assembly of large structures in space is a key challenge to implement future missions that will necessitate structures to be self-deployed as a single piece [15]. In the case of the James Webb telescope [23], the assembly is simply done by unfolding the mirrors once in space as shown in Figure 1-6. This technique limits the size of the telescope and increases the payload required. It is not the best way to proceeds to embark such large systems. Only substructures should be sent to space and then assembly [16]. The substructures already designed are not very adaptable, it is still composed of modular systems [24-26] or unfolding structures as on Figure 1-7 [27, 28]. The joints are rather not convenient for robotic assembly it usually requires the robot to make complex sequencing [29, 30].

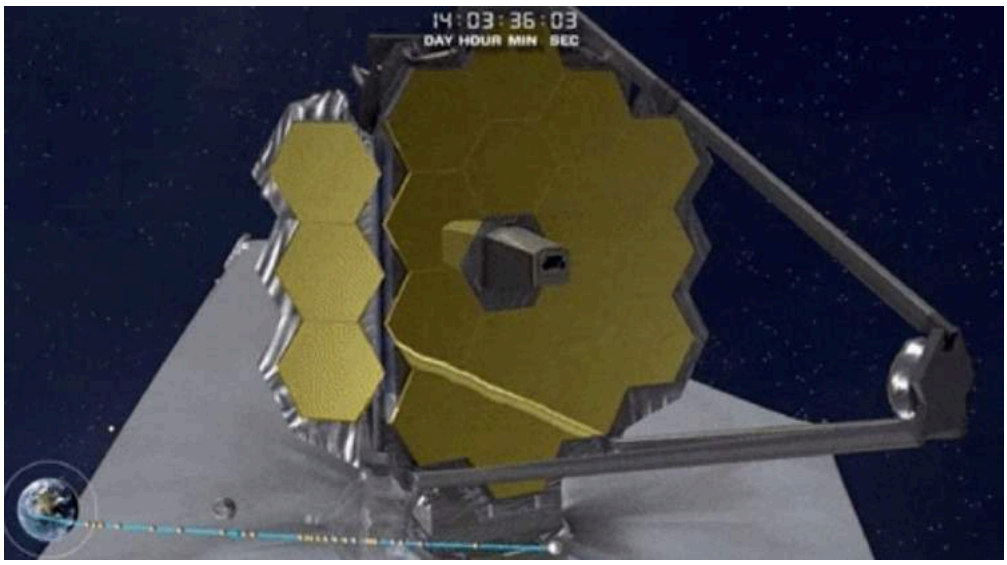


Fig. 1-6 James Webb telescope unfolding in space

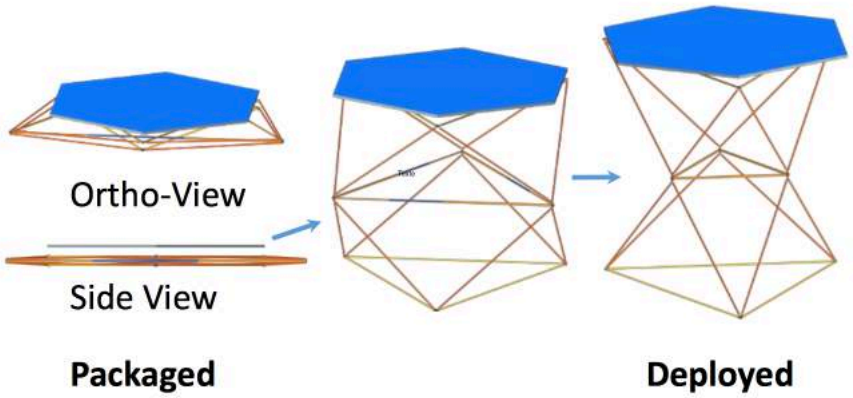


Fig. 1-7 Unfolding structures from [27]

A robot that could be highly adaptable could be used for many different assemblies and would be sent only once with no need to take it back to earth for reprogramming. A 6-DOF arm combined with a truss assembly also adaptable to different kind of structures (antenna, telescopes...) could completely change the space assembly and have benefits in many different fields;

- In astronomy, these assemblies could enable the construction of telescopes too large to be fully built on Earth and launched into orbit.
- In Earth science, on-orbit truss assembly could reduce the number of satellite launches for weather and climate observations through the creation of a persistent platform assembled in space.

- The payoff is not limited to science and exploration alone. On-orbit manufacturing and assembly could also provide a payoff for commercial missions, especially communication satellites in geosynchronous Earth orbit (GEO).

- On-orbit assembly has the potential to provide unique returns for the national security community. For reconnaissance missions, for example, the orbital assembly could provide the ability to assemble larger apertures than feasible on fully assembled satellites to achieve greater spatial resolution.

To apply robots in space for construction missions, lots of key robotic technologies are required. Robot-based assembly in the absence of gravity addresses fundamental technical questions that do not exist for terrestrial applications. While teleoperated or partially assisted assembly operations are possible on the ground, in space constructions require autonomous assembly [31]. The robotic assembly process is made through the combination of adaptable perception, integrated assembly and grasp planning, and compliant control of the manipulators [32]. The robot has to be able to reach the elements of assembly with precision then grasp them and finally place them with the right orientation to the desired position for an assembly.

In these challenging techniques, one of the most important technology is motion planning. The trajectories or steps for an assembly that may be obvious for a human have to be carefully studied and programmed for a robot. Motion planning, also path planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. Robotic motion planning is a well-studied field at the intersection of optimal control, artificial intelligence, and applied mechanics [33].

1.2 Relative Works

Orbital assembly is knowing a new age with the use of robots since the late 1990s [13, 21, 34]. Human-assisted assembly will continue to play a role but it's reaching its limits [35]. A Spectrum of robotic techniques could be used to supplement human assembly and services such as the new robot developed by NASA called "Robonaut" [36, 37] which is costly and pose a risk to human life. On another hand Reinforcement Learning applied to a robot, the manipulator has recently gained the attention of industrial and research teams around the world. In this section, the work related to this study will be presented for a better understanding of nowadays evolution.

1.2.1 Orbital assembly projects

On-Orbit Manufacturing and Assembly is just arising, in their work [8] explores the benefits of it. Made In Space launched a 3D printer to the ISS in partnership with NASA as a

technology demonstration project. Their work is now going further, they intend to deploy satellites that 3D prints itself and built its own solar array. Made in Space is working with NASA on the program OSAM-2 (On-Orbit Servicing, Manufacturing and Assembly) [38] for the next generation of orbit assembly previously called Archinaut One [39]. 3D printing is a big part of the project but there also the robotic part. A Universal Robot (UR) is used for the assembly of the printed truss.

Fig. 1-8 On-Orbit servicing, manufacturing, and assembly by Made in Space Inc

PULSAR which stands for Prototype for an Ultra Large Structure Assembly Robot [40] is a European Project that aims to develop and demonstrate key technologies for in-space assembly of the primary mirror of a 12m diameter telescope. It's the autonomous assembly by a robot of previously developed building blocks. The project focuses on the assembly of a mirror but the developed technology will apply to other large structures.

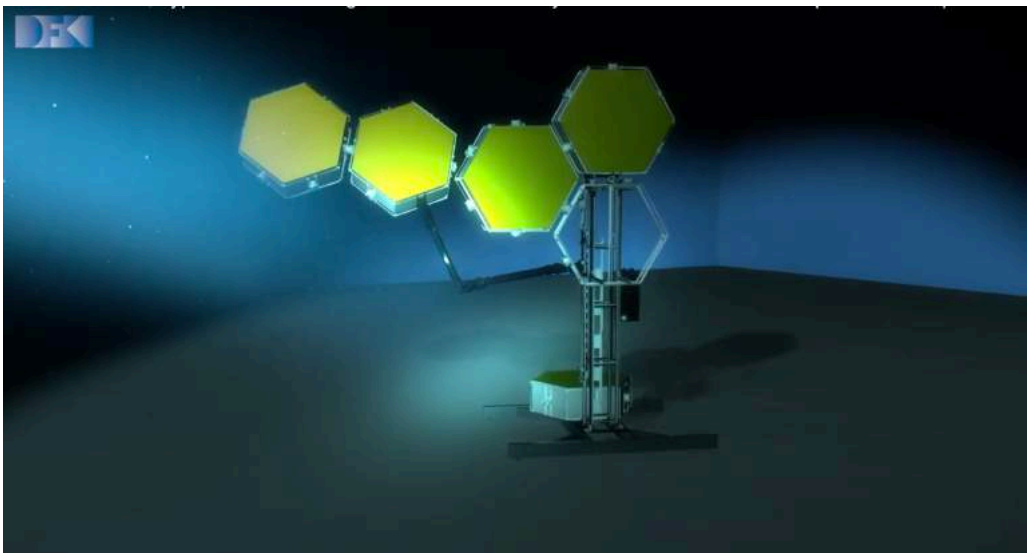


Fig. 1-9 Project PULSAR underwater testing

The Japanese Aerospace Exploration Agency (JAXA) is leading research on large-scale Structure assembly technology [41]. The Space Solar Power System (SSPS) requires robotic assembly technology that will be critical for the safe and affordable construction of kilometer-scale structures in orbit. As a first step, they have been researching a robotic assembly technology capable of assembling a 100-meter-scale space structure in orbit.

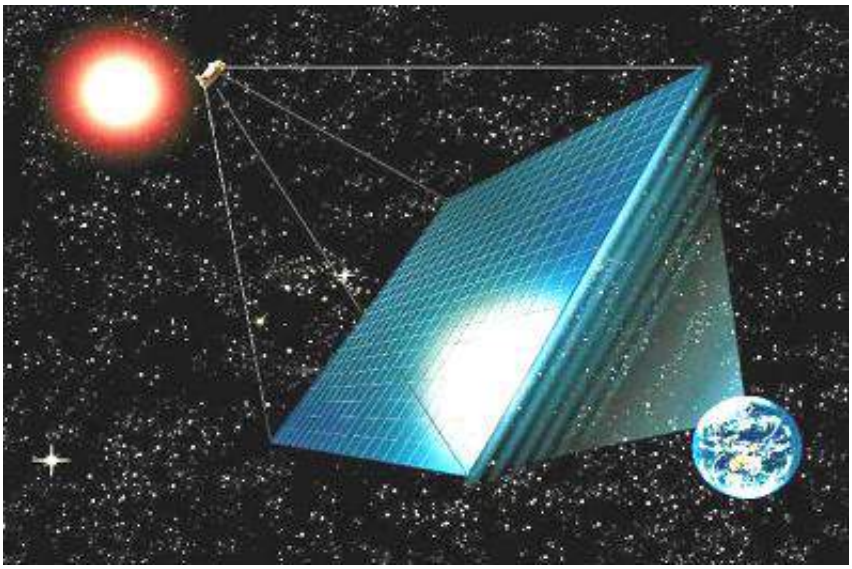


Fig. 1-10 JAXA SSPS project

1.2.2 Truss Assembly

In the early 1990s, researchers at NASA Langley Research Center realized the potential for automated assembly of space structures and began the development of a robotic system to assemble truss structures with equal length members [42]. Truss assembly plays an important role when one needs to build strong and light structures according to [43]. Truss allows constructing complex structures from a simple one. It considerably reduces the amount of space and weight in the launch vehicle. All the projects cited in the previous section use trusses.

A famous project using truss assembly is SpiderFab [44]. The vision of SpiderFab is to create a ”Satellite Chrysalis”, consisting of raw material in a compact and durable state. It is producing ”software DNA” assembly instructions that look like the web of a spider built with trusses to get an operational space system like a solar array for satellites, large antennas, or telescopes.

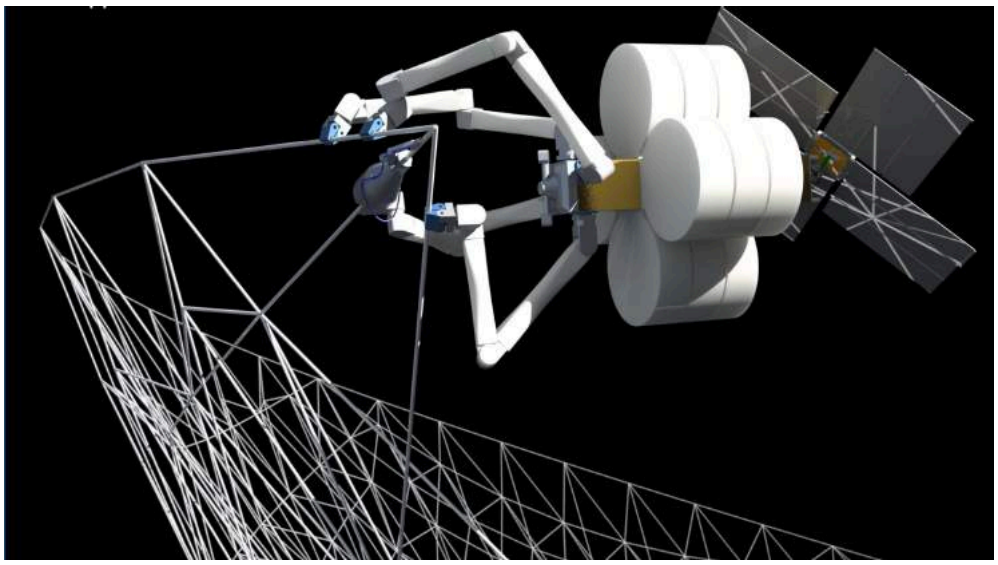


Fig. 1-11 SpiderFab bot building support structure

One very interesting technology for truss constructions is the ”Trusselator” developed by NASA for the project. The second generation can build up 50m of 50mm triangular cross-section trusses. With the ESPA payload (320kg) which is an adapter for launching secondary payloads on orbital launch vehicles, SpiderFab can build up to 7,000m first order Truss of 100mm thickness.

Fig. 1-12 Trusselator building triangular truss in continue

Once the robot has created a structural element and positioned it properly on the spacecraft structure, it will require means to bond the element to the structure. This bonding could be accomplished using welding, mechanical fasteners, adhesives, and other methods.

Another method is to have a connector such as the one developed by NASA [29]. The system is shown in Figure 1-13.

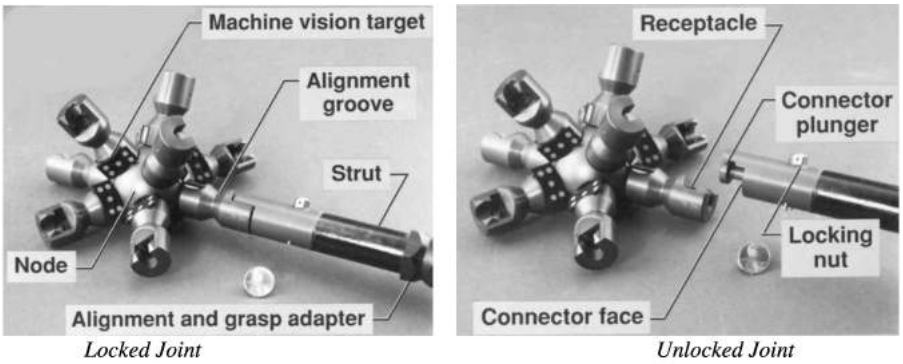


Fig. 1-13 Nodes developed by NASA for truss assembly

Other techniques involving 2 or more robotic arms are used for truss assembly [45], such as the one on Figure 1-14 it can make the robot have different degrees of freedom of operation and satisfy the operation function of the robot to the greatest extent by cooperating with three kinds of combined robots, namely, the handling robot, the transfer robot and the assembly robot, and fully considering the operation characteristics and requirements of the three kinds of robots [46].

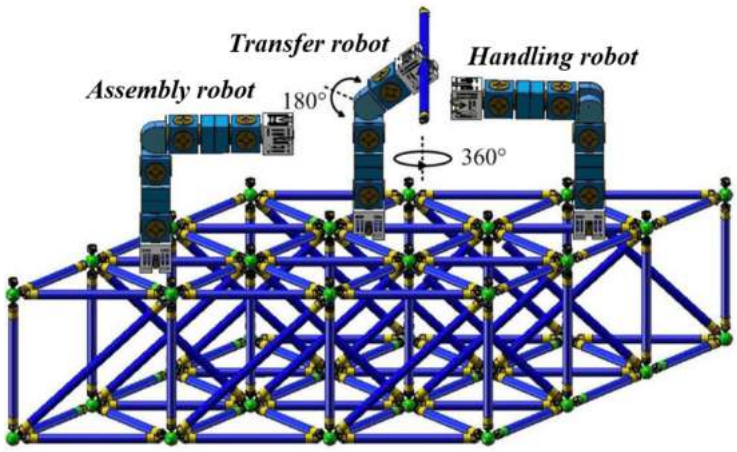


Fig. 1-14 Space cell robot assembling truss in orbit from [46]

1.2.3 Motion planning

Motion planning, also path planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. Robotic motion planning is a well-studied field at the intersection of optimal control, artificial intelligence, and applied mechanics. In their work [47] and [48] listed the current techniques for motion planning.

According to [48] motion planning can be divided into two categories, traditional algorithms, and Machine Learning (ML) algorithms. The different techniques are presented in Figure 1-15.

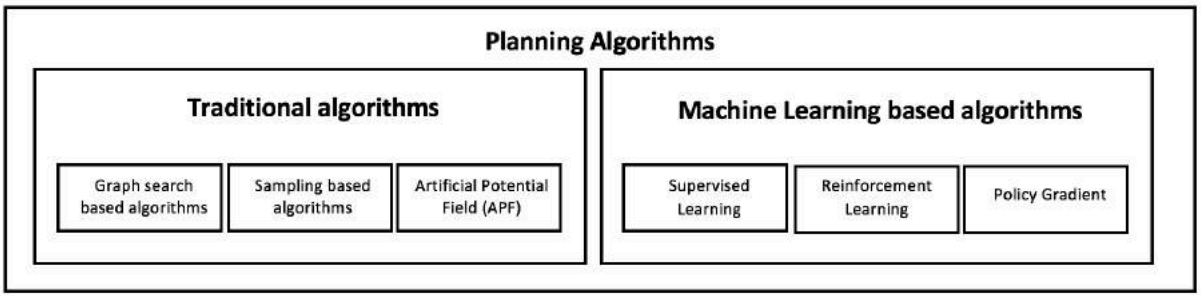


Fig. 1-15 Classification of planning algorithms

Traditional algorithms

Graph-search-based algorithms can be divided into the depth-first search, breadth-first search firstly introduced by Dijkstra’s algorithm [49], and best-first search. The depth-first search algorithm builds a search tree as deep and fast as possible from origin to destination until a proper path is found. The breadth-first search algorithm shares similarities with the depth-first search algorithm by building a search tree. The search tree in the breadth-first search algorithm, however, is accomplished by extending the tree as broad and quick as possible until a proper path is found.

Sampling-based algorithms randomly sample a fixed workspace to generate sub-optimal paths. The rapidly-exploring random tree (RRT) and the probabilistic roadmap method (PRM) are two algorithms that are commonly utilized in motion planning. RRT is an algorithm designed to efficiently search nonconvex, high-dimensional spaces by randomly building a space-filling tree. The tree is constructed incrementally from samples drawn randomly from the search space and is inherently biased to grow towards large unsearched areas of the problem [50], [51].

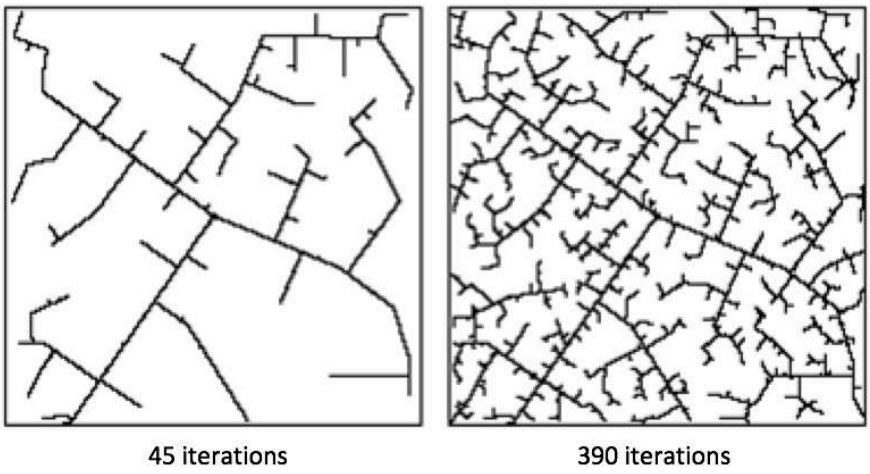


Fig. 1-16 Rapidly exploring random tree from [50]

The artificial potential field (APF) is based on the uptake of the robot to a particle, constrained to move in an APF [52]. The method is inspired by the electrical charges' concept introduced by Khatib in [53], whereby the objects in the configuration space, where the vehicle is traveling, are presumed to emit potential charges. The goal or target position is assumed to generate an attractive force that pulls the robot towards it. On the other hand, the obstacle creates a repulsive force the pushes the robot away as presented in Figure 1-17 [54].

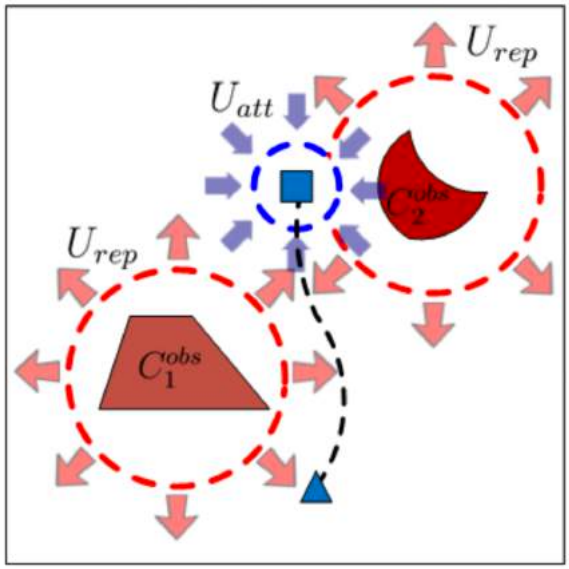


Fig. 1-17 Attractive and repulsive force field from [54]

Machine Learning algorithms

Supervised Learning is the machine learning task of learning a function that maps an input to an output based on example input-output pairs. It infers a function from labeled training data consisting of a set of training examples. One technique well-known is the support vector machine (SVM) [55] for classification. The basic principle of SVM is about drawing an optimal separating hyperplane between inputted data by training a maximum margin classifier. Other well-used techniques are Long-short term memory LSTM [56] which is a variant of a recurrent neural network, Monte Carlo tree search (MCTS) which is the combination of Monte-carlo method [57] and search tree [58].

Reinforcement Learning (RL) is a technique where the agent is learning from its experiments. The main policies in RL are Q-Learning where the agent updates a Q-Table through the Bellman equation at each step, and the second one is the Deep-Q-Learning, instead of a Q-Table a Neural Network is used at each episode, to that technique Experience Replay is usually used. These policies will be described in detail in the next section and Chapter 3.

Policy Gradient is a probability distribution $P\{a|s,\theta\} = \pi_\theta(a|s) = \pi(a|s,\theta)$ that is used to select action a in state s, where weight θ is a parameter matrix that is used as an approximation of policy $\pi(a|s)$. Policy gradient method [59] seeks an optimal policy and uses it to find optimal actions [60, 61].

1.2.4 Reinforcement Learning

To solve the motion planning issues RL will be used. Rather than programming in RL, the agent is learning from its experiments. The typical framing scenario is: an agent takes actions in an environment, which is interpreted into a reward and a representation of the state, which are fed back into the agent.

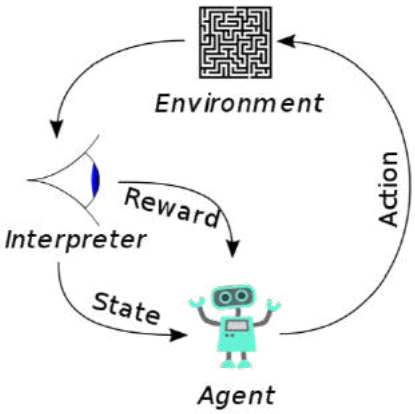


Fig. 1-18 Reinforcement Learning scenario

It has been growing rapidly, providing a wide variety of learning algorithms like Deep Q Learning, rather than using value iterations as in the Markov Decision Process (MDP) to determine the Q-values and find optimal Q-function, we alternatively use a function approximation to estimate optimal Q-function i.e. using Deep Neural Networks (DNN) [62-64]. On the following figure from [65], a UR5 is reaching a target in Mujoco using RL.

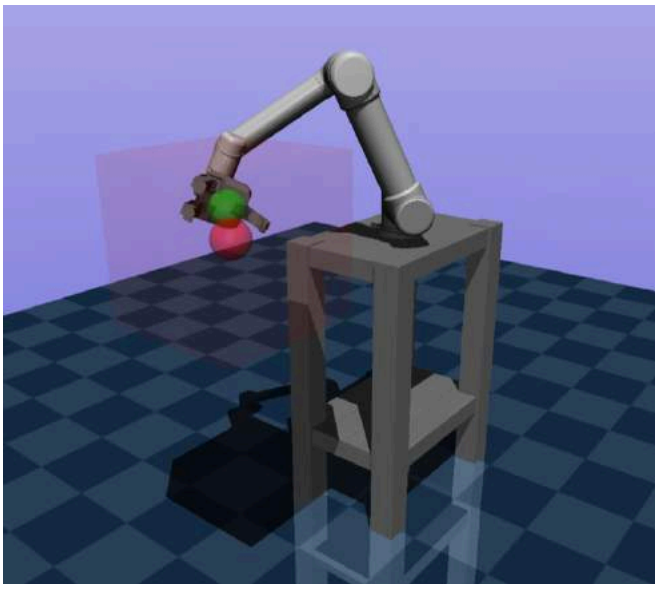
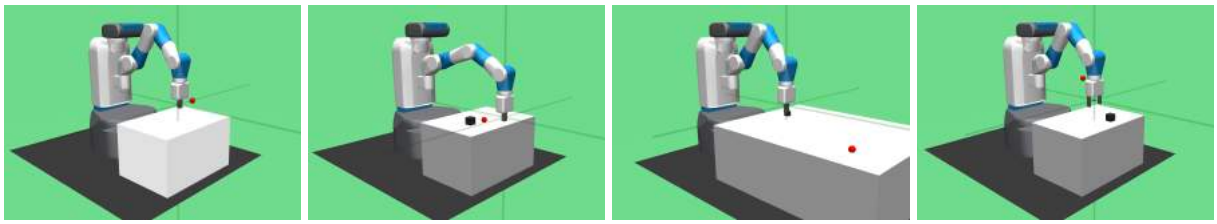


Fig. 1-19 UR5 reach task MuJoCo environment from [65]

More and more projects are starting to use Reinforcement Learning to program 6 Degree of Freedom (DOF) Robotic arms using deep learning [66-68], and Q-Learning [69, 70]. The tossing bot learns to pick objects in a box and throw them in another [71] with reinforcement learning. An interesting project is Fetch. Four different tasks are already proposed, FetchReach, FetchPush, FetchSlide, and FetchPickAndPlace. In all Fetch tasks, the goal is 3-dimensional and describes the desired position of the object (or the end-effector for reaching). Rewards are sparse and binary: The agent obtains a reward of 0 if the object is at the target location (within a tolerance of 5 cm) and −1 otherwise [72].



(a) FetchReach (b) FetchPush (c) FetchSlide (d) FetchPickPlace

Fig. 1-20 The Four Fetch environments

Great work has also been done by [73] in order to increase the use of Deep Reinforcement Learning (DRL) with real robots and reduce the gap between simulation and real-world robotics, they proposed an open-source toolkit: robo-gym. It's an environment of Openai Gym which is a tool for machine learning in robotics. The environments are created for two robots the UR10 and MiR100.

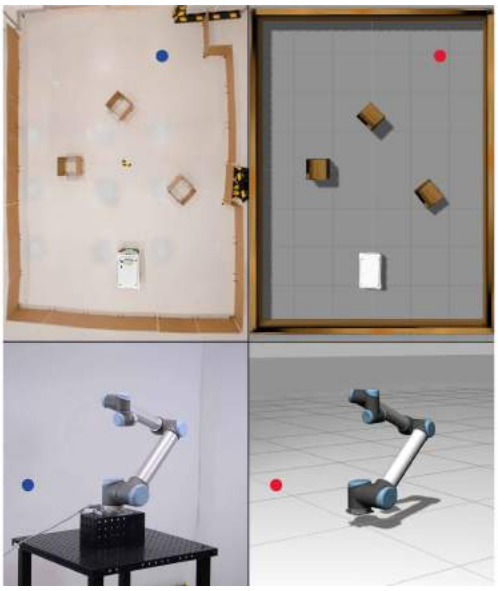


Fig. 1-21 Industrial robot use case scenarios (left: real environment, right simulation environment). Mobile navigation with obstacle avoidance of MiR100 on the top and end effector positioning of UR10 on the bottom.

The Shadow Robot Company developed an environment called "Smart Grasping Sandbox" with all usuals to get started with their gripper and the UR10. They used Machine learning to improve the grip and the accuracy of it. In the simulation, the robot grasps a ball, lifts it, and then shakes the hand to check if the ball has correctly been grasped then a reward is given and another experiment is started. The program is very convenient and rather easy to use and modify. It provides a variety of tools and libraries to get started quickly with playing with the robot: the robotic framework (ROS), the simulator (Gazebo), or also the planning libraries (MoveIt!).

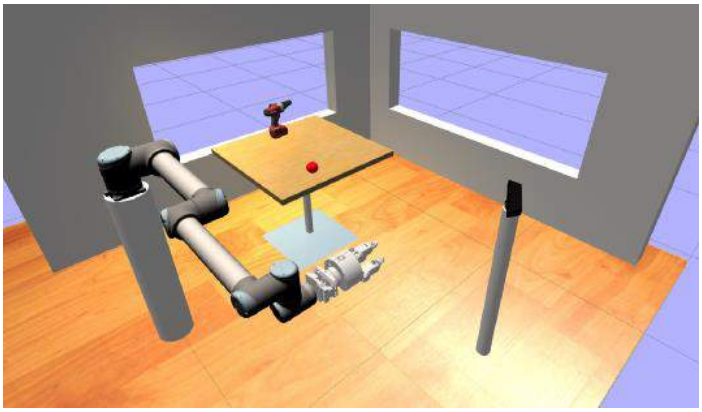


Fig. 1-22 Smart grasping sandbox in the simulator Gazebo

1.3 Objective of the study

The idea of this study is to use Reinforcement Learning for path planning and so improve the in-space assembly through this technique. Machine Learning has gained attention recently to control robot manipulators and a lot of work has been done to adapt it to the different fields of study such as Orbital Assembly. The complexity of this technique is compensated by the accuracy and autonomy it can produce. The main objectives here are:

- to provide a new design of bars to ease and gain in performance for future assembly of truss structures,
- to generate a program from scratch that enables to use UR10 with RL,
- to test the efficiency of the RL on a task similar to an assembly task which is picked and place.

The engineering problem has two concerns, first the algorithm for motion planning has to be resilient. It aims to be autonomous, robust to perturbations, and using few computing power. Then the truss elements have to be simplified for the robotic assembly, especially at the joint nodes and be fully adaptable to different kind of structures (telescopes, aerobrakes, solar array...).

1.4 Main Work and Organisation

The second Chapter is dealing with a new design of bars for truss structures to ease the assembly of the robot. The main idea is to think of a more efficient way to assemble mirrors for telescopes. Different forms of assembly (Triangles, Squares, Hexagons) are firstly compared to get the minimum number of bars and so lighten the structure. Then two types of bars are presented, one with the groove in which the mirrors are slide and the second one in form of a ”+” from the side. For this second design, magnets are used for the assembly of the mirror which also facilitates the assembly for the robot. The bars are bind together with magnetic balls instead of rather complex joint nodes.

In Chapter 3 the Kinematic of the UR10 is described. Then it deals with path planning which is made using Reinforcement Learning. Tools to understand machine learning are detailed. The Markov Decision Process (MDP) and the Bellman equation for updating the Q values are the two main techniques used in the reinforcement. First, the Q Learning Policy is explained and then the Deep Q Learning as well as Neural Networks.

Then in Chapter 4, the simulation is detailed. The choice of the software for the study is explained. During the first steps of the study different environments of work were tested before

to find the optimal one for the task. The program is made from scratch and the different part of a RL program is explained in this chapter. The main objectives in the experiments are, first to reach a target and compare the efficiency of the Policies for this task and then to pick and place an object.

Finally, a conclusion is made and a discussion is made about the study and the future works. The figure below is a summary of the thesis work.

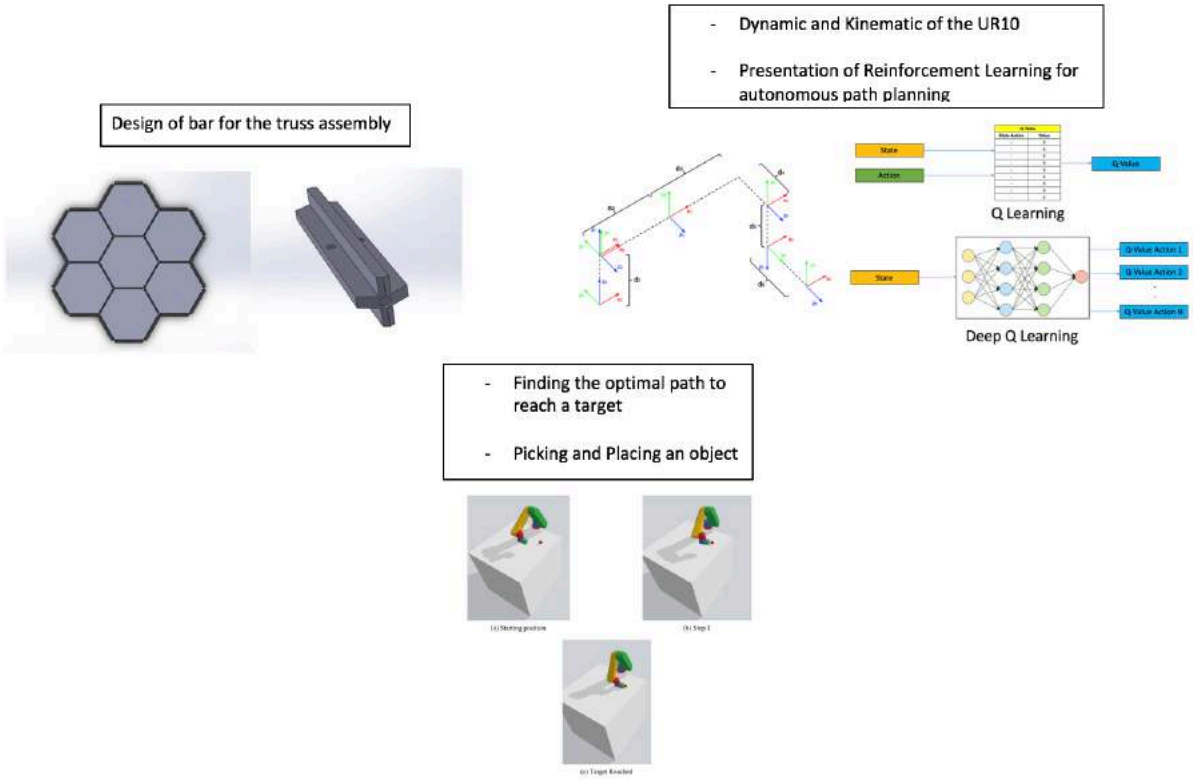


Fig. 1-23 Graphical summary of the thesis

2 Truss Assembly

The joint at the assembly nodes is a crucial part. For a robotic assembly, it is the part that requires the most advanced techniques but it could be facilitated with a well-engineered process. In this Chapter, different forms will be discussed for the assembly of mirrors and then magnetic joints will be presented.

2.1 Assembly elements

Telescopes are the most common large structures that require on-orbit assembly. The accuracy of a telescope depends in part on the size of the mirrors. For on-orbit assembly, the main constraint is the payload that the rocket can send into space. In this section, different forms will be compared for the assembly of mirrors for telescopes. A surface of $25m^2$ is taken as a reference and size of $1.48m$ for the bars.

1) Squares

For a total surface of 26.2 m^2 of mirrors, the assembly with squares requires 32 bars of 1.48m.

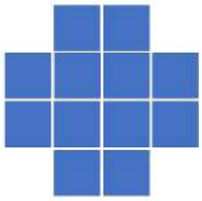


Fig. 2-1 Squares

2) Triangles

For a total surface of 22.7 m^2 of mirrors, the assembly with triangles requires 42 bars of 1.48m.

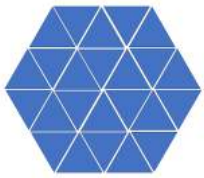


Fig. 2-2 Triangles

3) Hexagons

For a total surface of 35.8 m^2 of mirrors the assembly, squares require 72 bars of 0.74m which is 36 bars of 1.48m.

Fig. 2-3 Hexagons

4) Comparison

The Hexagon is the form that allows the minimum of bars compare to the surface that it could offer for the mirrors. On the Figure 2-4 the red circle represents a surface of $25m^2$ So that form will be chosen for the rest of the assembly.

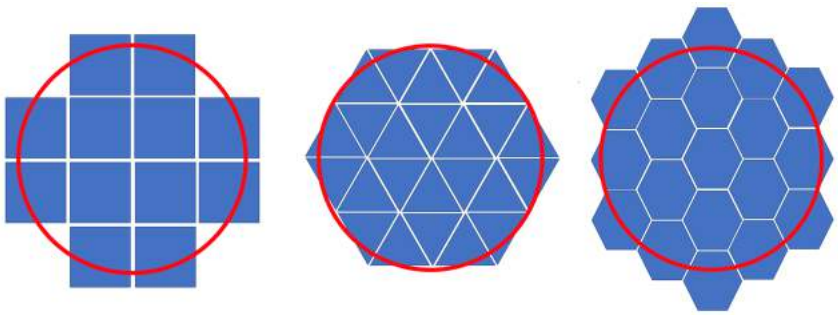


Fig. 2-4 Comparison of the forms

Tab. 2-1 Comparison of the assembly forms

	Number of bars (1.48m)	**Area of mirrors**
Squares	32	$26.2m^2$
Triangles	42	$22.7m^2$
Hexagons	36	$35.8m^2$

2.2 Magnetic assembly

The assembly of truss structures is made with a rather complicated mechanism such as the one presented by NASA in [29] in Figure 1-13. Some other techniques require two robotic arms as in Figure 1-14 from [46]. The magnets used will be first presented then the concept of a new design for truss assembly bars. The plans of the bars used in this section are detailed in the appendix.

2.2.1 Neodymium magnets

Neodymium is a chemical element with the symbol Nd and atomic number 60. Neodymium belongs to the lanthanide series and is a rare-earth element. To make the neodymium magnets it is alloyed with iron, which is a ferromagnet. Neodymium magnets (actually an alloy, Nd2Fe14B) are the strongest permanent magnets known. A neodymium magnet of a few grams can lift a thousand times its weight.

Neodymium magnets have a very high coercive force, and there will be no demagnetization and magnetic changes under the natural environment and general magnetic field conditions. Assuming under an appropriate environment, even after a long period of use, the magnetic performance of the magnet will not be greatly reduced. Therefore, in practical applications, we often ignore the influence of time on the magnetic performance of neodymium magnets. The pictures below are presented the magnets used for the assembly.

Fig. 2-5 Magnets used for the assembly

Usually, the magnets are named with a letter (N, M, H, SH, UH our EH) and a number (40, 42, 45...). The letter gives information about the maximum temperature of use. The number corresponds to the maximum rate of energy. In our case, we choose a magnet which is N52 so the maximum temperature of use is 80° and the energy of magnetization is the maximum we could find. The magnet can load 0.2 kg which is more than enough for the size of the bars.

2.2.2 Presentation of the grooved bars

The bars are 3D printed with a groove for the assembly of the mirror. Neodymium cubes are inserted in the center of the bars to enable connection with the other bars through a magnetic ball.

(a) Model view of the grooved bar

(b) Joint nodes

Fig. 2-6 3D models of the grooved bars

The following figure presents the bars during the printing. A 3D printer using resin has been used. A first model was made using polylactic acid (PLA) but the final result was not enough accurate for the assembly. In this method, the pieces are printed in a resin bath by solidifying the resin layer by layer with a UV lamp. This technique allows more continuous elements.

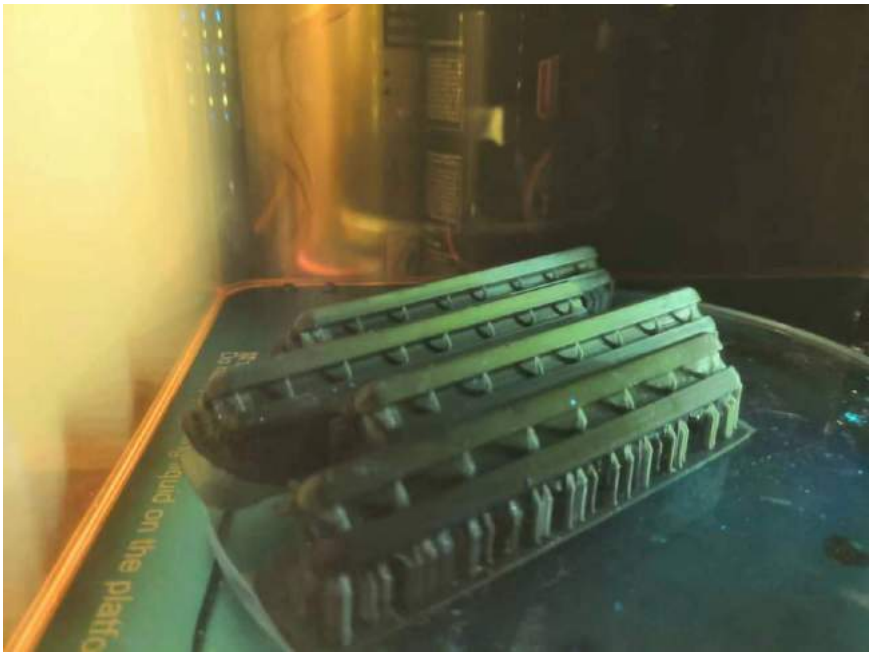


Fig. 2-7 3D printing of the bars with resin

In the following figures, the different views of the bars are presented.

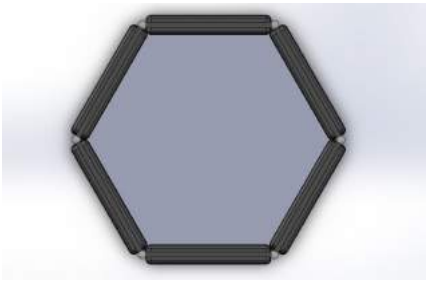
(a) Side view of the grooved bar

(b) Top view of the grooved bar

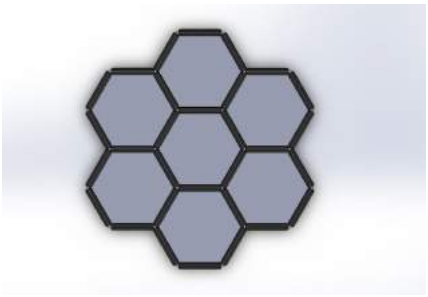
(c) Assembly of two bars

Fig. 2-8 Different views of the grooved bars

The 3D model of the expected final assembly with the hexagonal mirrors is presented in the following figures.



(a) One mirror



(b) Mirrors assembly of the grooved bars

Fig. 2-9 Model of the grooved bars assembly

2.2.3 Presentation of the flat bars

This second design of the bars for the truss is in a form of a ”+”. This form uses less resin than the grooved bars. A magnet is placed in the middle of the flat bar. The magnet is a small

(a) Side view of the flat bar

(b) Top view of the flat bar

(c) Assembly of two bars

Fig. 2-11 Different views of the flat bars

cylinder with a diameter of $3mm$ and a high of $1.5mm$. The bars are $2mm$ thick so the magnet easily fits in it. The same magnet is glued on the back of the 6 sides of the mirror.

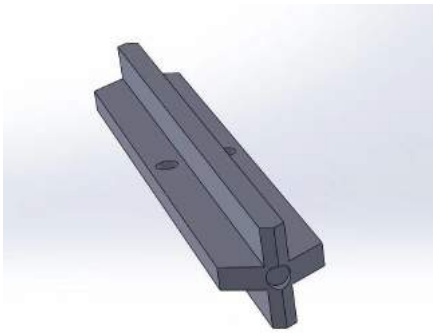


Fig. 2-10 3D model of the Flat bars

The advantage of this form is that it's symmetric so there is no need for the robot to take care of the orientation for the assembly. The assembly is simplified compared to the first design. The elements are just fixed together with magnets and the mirror is simply posed on top of the hexagon, no need to slide the mirror into the slit.

The different views of the bars are presented in Figure 2-11 and the model of the assembly in Figure 2-12.

(a) One mirror

(b) Assembly of the mirrors with flat bars

Fig. 2-12 Model of the flat bars assembly

2.2.4 Final assembly

For the grooved bars, at the beginning the robot assembles the four first bars, then the mirror is slide on the grooves, and finally, the contour of the closed with the two last bars.

Fig. 2-13 Sequence of assembly for the grooved bars

The assembly of one mirror is $14.8cm$ in length, this measure is made from one side to the opposite one. With these grooved bars, the thickness of the bars is to be taken into account for the total assembly. For the 7 mirrors assembly, the size from bottom to top is $42.5cm$ length,

compared to $40.9cm$ for the theoretical assembly on the software. It represents an error of 3.9%. For a telescope, this design takes a part of the mirror's reflection due to the grooves.

(a) Assembly of the 6 bars

(b) Back view of a mirror with the magnets

(c) Final assembly of one mirror

Fig. 2-14 Sequence of assembly for the flat bars

For the second design, the assembly is much faster, first the 6 bars and assemble and then the mirror is fixed just by posing it. The magnets allow the mirror to get in the right position, it plays the same role as the grooves.

Fig. 2-15 Final assembly of 7 mirrors with the second design

The average size measured of one mirror is $13.8cm$ and the total assembly $41.5cm$ compare

to $13.6cm$ and $40.2cm$ for the assembly on the software. The error is 3.2% for this assembly.

2.2.5 Comparisons of the designs

The second concept of bars is much more relevant than the first one. The flat bars are lighter, weighing about 7g compared to 10g for the grooved bars. The total size of the assembly is $42.5cm$ and $41.5cm$ respectively for the grooved bars and the flat bars.

Another important point is that the flat bars take less space in the assembly between the mirrors. It is $1cm$ smaller but the surface of the mirrors is greater, the total surface is exploited. With the grooved bars, the surface lost is $69.3cm^2$, due to the $0.3cm$ grooves, for a total surface of mirrors of $1022cm^2$. It represents 14.7% of the total surface of mirrors. The grooved bars are also more difficult to assemble with the mirrors for the robot because they have to be slide on them.

The comparison of the designs is summarized in the following tabular.

Tab. 2-2 Comparison of the Policies

	Grooved bars	**Flat bars**
System to fix the mirror	Slide in Grooves	Magnets
Steps for assembly	3	2
Weight	10g	7g
Theoretical size	$40.9cm$	$40.2cm$
Assembly size	$42.5cm$	$41.5cm$
Assembly error	3.9%	3.2%
Lost surface	$69.3cm^2$	None

This technique using magnets ease the assembly. Compare to [29] where the nodes are used to position the receptacles to form a specific structural geometry. The receptacles form half of the joint and include an alignment groove that is grasped by the end-effector receptacle fingers to fix the end-effector position during strut installation. The other half of the joint is affixed to the strut and includes a locking nut that is rotated by a nut driver on the end-effector as shown in Figure 1-13.

2.3 Large structure assembly

These designs of truss bars address a new pathway for large structure assembly. Instead of unfolding telescopes such as the James Webb telescopes, this bars using magnets allows to build directly the structure into space. The assembly of mirrors is made close to a telescope

shape in order to prove the feasibility of this project for this type of large structures. Finally the path planning is eased with a system using magnets. The grasping part can be removed from the problems because the bars are directly grasped with a magnetized end effector. The same remark can be applied to the joint nodes which made by magnets as well. Third point is the orientation of the bars which is also facilitated the path planning, the manipulator doesn't need to take the orientation of the bars for the assembly.

2.4 Summary

The best design as it has been shown is the second one using flat bars and magnets for the fixation of the mirror. This design ease the assembly for the robot. First, the grasping is made using magnetization so no need for special grasping. There is no lost surface and the bars are lighter which is a very important consideration for an on-orbit assembly where the payload is very costly. Then because of the symmetry of the bars, the assembly needs to care only about the orientation of the element on the z ax. Finally, this design can easily be printed in space same as the "Trusselator" which considerably reduces the payload for big structures assembly.

3 Autonomous path planning

As said before motion planning, also path planning is a computational problem to find a sequence of valid configurations that moves the object from the source to destination. In this Chapter, the kinematic of the Universal Robot (UR) is firstly presented. Then the path planning is made using reinforcement learning which is a technique that perfectly fits robotic when autonomy and adaptability are expected.

3.1 Robotic Kinematic

Robotic motion planning deals with robot kinematics in low dimensions, with constraints such as collision avoidance and self-intersection. The kinematic state space is often referred to as the configuration C-space, equal to the number of DoF. Robot kinematics governs how linkages move, restricting the feasible configuration space due to collision and linkage geometry. In some cases, continuous planning for robot kinematics can be turned into decisions of a discrete set of actions, resulting in discrete planning. The study of the Kinematic equation will be conducted to have a description of the movement of the robotic arm.

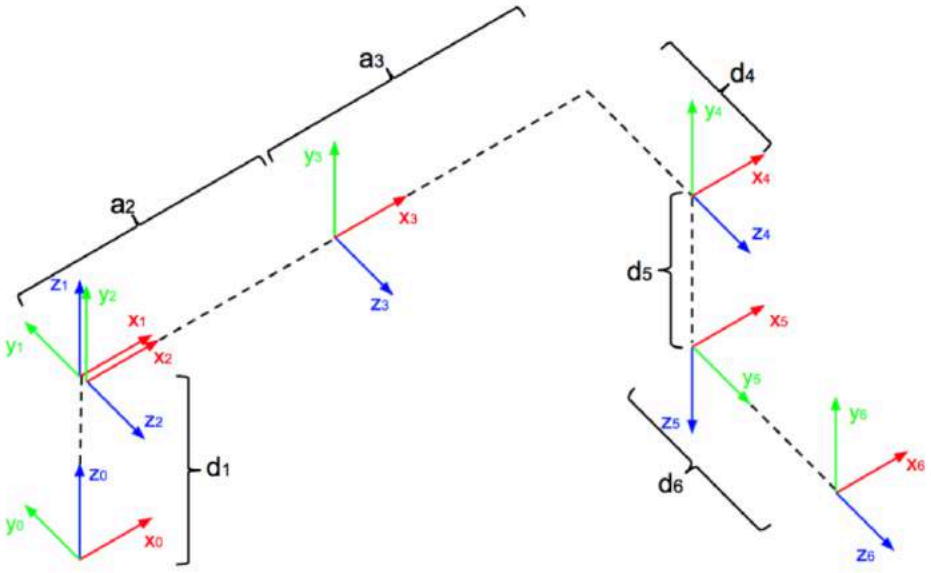


Fig. 3-1 Coordinate frames for UR arm. Joints rotate around the z-axes and are pictured at θ_i=0 for 1≤i≤6

3.1.1 Forward Kinematic

We start by the forward Kinematic that allows us to describe the position of the end effector as a function of joint angles:

$$B_6(\theta_1, \theta_2, \theta_3, \theta_4, \theta_5, \theta_6) = B_1(\theta_1)_1 B_2(\theta_2)_2 B_3(\theta_3)_3 B_4(\theta_4)_4 B_5(\theta_5)_5 B_6(\theta_6)_6 = \tag{3-1}$$

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-2}$$

where $B_n(\theta_n)_n$ describes the desired joint angles of the robot. The calculation is detailed in [74], we finally obtain the Denavit–Hartenberg parameters:

Tab. 3-1 Denavit-Hartenberg parameters for the UR10

Kinematic	θ [rad]	a[m]	d[m]	α [rad]	**Dynamics**	Mass [kg]	Center of Mass [m]
Joint 1	0	0	0.1273	$\pi/2$	**Link 1**	7.1	[0.021, 0.000, 0.027]
Joint 2	0	-0.612	0	0	**Link 2**	12.7	[0.38, 0.000, 0.158]
Joint 3	0	-0.5723	0	0	**Link 3**	4.27	[0.24, 0.000, 0.068]
Joint 4	0	0	0.1639	$\pi/2$	**Link 4**	2	[0.000, 0.007, 0.018]
Joint 5	0	0	0.1157	-$\pi/2$	**Link 5**	2	[0.000, 0.007, 0.018]
Joint 6	0	0	0.0922	0	**Link 6**	0.365	[0.000, 0.000, -0.026]

All results presented in the table are used to create the Unified Robot Description File (URDF) later in this study for the experiments and simulations.

3.1.2 Inverse Kinematic

The analytic inverse kinematics problem is to find the set of joint configurations $Q = q_i$ where $q_i = (\theta_1^i, ..., \theta_6^i) \in [0, 2\pi]$ that satisfies :

$$B_6(\theta_1^i, \theta_2^i, \theta_3^i, \theta_4^i, \theta_5^i, \theta_6^i) = (B_6^d) = \tag{3-3}$$

$$\begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3-4}$$

Where (B_6^d) describes the desired position and orientation of the final link. The idea here is then the opposite of the forward kinematic, we want to know the joint configuration for a given position. The analytic calculation is detailed also in [74]. It can also be calculated with Matlab using the Robotic System toolbox and the function robotics $InverseKinematics$.

Given the desired SE3 pose of an end-effector, the inverse kinematics solver computes the joint configurations that realize the desired end-effector pose. The end effector needs some kind of calibration to calculate the Inverse Kinematic.

```
Robot: (7 bodies)

 Idx     Body Name     Joint Name     Joint Type     Parent Name(Idx)   Children Name(s)
 ---     ---------     ----------     ----------     ----------------   ----------------
   1         link1         joint1       revolute             base(0)   link2(2)
   2         link2         joint2       revolute           link1(1)   link3(3)
   3         link3         joint3       revolute           link2(2)   link4(4)
   4         link4         joint4       revolute           link3(3)   link5(5)
   5         link5         joint5       revolute           link4(4)   link6(6)
   6         link6         joint6       revolute           link5(5)   tool(7)
   7          tool           fix1          fixed           link6(6)
```

Fig. 3-2 Inverse kinematic table

In the following case a loop through the trajectory of points to trace the circle. Call the ik object for each point to generate the joint configuration that achieves the end-effector position. Store the configurations to use later. The final result is the end effector following the circle.

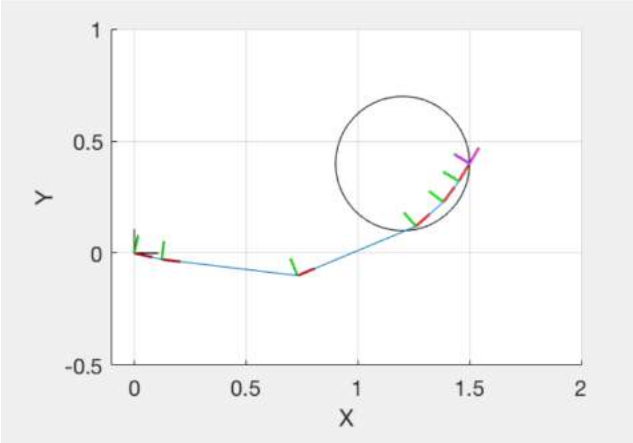


Fig. 3-3 Inverse Kinematic solver on $Matlab$

This technique is the most used one to control the position of robot manipulators. However, it is not adapted to obstacle avoidance, assembly, or any task that requires the robot to stand alone.

The kinematic is required for the rest of the study, it's the base for the comprehension of the robotic arm. This work will be reused for the simulation.

3.2 Reinforcement Learning Theory

Machine Learning could be considered as a new branch of optimal control theory. There are different types of Machine Learning: Supervised Learning, Unsupervised Learning, and Reinforcement Learning which are the most suitable for robotic. This section presents the principle of Reinforcement Learning and the tools and mathematical theories needed for it.

3.2.1 Overall Principle

The main concepts of RL are: the **environment**, the **state**, **actions**, **reward** and **penalties** and the **policy**. Let's explain it with a concrete example. In this thesis we basically want to teach a robotic arm to pick an object on a table and place it in an other place to build a structure.

- We already have the notion of **agent** which is the robotic arm itself.
- Then the **environment** which could be reduced to the room in which the robot is, the table, and the objects. This is where the robot will evolve.
- The first **state** would be the robot at the starting position then the other state will be the end effector that "grasp" the object with magnetization and the final state would be the object placed at the desired position on the table.
- The transition between each state is made through **actions**. Here actions are made by moving the different joints of the 6 DOF arm.
- The **reward** could be for example +1 if the robot goes closer to the assembly element **penalties** are given when it goes in the wrong directions and too far from the target.
- Finally the **policy** is the strategy of choosing an action given a state in expectation of better outcomes.

Reinforcement Learning lies between the spectrum of Supervised Learning and Unsupervised Learning, and there are a few important things to note:

- Being greedy doesn't always work
  Some things are easy to do for instant gratification, and some things provide long-term rewards. The goal is to not be greedy by looking for quick immediate rewards, but instead to optimize for maximum rewards over the whole training.

- Sequence matters in Reinforcement Learning
  The reward agent does not just depend on the current state, but the entire history of states. Unlike supervised and unsupervised learning, time is important here.

3.2.2 Markov decision Process (MDP)

MDPs are meant to be a straightforward framing of the problem of learning from interaction to achieve a goal. The agent and the environment interact continually, the agent selecting actions and the environment responding to these actions and presenting new situations to the agent. Formally, an MDP is used to describe an environment for reinforcement learning, where the environment is fully observable. Almost all RL problems can be formalized as MDPs.

1) Markov Property

The $Markov$ propriety states, "The future is independent of the past given the present." In mathematical terms, a state S_t has the Markov property, if and only if:

$$P[S_{t+1}|S_t] = P[S_{t+1}|S_1, ..., S_t] \tag{3-5}$$

the state captures all relevant information from history.

For a Markov state S and successor state S', the state transition probability function is defined by,

$$\mathcal{P}'_{ss} = \mathbb{P}[S_{t+1} = s'|S_t = s] \tag{3-6}$$

It' s a probability distribution over the next possible successor states, given the current state, i.e. the agent is in some state, there is a probability to go to the first state, and another probability to go to the second state, and so on.

2) Markov Process

A Markov process is a memory-less random process, i.e. a sequence of random states S_1, S_2, ... with the Markov property. A Markov process or Markov chain is a tuple (S, P) on state-space S, and transition function P. The dynamics of the system can be defined by these two components S and P. It's a sequence of states (or "episodes").

3) Markov Reward Process

A Markov Reward Process or an MRP is a Markov process with value judgment, saying how much reward accumulated through some particular sequence that is sampled. An MRP is a tuple (S, P, R, γ) where S is a finite state space, P are the state transition probability function, R is a reward function where,

$$R_s = \mathbb{E}[R_{t+1}|S_t = S], \tag{3-7}$$

it says how much immediate reward it is expected to get from state S at the moment.

There is the notion of the return G_t, which is the total discounted rewards from time step t. The goal is to maximize this return,

$$G_t = R_{t+1} + \gamma R_{t+2} + \gamma^2 R_{t+3} + ... = \sum_{k=0}^{\infty} \gamma^k R_{t+k+1} \tag{3-8}$$

γ is a discount factor, where $\gamma \in [0, 1]$. It informs the agent of how much it should care about rewards now to rewards in the future. If ($\gamma = 0$), that means the agent only cares about the first reward. If ($\gamma = 1$), that means it cares about all future rewards. The goal is to maximize the total rewards.

The value function informs the agent of how much reward to expect if it takes a particular action in a particular state i.e. how good is it to be in a particular state, and how good is it to take a particular action.

The state-value function of an MRP is the expected return starting from state s,

$$v(s) = \mathbb{E}[G_t|S_t = S], \tag{3-9}$$

3.2.3 Bellman equation

An important point is the policy π. It is a distribution over actions given states. A policy fully defines the behavior of an agent,

$$\pi(a|s) = \mathbb{P}[A_t = a|S_t = S], \tag{3-10}$$

There are many different policies for reinforcement learning, Q-Learning and Deep Q Learning will be described later in the study.

1) Bellman Expectation Equation

The state-value function can be decomposed into immediate reward R_{t+1}, and discounted value of successor state $\gamma V\pi(S_t+1)$ on policy π,

$$v_\pi(s) = \mathbb{E}[R_{t+1} + \gamma v_\pi(S_{t+1})|S_t = s], \tag{3-11}$$

It gives the agent a quantitative result on how good the original state was by adding the immediate reward for the step and the value it ended up.

Similarly, the action-value function can be decomposed,

$$q_\pi(s,a) = \mathbb{E}_\pi[R_{t+1} + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t = s, A_t = a] \tag{3-12}$$

The sum of the immediate reward (R_{t+1}) for the action a and the action-value ($\gamma q_\pi(S_{t+1}, A_{t+1})$ tells how good it was to take that action from that particular state.

Since there are multiple actions from one state S, and the policy defines a probability distribution over those actions. The average gives the Bellman expectation equation,

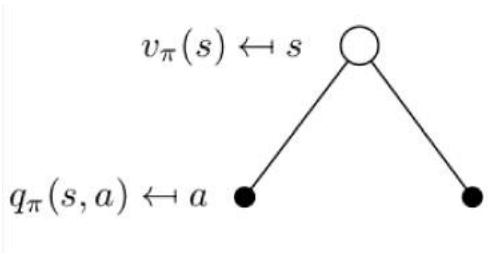


Fig. 3-4 Graphical state value function

$$v_\pi(s) = \sum \pi(a|s) q_\pi(s,a) \tag{3-13}$$

From a particular state S, there are multiple actions. There is a probability of taking the first action and another probability of taking the second action and so on. This probability distribution is defined by a policy π. The state value function is then the sum of probabilities of the actions under the policy π as the equation (3-13) translate.

When the action value q is reached for the action taken, it tells how good it is to take that action from that state. Averaging over possible action-values tells how good it is to be in state S.

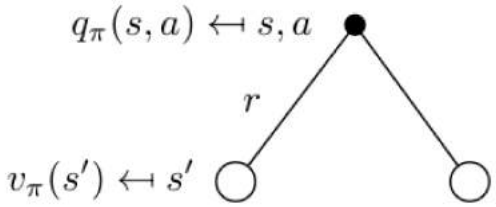


Fig. 3-5 Graphical action value function

$$q_\pi(s,a) = \mathcal{R}_s + \gamma \sum \mathcal{P}_{ss'} v_\pi(s') \tag{3-14}$$

The next step is to know the value of being in the next state following the policy onwards.

So the average over possible things that might happen is taken, i.e. possible successor states the agent might land in, meaning multiplying each state value on policy π the agent might land in by the probability that the agent land in it.
Remember $V_\pi(s)$ tells how good it is to be in a particular state, and $q_\pi(s,a)$ tells how good it is to take a particular action from a given state.

So the Bellman expectation equation for $V_\pi(s)$ is,

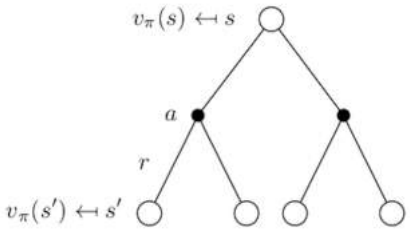


Fig. 3-6 Graphical Bellman state value equation

$$v_\pi(s) = \sum \pi(a|s)(\mathcal{R}_s + \gamma \sum \mathcal{P}_{ss'} v_\pi(s')) \tag{3-15}$$

And the Bellman expectation equation for $q_\pi(s,a)$ is,

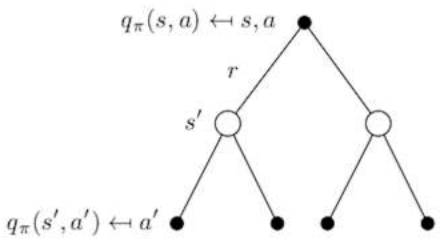


Fig. 3-7 Graphical Bellman action value equation

$$q_\pi(s,a) = \mathcal{R}_s + \gamma \sum \mathcal{P}_{ss'} \sum \pi(a'|s') q_\pi(s',a') \tag{3-16}$$

In practice, the most used equation is the equation of the action value (3-16) and the different forms of it. It is used to give the new value of the next state.

2) Optimal Value Function

The optimal state-value function $V^*(s)$ is the maximum value function over all policies.

$$v^*(s) = \max_\pi v_\pi(s) \tag{3-17}$$

It' s the best possible solution for an MDP. Of all kinds of different policies that could be followed in a Markov chain. The goal is the maximum possible rewards that we can extract from an MDP.

The optimal action-value function $q^*(s,a)$ is the maximum action-value function over all policies.

$$q^*(s,a) = \max_\pi q_\pi(s,a) \tag{3-18}$$

For the state-action pair (s, a), this function gives the expected return for taking action an in state S, and thereafter following an optimal policy, i.e. the maximum amount of rewards extracted starting in state S, and taking action a.

If $q^*(s,a)$ is known, then the problem is solved. It tells the right action to take. The optimal value function specifies the best possible performance in the MDP. An MDP is solved when the optimal value function is known. For example in Q Learning the optimal value function is found when the Q table is completed.

3.3 Q-Learning

Essentially, Q-learning lets the agent use the environment's rewards to learn, over time, the best action to take in a given state.

Q-values are initialized to an arbitrary value, and as the agent exposes itself to the environment and receives different rewards by executing different actions, the Q-values are updated using the equation:

$$Q(s,a) = r(s,a) + \gamma \max_a Q(s',a) \tag{3-19}$$

The above equation states that the Q-value yielded from being at state s and performing action a is the immediate reward $r(s,a)$ plus the highest Q-value possible from the next state s'. γ here is the discount factor that controls the contribution of rewards further in the future.

$Q(s',a)$ again depends on $Q(s'',a)$ which will then have a coefficient of gamma squared. So, the Q-value depends on Q-values of future states as shown here:

$$Q(s,a) \rightarrow \gamma Q(s',a) + \gamma^2 Q(s'',a)...\gamma^n Q(s''^{...n},a) \tag{3-20}$$

Adjusting the value of gamma will diminish or increase the contribution of future rewards.

Since this is a recursive equation, we can start with making arbitrary assumptions for all q-values. With experience, it will converge to the optimal policy. In practical situations, this is implemented as an update:

$$Q(s,a) \leftarrow \gamma Q(S_t,A_t) + \alpha[R_{t+1} + \gamma \max_a Q(S_{t+1},a) - Q(S_t,A_t)] \tag{3-21}$$

where alpha is the learning rate or step size. This determines to what extent newly acquired information overrides old information.

3.4 Deep Q learning

Deep Q Learning is used for continuous environments compare to Q Learning which is used for discrete and rather small environments.

3.4.1 Feedforward neural networks

One of the key features of deep learning is that the computational models, called deep neural networks are composed of multiple processing layers and can learn representations of data with multiple levels of abstraction. The learning of a deep neural network (DNN) is made by using the backpropagation method to indicate how the internal parameters should be changed. On the other hand, the prediction of the output is calculated by using the forward propagation

method: the data is fed to the input layer, the neurons do a linear transformation on the input by the weights and biases, the activation function transforms the linear function into a nonlinear function, the information moves from layer to layer, and finally output the result.

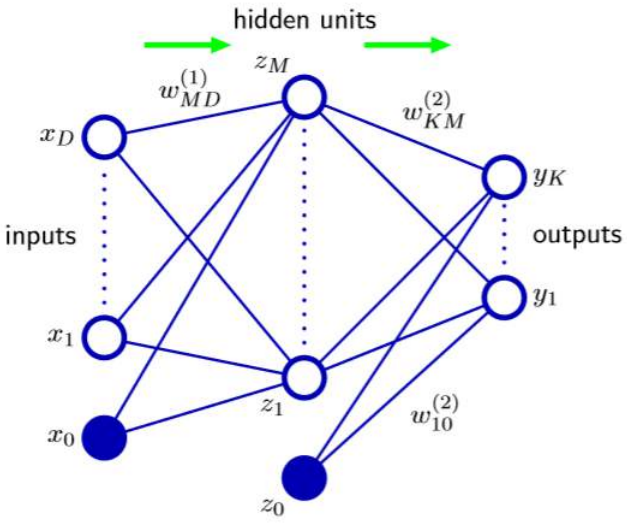


Fig. 3-8 Neural Network

In Figure 3-8, a generic feedforward Neural Network. The network has three layers, an input layer with three input units, a hidden layer, and an output layer consisting of two output units. The number of circles in each layer indicates the dimensions of the corresponding layers. The circles represent neurons of the network, and arrows represent the connections and data between the neurons of the network.

The deep neural network is a kind of nonlinear function that usually contains a large number of parameters. Formally, the feedforward neural network can be expressed as a function

$$\hat{y} = f_\theta(x) \tag{3-22}$$

where **x** is the input vector, θ is the parameter vector of the network, and y is the output values. The goal of training a deep neural network is to find the optimal parameters by minimizing a defined loss function whose gradients concerning the parameters are calculated with forwarding and backward propagation.

The loss function for our study is the mean squared error of the predicted $QValue$ and the target Q^*. This is a regression problem.

$$Loss = \frac{1}{n}\sum_{i=1}^{n}(QValue_i - Q_i^*)^2 \tag{3-23}$$

3.4.2 Error Backpropagation

Error backpropagation is an efficient technique to evaluate the gradient of an error function $E(w)$ for a feed-forward neural network. It decreases the complexity of a model. The formula provides is the following one,

$$\delta_j = h'(a_j) \sum_k w_{kj} \delta_k \tag{3-24}$$

This tells us that the value of δ for a particular hidden unit can be obtained by propagating the δ's backward from units higher up in the network,

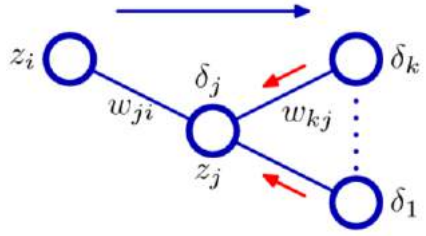


Fig. 3-9 Illustration of backpropagation

Illustration 3-9 of the calculation of δ_j for hidden unit j by backpropagation of the δ's from those units k to which unit j sends connections. The blue arrow denotes the direction of information flow during forwarding propagation, and the red arrows indicate the backward propagation of error information.

Thus backpropagation procedure can be applied as follow,

1. Apply an input vector x_n to the network and forward propagate through the network to find the activation of all the hidden and output units.
2. Evaluate the δ_k for all the output units
3. Backpropagate the δ's to obtain δ_j for each hidden unit in the network.
4. Evaluate the required derivatives.

3.4.3 Weights and Bias

Weights and biases (commonly referred to as w and b) are the learnable parameters of a machine learning model. Neurons are the basic units of a neural network. In a DNN, each neuron in a layer is connected to each neuron in the next layer. When the inputs are transmitted between neurons, the weights are applied to the inputs along with the bias.

$$\sum(weights \times input) + bias \tag{3-25}$$

Weights control the signal (or the strength of the connection) between two neurons. In other words, a weight decides how much influence the input will have on the output.
Biases, which are constant, are an additional input into the next layer that will always have the value of 1. Bias units are not influenced by the previous layer (they do not have any incoming connections) but they do have outgoing connections with their weights. The bias unit guarantees that even when all the inputs are zeros there will still be activation in the neuron.

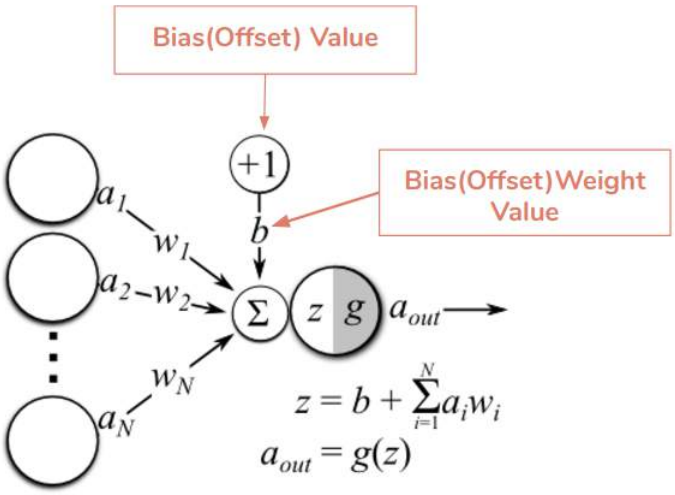


Fig. 3-10 Weights and Biases

3.4.4 Deep Q Network

In Deep Q-learning, a neural network is used to approximate the Q-value function. The state is given as the input and the Q-value of all possible actions is generated as the output. The comparison between Q-learning and deep Q-learning is illustrated below:

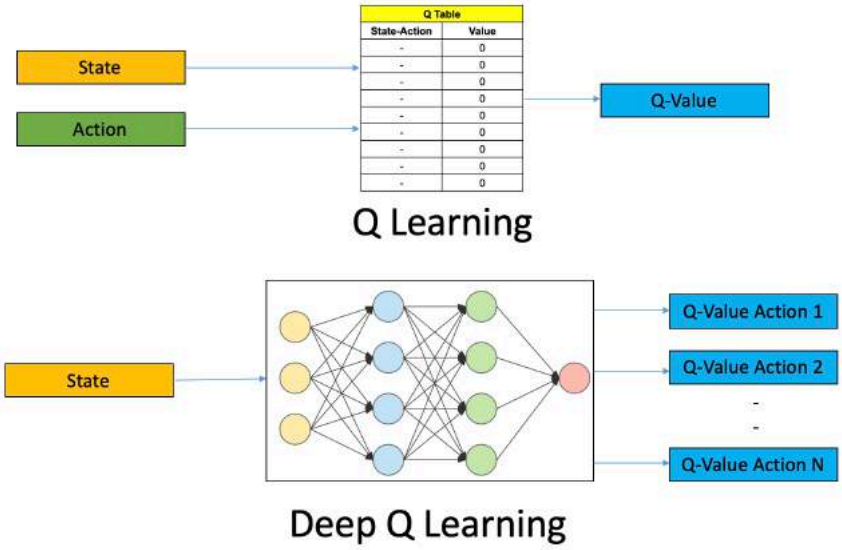


Fig. 3-11 Comparison between Q-learning and deep Q-learning

The steps involved in reinforcement learning using deep-Q-learning networks are:

- All the experience is stored by the user in memory
- The next action is determined by the maximum output of the Q-network
- The loss function here is mean squared error, equation (3-23). However, the target or actual value here is not known as we are dealing with a reinforcement learning problem. Going back to the Q-value update equation derived from the Bellman equation (3-21), $R_{t+1} + \gamma \max_a Q(S_{t+1}, a)$ represents the target. Since R is the unbiased true reward, the network is going to update its gradient using backpropagation to finally converge.

3.4.5 Target Network

Since the same network is calculating the predicted value and the target value, there could be a lot of divergence between these two. So, instead of using one neural network for learning, two could be used as in the work made in [75, 76].

A separate network to estimate the target could also be used. This target network has the same architecture as the function approximator but with frozen parameters. For every C iterations (a hyperparameter), the parameters from the prediction network are copied to the target network. This leads to more stable training because it keeps the target function fixed (for a while):

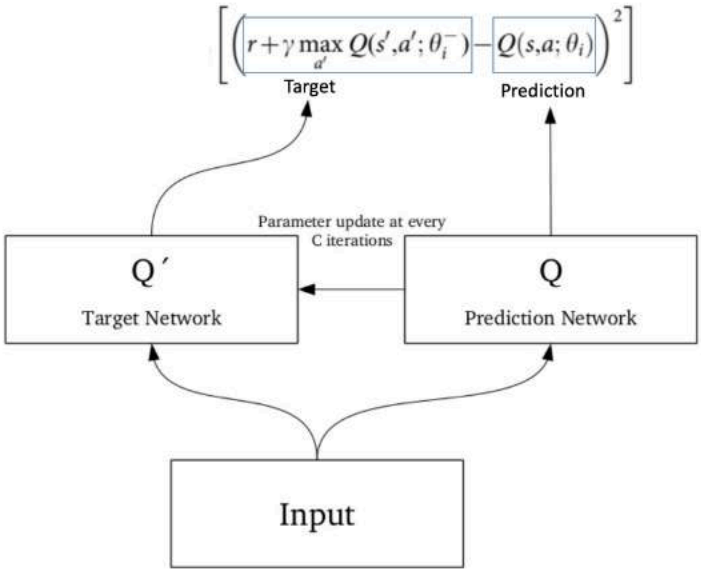


Fig. 3-12 Target network

3.5 Hyperparameters

The three most important hyperparameters for your agent are as follows:

- α: The learning rate
- γ: The discount rate
- ϵ: The exploration rate

3.5.1 Alpha –deterministic versus stochastic environments

The agent's learning rate alpha ranges from zero to one. Setting the learning rate to zero will cause the agent to learn nothing. All of its exploration of its environment and the rewards it receives will not affect its behavior at all, and it will continue to behave completely randomly.

Setting the learning rate to one will cause the agent to learn policies that are fully specific to a deterministic environment. One important distinction to understand is between deterministic and stochastic environments and policies. Briefly, in a deterministic environment, the output is determined by the initial conditions and there is no randomness involved. The same action is always taken from the same state in a deterministic environment.

In a stochastic environment, there is randomness involved and the decisions that are made are given as probability distributions. In other words, a different action is taken from one state to the other.

3.5.2 Gamma –current versus future rewards

The agent's discount rate gamma has a value between zero and one, and its function is to discount future rewards against immediate rewards.

The agent is deciding what action to take based not only on the reward it expects to get for taking that action but on the future rewards it might be able to get from the state it will be in after taking that action.

When a future reward is discounted, it is less valuable than an immediate reward (similar to how we take into account the time value of money when making a loan and treat a dollar received today is more valuable than a dollar received a year from now).

The value of gamma chosen varies according to how highly a future reward is valued:

- If a value of zero is chosen for gamma, the agent will not care about future rewards at all and will only take current rewards into account
- Choosing a value of one for gamma will make the agent consider future rewards as high as current rewards

3.5.3 Epsilon –exploration versus exploitation

The agent's exploration rate epsilon also ranges from zero to one. As the agent explores its environment, it learns that some actions are better to take than others, but what about states and actions that it hasn't seen yet? We don't want it to get stuck on a local maximum, taking the same currently highest-valued actions over and over when there might be better actions it hasn't tried to take yet.

When the epsilon value is set, there will be a probability equal to epsilon that the agent will take a random (exploratory) action, and a probability equal to 1-epsilon that it will take the current highest Q-valued action for its current state. The value that is chosen for epsilon affects the rate at which the Q-table converges and the agent discovers the optimal solution. As the agent gets more and more familiar with its environment, it is expected to start sticking to the high-valued actions it's already discovered and do less exploration of the states it hasn't seen. It is achieved by having epsilon decay over time as the agent learns more about its environment and the Q-table converges on its final optimal values.

There are many different ways to decay epsilon, either by using a constant decay factor or basing the decay factor on some other internal variable. In the experiments, a constant decat factor will be applied to epsilon.

3.6 Reinforcement Learning Process

In a way, Reinforcement Learning is the science of making optimal decisions using experiences. Breaking it down, the process of Reinforcement Learning involves these simple steps:

- Observation of the environment
- Deciding how to act using some strategy
- Acting accordingly
- Receiving a reward or penalty
- Learning from the experiences and refining our strategy
- Iterate until an optimal strategy is found

Different aspects need to be considered here while modeling an RL solution to this problem: rewards, states, and actions.

3.6.1 Rewards

Since the agent (the 6 DOF Robotic arm, the UR10) is reward-motivated and is going to learn how to control the arm by trial experiences in the environment, we need to decide the rewards and/or penalties and their magnitude accordingly. Here a few points to consider:

- The agent should receive a high positive reward for a successful reaching because this behavior is highly desired
- The agent should be penalized if it goes too far from the target
- The agent should get a slight negative reward for not making it to the destination after every time-step. ”Slight” negative because we would prefer our agent to reach late instead of making wrong moves trying to reach to the destination as fast as possible

3.6.2 State Space

In RL, the agent encounters a state and then takes action according to the state it’s in. The State Space is the set of all possible situations the UR10 could inhabit. The state should contain useful information the agent needs to make the right action. The state-space would be composed of the positions of the end effector for all the combinations of the joint’s angles.

3.6.3 Action Space

The action space is the set of all the actions that the agent can take in a given state. It’s all the possible joints angles that the UR10 can make.

3.7 Integration of RL for path planning

3.7.1 RL-related algorithms

Q learning directly uses maximum estimated action value $\max_{Q} Q(S_{t+1}, A_{t+1})$ at time step t+1 to update its action value. It also pays attention to the maximum estimated action value of the next step and selects optimal actions eventually. The same algorithm as presented below has been used for different studies for motion planning [77-79].

Algorithm 1 Q-Learning algorithm

```
Initialize α, ϵ, γ;
Initialize Q(s, a) arbitrarily;
while s not terminal do
  for each episode do
    Initialize s;
    for each step of episode do
      if ϵ < Random Uniform (0,1) then
        Choose random a
      else
        Max a in the Q-Table for s
      end if
      Take action a, observe r, s';
      Q(s, a)←Q(s, a) + [r + max'_a Q(s', a')−Q(s, a)];
      s←s';
    end for
    Update α, ϵ, γ;
  end for
end while
```

DQN became a research focus when it was invented by Google DeepMind [80, 81]. It is a combination of Q-leaning and neural networks. DQN uses NN to approximate Q-values by its weight θ. The principle of the DQN algorithms is exposed in the following algorithm.

Algorithm 2 Deep-Q-Learning algorithm

```
Initialize action-value Network with random weights;
Initialize α, ϵ, γ;
while s not terminal do
  for each episode do
    Initialize sequence s_1 = {x_1} and preprocessed sequenced φ_1 = φ(s_1);
    for each step of episode do
      if ϵ < Random Uniform (0,1) then
        Choose random a
      else
        Predict max_a Q*(φ(s_t, a; θ) with the Network
      end if
      Take action a, observe r, s';
      Q(s,a)←Q(s,a) + [r + max'_a Q(s',a') − Q(s,a)];
      s←s';
    end for
    Update α, ϵ, γ;
    Save θ;
  end for
end while
```

3.7.2 RL applied to the study

With RL, motion planning is realized by attaching destination and safe paths with big reward (numerical value), while obstacles are attached with penalties (negative reward). An optimal path is found according to total rewards from the initial place to the destination. The UR is first discovering the environment. To have this discovering behavior, the hyperparameters are set to the lowest for epsilon (exploration) and gamma (current reward). That way the robot is mapping the workspace and start to ”learn” how its joints work.

The robot is guided only with the rewards which increase for a closer position to the target. The hyperparameters are updated at each episode. The epsilon and gamma increase which induces more specific tasks. After a few episodes, the exploration is no more required and the goal reward is more considered than the current rewards. To maximize the reward, updated with the Bellman equation and so the hyperparameters, the robot has to find an optimal path. The UR is then finding a sequence of joints to get to the goal and have the final reward. The learning rate, alpha, is set to 0.5 to have a good balance between stochastic and deterministic environments.

Noise in DQN leads to bias and false selection of next action a' follows, therefore leading to over-estimation of next action value $Q(s', a', \theta')$. To reduce the over-estimation caused by noise, the DQN algorithm uses a double network such as the target network explained previously. The DQN algorithm used for the study is a combination of two different techniques: Experience Replay to avoid overfitting and a target network.

3.8 Summary

In this chapter, the kinematic of the robotic has been seen. This part is necessary for the URDF to simulate the robot arm. The inverse and forward kinematic has also been explained for the UR10. Then the RL theory was drawn up with mathematical tools such as MDP and the Bellman equation. Finally, Q-Learning and DQL which are the two Policies used in this study were explained. The concept of Neural Networks was also detailed.

4 Simulation and experiment

This chapter will present the process that leads to the final simulation and the experiments. The software to use for the simulation has to carefully choose, it has to be convenient to use and powerful enough to obtain results for the experiments. In this chapter, we will compare two policies the Q-Learning which the most basic policy in RL, and the DQL which uses a Neural Network instead of a Q-table. The results produced by the simulation of these two policies will be confronted to find the optimal one for path planning.

4.1 Choosing the environment

4.1.1 ROS and SmartGrasping Sandbox

Robot Operating System (ROS) is an open-source robotics middleware suite. Although ROS is not an operating system but a collection of software frameworks for robot software development, it provides services designed for a heterogeneous computer cluster such as hardware abstraction, low-level device control, implementation of commonly-used functionality, message-passing between processes, and package management. Running sets of ROS-based processes are represented in a graph architecture where processing takes place in nodes that may receive, post, and multiplex sensor data, control, state, planning, actuator, and other messages.

ROS works with $topics$, $nodes$, and $publisher$ to communicate with the sensors and robots. That's naturally that this study first attempted the use of this common robotic tool. A first simulation was made using in part the work made by [82]. This simulation aimed to catch a ball using the shadow hand developed by the shadow robot company. In Figure 4-1 the simulation is presented, on the left side, the gazebo simulator allows to have a graphical return of the actions.

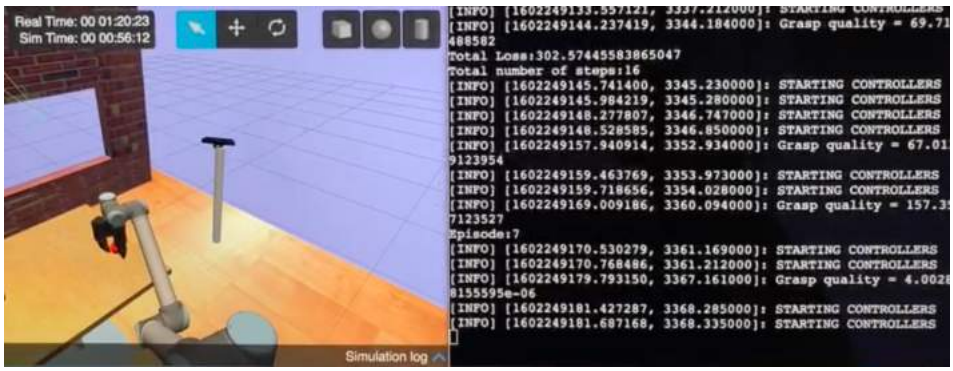


Fig. 4-1 Shadow hand with ROS + Gazebo

The first issue that this configuration leads to is the lack of flexibility. There is no possibility to speed up the training by disconnecting the Gazebo simulation, the code needs the right version of Gazebo, ROS, Python, and Linux without what it can't be exploited by someone else. Then the environment was designed in 2018 so without the newest techniques of reinforcement learning. Finally, ROS is very powerful and convenient but sometimes too complex for simple tasks.

4.1.2 Robo-gym

In the second place, the study turned to use Robo-gym which was developed by [73]. The team developed Open Ai Gym environments for the use of Mir100 and the UR10. The environments are developed in a $docker$ container. The communication between the robot and the environment is made through $ServiceManager$. This technique is supposed to be more convenient to use but the project is still not fully stable and many errors occur during the training. The connection through $ServiceManager$ was very slow and couldn't be properly used.

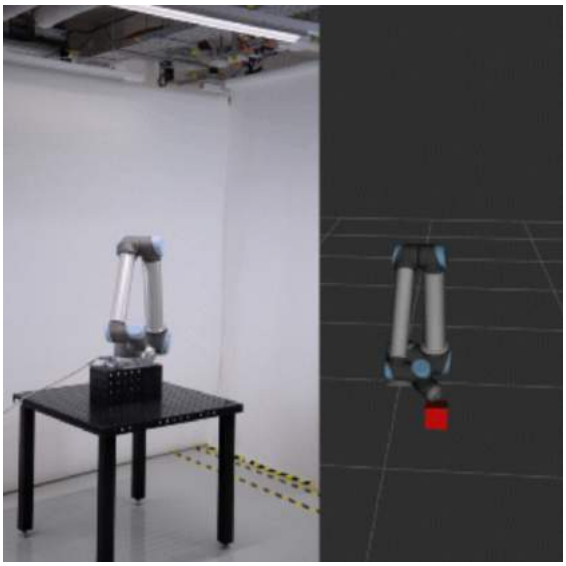


Fig. 4-2 "EndEffectorPositioningUR10Sim-v0" in Reality and in Gazebo

4.1.3 Pybullet

After multiple trials and research on different simulators (Gazebo, Mujoco, Unity...) the choice has been done to use Pybullet. All the programs, the Open Ai Gym environments, the agent training programs, the Unify Robot Description File (URDF) were made from scratch using Pybullet for the simulation and the connections to the robot. That way all the simulations could be fully handled and modify if needed.

The main advantage of Pybullet is a DIRECT mode i.e. without Graphical Unit Interface (GUI) which allows to speed up the training. The interface is rather easy to manipulate and offers many interesting functions to control the robot. Pybullet is also open source so there is no license to pay for its use (compare to Mujoco) and is updated with a community. Pybullet is widely for robotic and RL [83-85]. Finally, thank its rather simple interface the training was faster than with Gazebo for example which need time in every reset to set the ”world”.

The graphical interface is presented on the Figure 4-3.

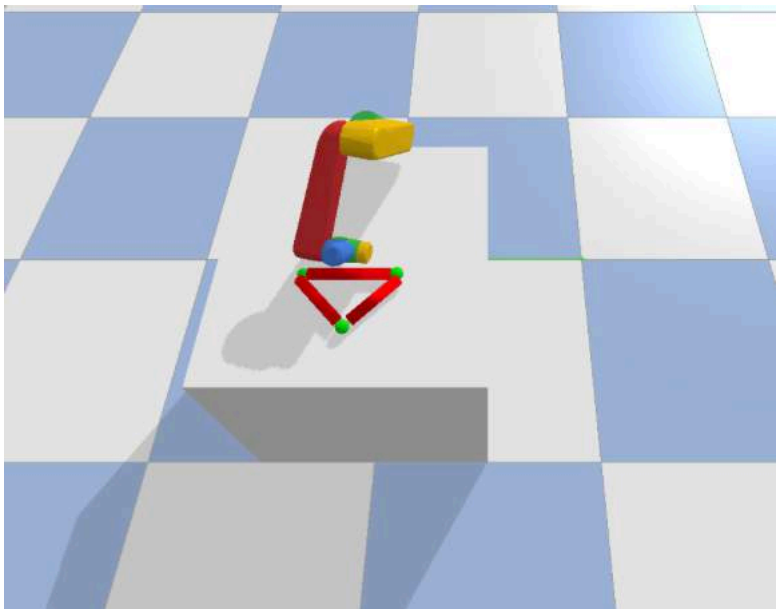


Fig. 4-3 Pybullet GUI

4.1.4 Comparison of the simulation environment

The 3 environments presented are compared in the following tabular.

Tab. 4-1 Comparison of simulations tools

Techniques	**GUI**	**Open Sourced**	**Possibility to speed training**	**General handling**
Smart Grasping Sandbox	Gazebo	Yes	No	Complicated to get started
RoboGym	Service Manager	Yes	No	Difficult communication with the robot
Pybullet	Yes	Yes	With DIRECT mode	Easy to get started

4.2 Simulation

The building of the simulation is made in 3 different parts that communicate together.

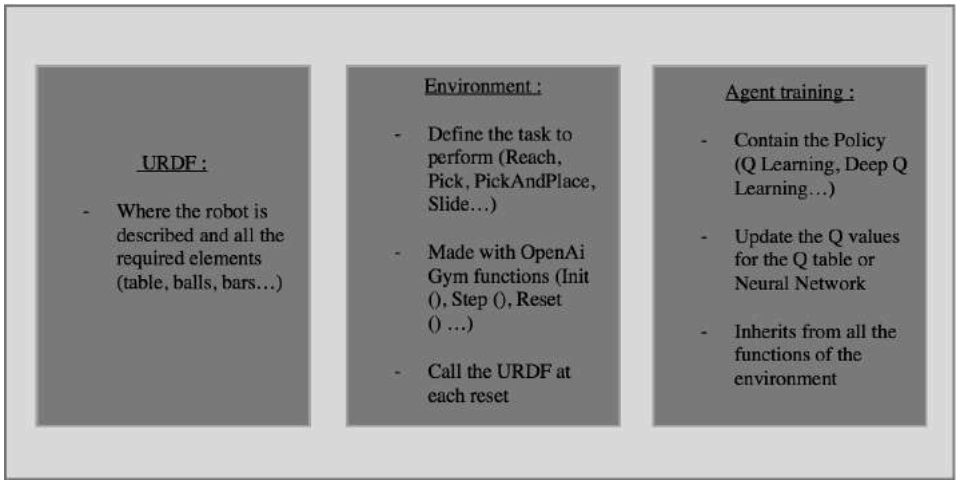


Fig. 4-4 Three different parts of the simulation

4.2.1 State and Action Space

To speed up the training a discrete State Space has been set up. Only the interesting joint angles were selected for the study and experiments. In their work, [70] used the same way to proceed by constructing a reasonable environment and state space. A similar technique was used by [69] where approximate regions instead of accurate measurements are used to define new state space and joint actions.

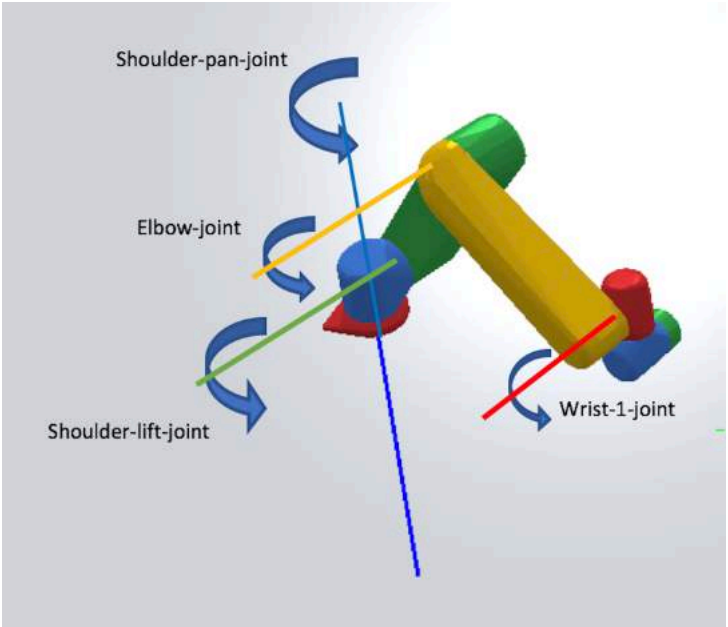


Fig. 4-5 Presentation of the simulated UR

The possible angles for the 4 joints are as followed:

- Shoulder-pan-joint = [1.8 1.56 1.32 1.08 0.84 0.6 0.36 0.12 -0.12 -0.36 -0.6 -0.84 -1.08 -1.32 -1.56 -1.8]
- Shoulder-lift-joint = [-0.3 -0.4 -0.5 -0.6 -0.7 -0.8 -0.9 -1.1 -1.2 -1.3]
- Elbow-joint = [0.8 0.935 1.07 1.205 1.34 1.475 1.61 1.745 1.88 2.015 2.15]
- Wrist-1-joint = [-0.5 -1.0 -1.5 -2.0]

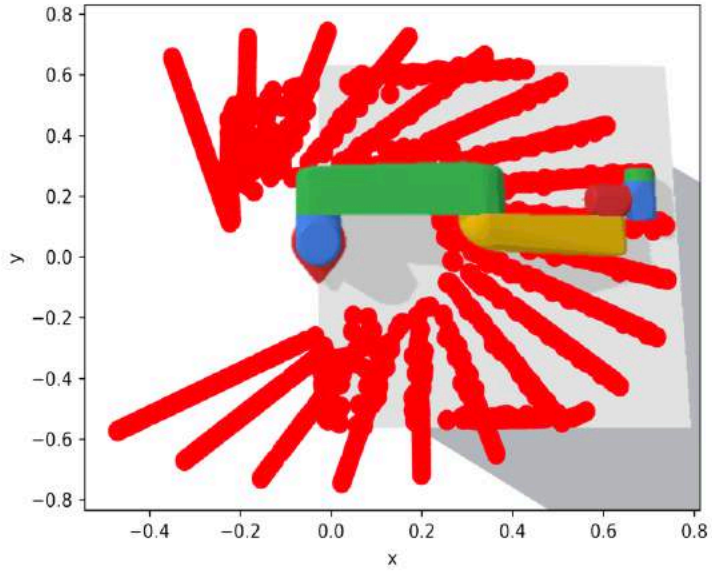


Fig. 4-6 Workspace map of the UR

These joints were chosen so that the robot could reach almost every part of the table in front of it as it is presented in Figure 4-6. As long as the target is $20mm$ away from the end effector it can catch it with the magnetized end-effector. The collision of the table is directly managed in the URDF. With all these possible angles the actions of the UR10 can be considered in our study as continuous. The state space is then 16 x 10 x 11 x 4 = 7040 states. The action space is 16 + 10 + 11 + 4 = 41 actions.

4.2.2 Environment

As previously said the environment is where the objects interact to complete a task. Usually, the environment takes the name of the task it is aimed for. For this part, Open Ai Gym was used. Gym is a toolkit for developing and comparing reinforcement learning algorithms. This

means that it's the same functions that are developed for every environment [86]. The main functions are:

- Init(): It's the first function called that starts the environment.
- Step(): This function is called every time the robot makes an action. In our case Step() choose a random joint to move in a random position for the training.
- Reset(): It simply reset the environment every time an epoch is finished.
- Observation(): It gives the state in which the robot is. It returns a 4 vector with the angle of the joints.

1) Task

The final environment that leads to the assembly is the $pick\ and\ place$ environment. It is a composition of two $reach$ environments. That's why for the comparison of the policies, only the $reach$ environment will be considered.

2) Reward

Different rewards are given, all of them are based on the distance between the target and the end-effector. A simple function for the robot is (4-1) which is updated at each step. A reward of 10 is given when the object to grasp is reached and then a reward of 20 when the assembly point is reached. Penalties are given if the arm is going too far.

$$R(s,a) = \begin{cases} 10 & \text{if } distance < 0.8m \\ 20 & \text{if } distance < 0.8m \text{ for final destination with the object} \\ -1 & \text{if } distance > 1m \\ \frac{1}{distance} & \text{else} \end{cases} \tag{4-1}$$

4.3 Q-Learning

Once the environment is defined, the policy is then applied to it. That's where the robot is learning.

4.3.1 Q table

In Q Learning a Q table is used as the memory. The table is composed of the actions and states and is updated with the Bellman equation at every step. The table is initialized with zeros.

Tab. 4-2 Q table initialized with zeros

Q Table		Actions											
		Shoulder_pan_joint			Shoulder_lift_joint			Elbow_joint			Wrist_1_joint		
		1.8	...	-1.8	-0.3	...	-1.3	0.8	...	2.15	-0.5	...	-2.0
State	1	0	...	0	0	...	0	0	...	0	0	...	0
	2	0	...	0	0	...	0	0	...	0	0	...	0
	3	0	...	0	0	...	0	0	...	0	0	...	0
	...	...	...	...	...	...	...	...	...	...	...	...	...
	7039	0	...	0	0	...	0	0	...	0	0	...	0
	7040	0	...	0	0	...	0	0	...	0	0	...	0

After multiple episodes of training the Q table is filled with coefficient through the Bellman equation.

Tab. 4-3 Q table after training

Q Table		Actions											
		Shoulder_pan_joint			Shoulder_lift_joint			Elbow_joint			Wrist_1_joint		
		1.8	...	-1.8	-0.3	...	-1.3	0.8	...	2.15	-0.5	...	-2.0
State	1	1.43	...	2.79	0	...	1.31	3.78	...	3.95	4.47	...	1.46
	2	2.67	...	3.23	0	...	1.78	4.29	...	1.43	2.78	...	2.53
	3	1.58	...	2.12	0	...	1.98	1.23	...	1.76	3.57	...	2.34
	...	...	...	...	...	...	...	...	...	...	...	...	...
	7039	3.89	...	1.45	0	...	2.34	3.67	...	3.87	1.32	...	2.97
	7040	1.34	...	0.56	0	...	2.87	1.83	...	2.63	1.89	...	3.65

When the robot exploits the Q table it looks at the line for a certain state. For example in the state 3, Shoulder-pan-joint = 1.32, Shoulder-lift-joint = -0.3, Elbow-joint = 0.8 and Wrist-1-joint = -0.5 the optimal action according to the Q table is to move the Wrist-1-Joint to -2.0.

Tab. 4-4 Exploitation of the Q table

Q Table		Actions											
		Shoulder_pan_joint			Shoulder_lift_joint			Elbow_joint			Wrist_1_joint		
		1.8	...	-1.8	-0.3	...	-1.3	0.8	...	2.15	-0.5	...	-2.0
State	1	1.43	...	2.79	0	...	1.31	3.78	...	3.95	4.47	...	1.46
	2	2.67	...	3.23	0	...	1.78	4.29	...	1.43	2.78	...	2.53
	3	1.58	...	2.12	0	...	1.98	1.23	...	1.76	2.34	...	3.57
	...	...	...	...	...	...	...	...	...	...	...	...	...
	7039	3.89	...	1.45	0	...	2.34	3.67	...	3.87	1.32	...	2.97
	7040	1.34	...	0.56	0	...	2.87	1.83	...	2.63	1.89	...	3.65

4.3.2 Training the agent

First, we'll initialize the Q-table to a matrix of zeros. Then the training algorithm is created, it will update this Q-table as the agent explores the environment over one thousand episodes.

In the first part of **while not done**, it is decided whether to pick a random action or to exploit the already computed Q-values. This is done simply by using the epsilon value and comparing it to the **random.uniform(0, 1)** function, which returns an arbitrary number between 0 and 1.

The chosen action is executed in the environment to obtain the **next_state** and the **reward** from performing the action. After that, the maximum Q-value is calculated for the actions corresponding to the **next_state**, and with that, the Q-value can easily be updated to the **new_q_value**. It is the same principle as the Algorithm 1 in Section 3.7.

One epoch is completed once the end-effector reaches the target with a minimum distance of $20mm$. This distance is a reasonable distance from magnetization. It has been tested up to $50mm$ with the neodymium magnets presented in Chapter 2 Figure 2-15. The hyperparameters are updated at each episode, especially ϵ which is decreasing during the learning.

4.3.3 Evaluating the agent

After one thousand episodes the end effector can reach the target with around 25 actions (Epochs). The training took around 2 hours to be completed. 1000 Episodes is the number of episodes required to make the robot learn.

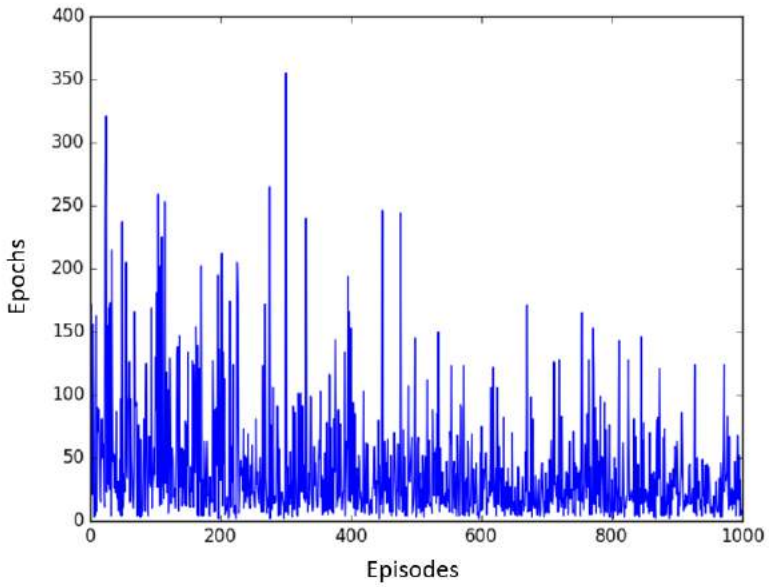
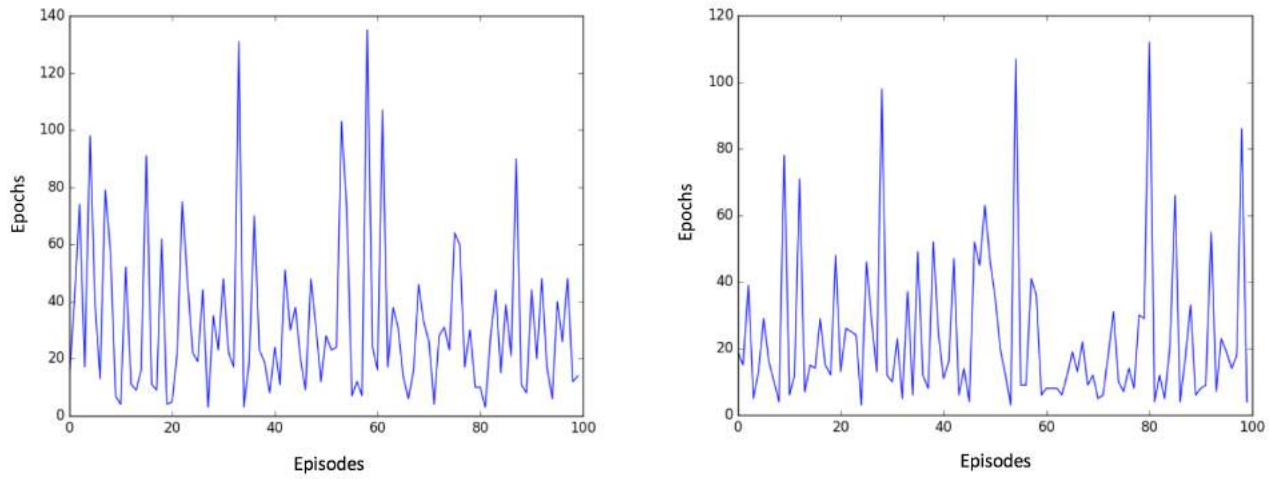


Fig. 4-7 Training after 1000 Episodes

In a second algorithm, the efficiency of this training was tested for 100 Episodes during which the robot was only exploiting the Q Table. It has been compared to an untrained robot.



(a) Untrained robot (b) Trained robot

Fig. 4-8 Comparison after training

One step represents a modification of the angle of the robot. This is what gives us feedback on how good the robot is to find the optimal path. For the untrained robot 32.34 steps were required to reach the target against 22.52 steps for the trained robot. So after training the robot learned to reduce the number of steps on average from 10 steps. This is not significant. This could be explained by the fact that the Q-Learning is not adapted to continuous work such as this reach task.

4.4 Deep Q Learning

In Deep Q Learning, a Neural Network is used to approximate the Q-value function using $Keras$ [87] which is a library of $TensorFlow$ [88]. Tensorflow is a widely used tool for Machine Learning.

4.4.1 Define Network

Neural networks are defined in $Keras$ as a sequence of layers. The container for these layers is the Sequential class. The first step is to create an instance of the Sequential class. Then the layers are created and added in the order that they should be connected. The first layer in the network must define the number of inputs to expect. The way that this is specified can differ depending on the network type, but for a Multilayer Perception model, this is specified by the input_dim attribute. In the case of the study the first layer is of the size of the $state\_size$ which

is the number of possible moves for the robot, so 4.

A $Sequential$ model could be seen as a pipeline with the raw data fed in at the bottom and predictions that come out at the top. This is a helpful conception in $Keras$ as concerns that were traditionally associated with a layer can also be split out and added as separate layers, clearly showing their role in the transform of data from input to prediction. For example, activation functions that transform a summed signal from each neuron in a layer can be extracted and added to the Sequential as a layer-like object called $Activation$.

The choice of activation function is most important for the output layer as it will define the format that predictions will take. An activation function is a node, added the output of a layer or between two layers. Literature also calls it neuron or unit. It allows the output of the neural network to have resulting values between 0 to 1 or -1 to 1. Some of them are more popular such as logistic $sigmoid$, $tanh$, and $ReLU$. Activation function takes though a number to perform a mathematical operation associated with.

(1) Logistic Sigmoid or Sigmoid uses a real value x input to arrange it into the range [0,1],

$$\sigma(x) = \frac{1}{1 + \exp -x} \tag{4-2}$$

(2) Hyperbolic Tangent or tanh uses a real value x input to arrange it into the range [-1,1],

$$tanh(x) = 2\sigma(2x) - 1 \tag{4-3}$$

(3) Rectified Linear Unit or ReLU uses a real value input and threshold negative values at zero,

$$f(x) = max(0, x) \tag{4-4}$$

Following graphs 4-9 show each activation function,

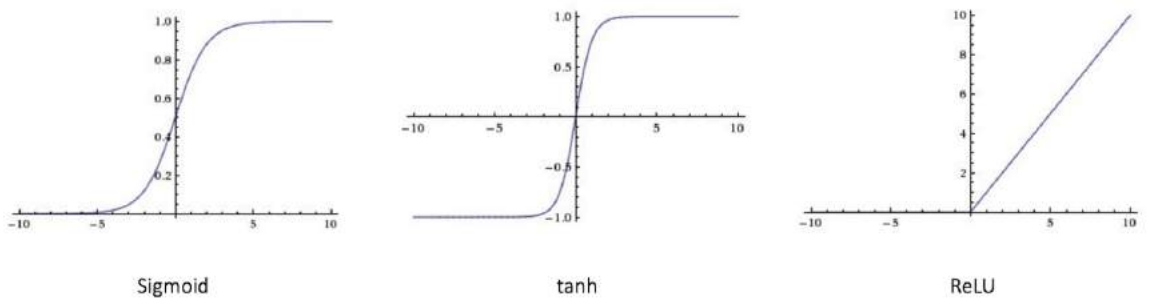


Fig. 4-9 Three different activation functions

Thus, for linear regression we use the sum of square error; for binary classification, we use logistic sigmoid and cross-entropy; for multiclass classification, we use soft-max and cross-entropy; and for NN ReLU is traditionally more used.

For the study, the regression activation has been chosen because the number of neurons matches the number of outputs which is the number of actions (41). So that each action weights each input state.

4.4.2 Compile Network

Once we have defined the network with $Sequential$ and choose the activation function ($ReLu$) we head up to compilation which is an efficiency step. It transforms the simple sequence of layers that we defined into a highly efficient series of matrix transforms in a format intended to be executed on the Graphical Process Unit (GPU).

The compilation is always required after defining a model. This includes both before training it using an optimization scheme as well as loading a set of pre-trained weights from a save file. The reason is that the compilation step prepares an efficient representation of the network that is also required to make predictions on the hardware.

Compilation requires several parameters to be specified, specifically tailored to training the network. Specifically, the optimization algorithm to use to train the network and the loss function used to evaluate the network is minimized by the optimization algorithm.

The type of predictive modeling problem imposes constraints on the type of loss function that can be used. For example, below are some standard loss functions for different predictive model types:

- Regression: Mean Squared Error or ”mse”.
- Binary Classification (2 class): Logarithmic Loss, also called cross entropy or ”binary_crossentropy”.
- Multiclass Classification (>2 class): Multiclass Logarithmic Loss or ”categorical_crossentropy”.

The most commonly used predictive model is the mean square error which is the one also used for our Deep-Q-Network.

4.4.3 Fit Network

Once the network is compiled, it can be fit, which means adapting the weights on a training dataset. Fitting the network requires the training data to be specified, both a matrix of input

patterns X and an array of matching output patterns y. So the data were reshaped to get the input in form of a matrix $(state\_size, 1)$,

$$S_t = \begin{bmatrix} s_1 \\ s_2 \\ s_3 \\ \vdots \\ s_k \end{bmatrix} \tag{4-5}$$

and the output $(action\_size, 1)$,

$$Q(s_t) = \begin{bmatrix} Q(s_t, a_1) \\ Q(s_t, a_2) \\ Q(s_t, a_3) \\ \vdots \\ Q(s_t, a_n) \end{bmatrix} \tag{4-6}$$

We denote $Q(s_t)$ a vector of all action-values in the state s_t, and use $Q(s_t, a_t)$ to specify the Q-value of taking at in s_t. The action value iteration is realized by updating the neural network by the means of its weights.

The network is trained using the $backpropagation$ algorithm and optimized according to the optimization algorithm and loss function specified when compiling the model. The optimizer choose is Adam. Adam optimization is a stochastic gradient descent method that is based on adaptive estimation of first-order and second-order moments. As for the loss function the mean square error was used. The backpropagation algorithm requires that the network be trained for a specified number of epochs or exposures to the training dataset.

Each epoch can be partitioned into groups of input-output pattern pairs called batches. This defines the number of patterns that the network is exposed to before the weights are updated within an epoch. It is also an efficiency optimization, ensuring that not too many input patterns are loaded into memory at a time.

4.4.4 Evaluate Network

The network can be evaluated on the training data, but this will not provide a useful indication of the performance of the network as a predictive model, as it has seen all of this data before. The performance of the network can be evaluated on a separate datasheet, unseen during testing. This will provide an estimate of the performance of the network at making predictions for unseen data in the future.

The model evaluates the loss across all of the test patterns, as well as any other metrics specified when the model was compiled, like accuracy. A list of evaluation metrics is returned. The evaluation of the network is detailed later in the section ”evaluating the agent”.

4.4.5 Overfitting

Overfitting is ”the production of an analysis that corresponds too closely or exactly to a particular set of data, and may therefore fail to fit additional data or predict future observations reliably”. An overfitted model is a statistical model that contains more parameters than can be justified by the data.

It is a model that has learned too much how to do a specific task. The robot won't be able to learn from new inputs. Graphically it is observed when the accuracy is getting a fluctuation in its variations compared to the training.

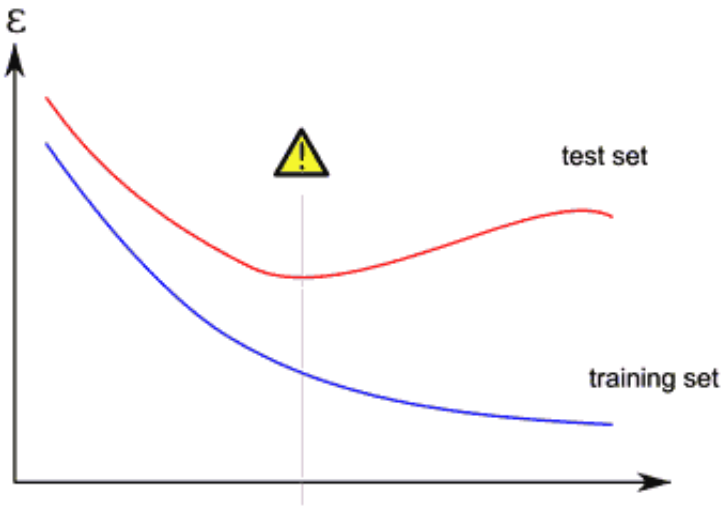


Fig. 4-10 Overfitting

4.4.6 Make Predictions

Finally, once we are satisfied with the performance of our fit model, we can use it to make predictions on new data. This is made by calling the $predict()$ function on the model with an array of new input patterns. The predictions will be returned in the format provided by the output layer of the network. In the case of a regression problem, these predictions will be in the format of the problem directly, provided by a linear activation function.

4.4.7 Experience Replay

The DQN is easily overfitted over current episodes [89, 90]. Once DQN is overfitted, it's hard to produce various experiences. To solve this problem, Experience Replay stores expe-

riences including state transitions, rewards, and actions, which are necessary data to perform Q learning, and makes mini-batches to update neural networks. This technique expects the following merits.

- reduces the correlation between experiences in updating DQN
- increases learning speed with mini-batches
- reuses past transitions to avoid catastrophic forgetting

Algorithm 3 Experience Replay with target network

```
Initialize the Batch with random selection in the experiences;
for a, r, s, s' in the Batch do
    if done then
        Target = reward
    else
        Target ← r + γ × (max_a Q*(φ(s_{t+1}, a; θ))
    end if
    Target_f = predict(state);
    Target_f[0][action] = target;
    Evaluate the Network;
end for
```

Experience replay is a rather new technique that provides very good results combined with a target network has it can be seen in the following sections where the agent is evaluated. Overfitting has been avoided through this technique.

4.4.8 Training the agent

The process to train the agent is almost the same as for the Q Learning the first actions taken are random and the rewards are given according to the distance to the target. Now the agent has a memory in which it can store the $state, action, reward, nextstate$, and $done$. And after a while, the actions taken are predicted with the Neural Network. In this program, hyperparameter are updated at each episode.

A mini-batch is created, the size of this batch is set to 256, it has been determined through experiences that it's the best size to obtain good results and minimize the time of learning. The experience replay is working on that mini-batch so if it's too small it won't have enough data to give results but if it's too large the calculations take too long.

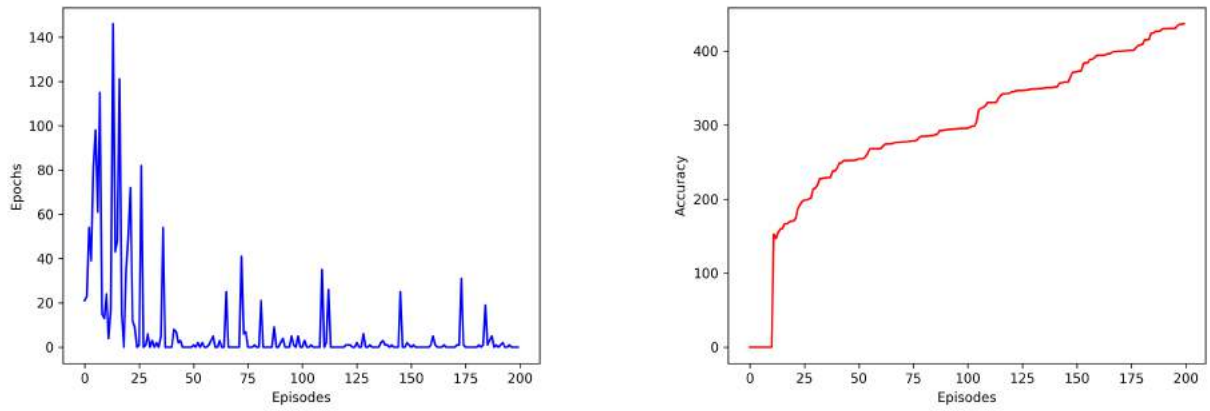
Also after experiences, it has been understood that the training of the agent had to be done on at least 8 times the mini-batch size so that there are enough random values of the batch for the replay. The weights are saved in a $model.h5$ file so that they could be reused later.

Algorithm 4 Deep Q Learning with Experience Replay

Initialize the Network with $state\_size$ and $action\_size$;
Initialize the Memory D;
Initialize α, ϵ, γ;
for each episode **do**
  Initialize the environment;
  Initialize sequence $s_1 = \{x_1\}$ and preprocessed sequenced $\phi_1 = \phi(s_1)$;
  Reshape the states;
  for each epochs **do**
    while not done **do**
      if Ramdom value in $(0, 1)$ < epsilon **then**
        Choose Random a
      else
        Predict $\max_a Q^*(\phi(s_t, a; \theta)$ with the Network
      end if
      Take action a;
    end while
    if Done **then**
      Reward = Reward
    else
      Reward = -1
    end if
    Store a, r, s, s', done in D;
    $s \leftarrow s'$
    if D > 8 $\times$ Batch_size **then**
      Experience Replay;
    end if
    Update α, ϵ, γ;
    Save θ;
  end for
end for

4.4.9 Evaluating the agent

For DQL another parameter is to take into account compared to Q-Learning. The network accuracy needs to be followed to avoid overfitting.



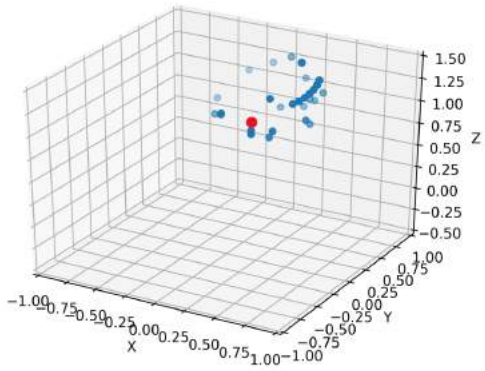
(a) Learning evolution

(b) Network evolution

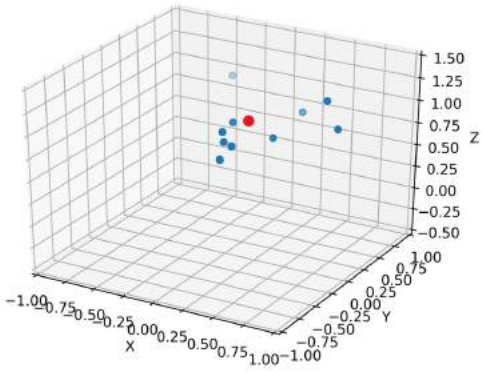
Fig. 4-11 Evaluation of the agent for $reach$

On graph (a) we can see the number of steps decreasing Episodes after Episodes. The training could have been stopped at 100 Episodes but the last 100 Episodes allow to obtain a more stable model. Graph (b) ensures to not have overfitting. The curve is starting after around 15 Episodes because before that the Experience Replay wasn't involved. So there is only an increasing accuracy, non-fluctuation so no overfitting. The training duration was about 30 minutes for 200 Episodes.

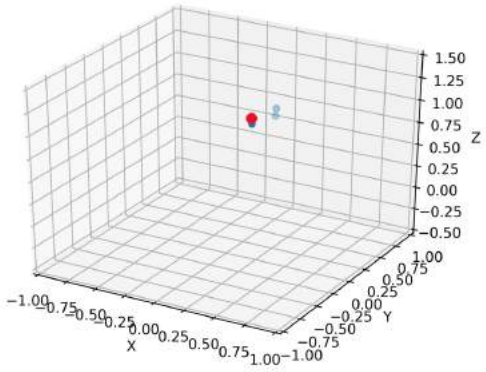
The distribution of the end effector is shown in the following figures. We can see the evolution of the position of the robot to reach the target in red. In Episode 1 the robot is exploring the environment and then it starts to focus on where the reward is the best for him to finally find the optimal path to the target at Episode 200.



(a) Tip pose episode=0



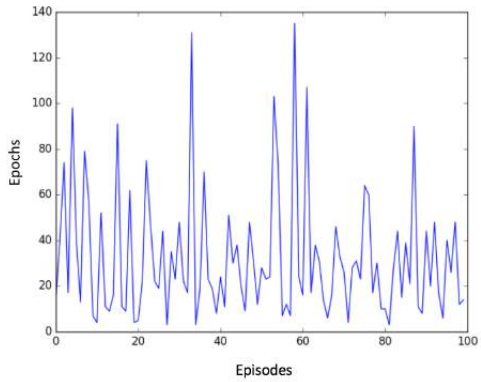
(b) Tip pose episode=100



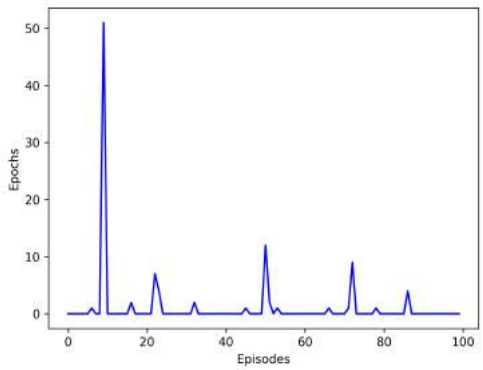
(c) Tip pose episode=200

Fig. 4-12　Tip pose

Let’s now compare to an untrained robot during 100 Episodes same as it has been made for the Q-Learning to observe the efficiency of the training.



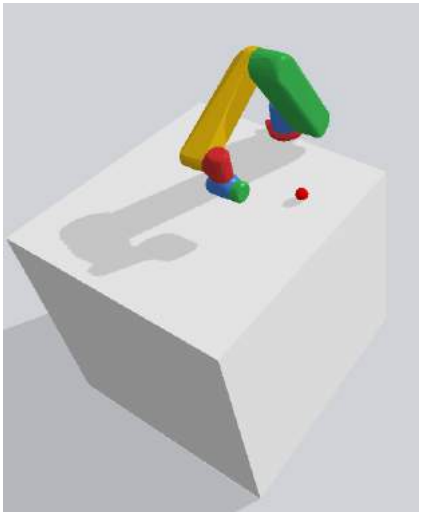
(a) Untrained robot



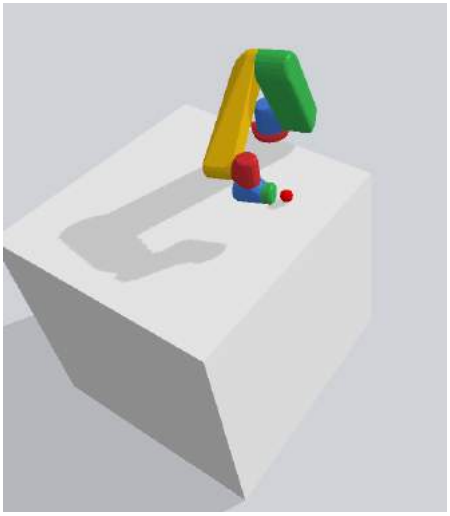
(b) Trained robot

Fig. 4-13　Comparison after training

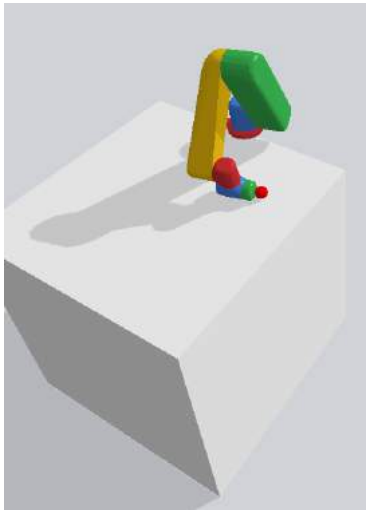
If we remove the pic at around 50 steps, the average number of steps is around 3 steps. Compare to more than 30 Steps for an untrained robot the conclusion is that the robot is very efficient after training. 3 steps it is the minimum expected for the robot to reach the target for its starting position to the target. This optimal path can be verified with the GPU. As presented in the following figures, the robot needs 3 steps that represent the optimal path.



(a) Starting position



(b) Step 1, Shoulder-pan-joint to -1.08



(c) Target Reached, Elbow-joint to 1.07

Fig. 4-14 Steps to reach the target

4.5 Comparison of the Policies

Q Learning was the first policy treated because it's the easiest to manipulate. It gives the first view of Reinforcement Learning. The results provided by this technique stay rather low in terms of efficiency with only a difference of 10 steps compare to the untrained robot. But it has been interesting to realize through the experience that it is not adapted to the problem.

The second technique which is Deep Q Learning has reused the same concept but added to that Neural Network. The experiences showed that it was more adapted to the robotic task. The number of steps was reduced to the minimum (3 steps) after training. The robot found the optimal path to reach the target. This Policy will be used in the next section to realize the final assembly.

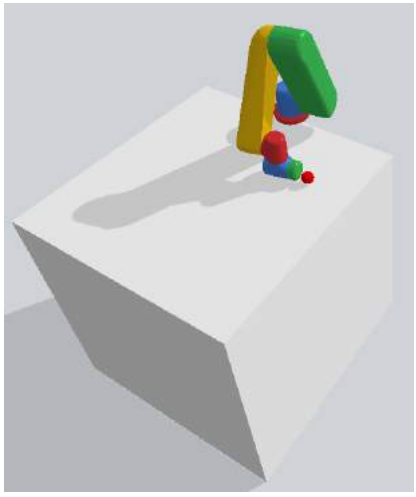
Tab. 4-5 Comparison of the Policies

	Q Learning	**Deep Q Learning**
Number of episodes for results	1000	100
Training time	2 hours	30 minutes
Steps to reach the target	22	3
Adapted to the problem	No	Yes

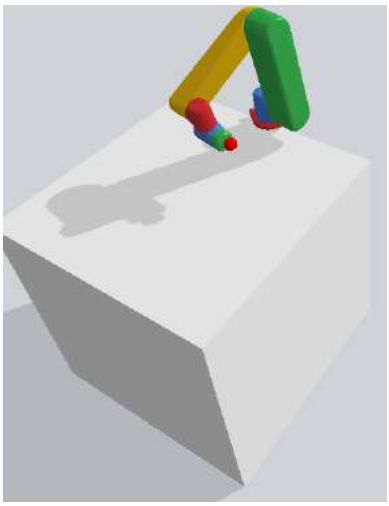
4.6 Pick and Place

4.6.1 Environment

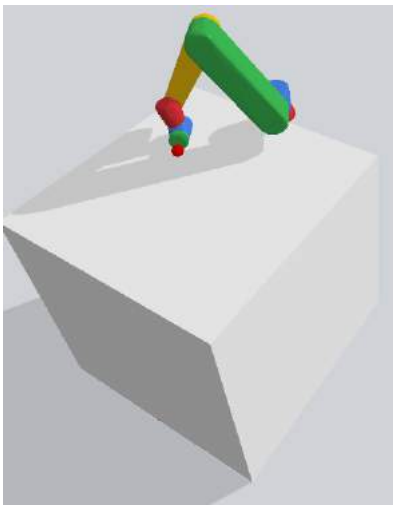
The environment describes here is the $pick\ and\ place$ which is necessary for the assembly. As seen in the last section the DQL is more suitable for this task. So the robot is trained to first reach an assembly element (a red ball) and then place it in a desirable location. Again a minimum distance is set to obtain the magnetization between the end effector and the assembly element. After the first element is "fixed" to the tip, it will be unmagnetized once it has reached the final location for the assembly. Then the UR will pick another element and so on to get the desired assembly.



(a) Reaching the element and grasping with magnetization



(b) Moving the element to the final position

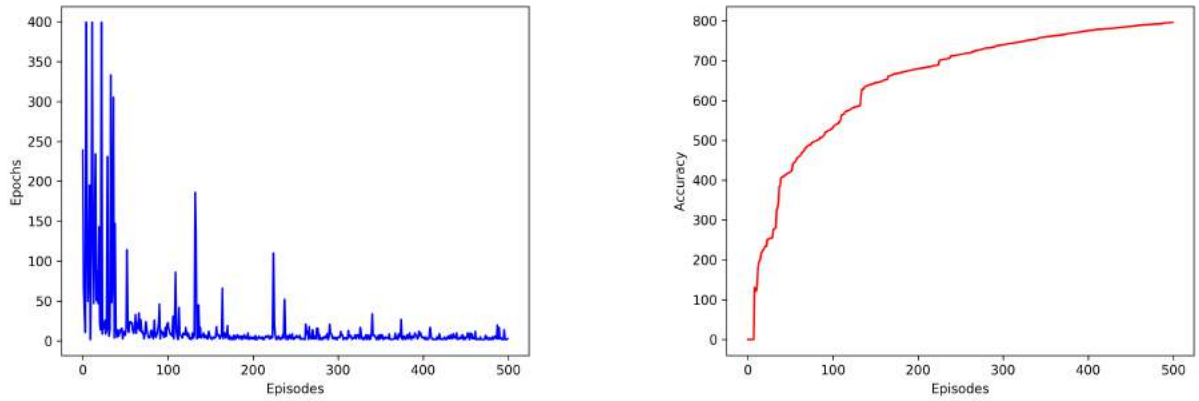


(c) Placing the element to the final postion

Fig. 4-15 Steps to pick and place

4.6.2 Training the agent and Evaluating the Network

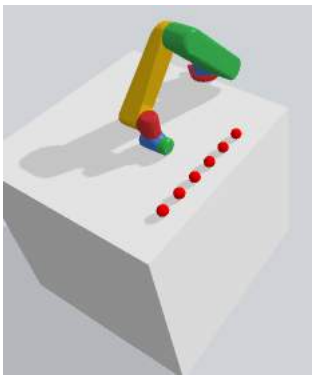
The training took about 5 hours and it required around 500 steps. The accuracy doesn't present any fluctuation so there is no overfitting. However, we can see that the accuracy is almost stabilizing at around 200 Episodes. The training could have been stopped at 200 Episodes but again for a more stable network, the training was continued until 500 Episodes. The minimum number of steps has been also meeting with an average of 5 steps to pick and place the object. The accuracy of the position is still set at 0.8m so the robot can place the ball with an accuracy of 0.8m. A more accurate model with more positions and longer training could provide a better positioning but the material used for the study is not able to increase the accuracy. The behavior of the robot already meets the requirements for this task by finding the optimal path.



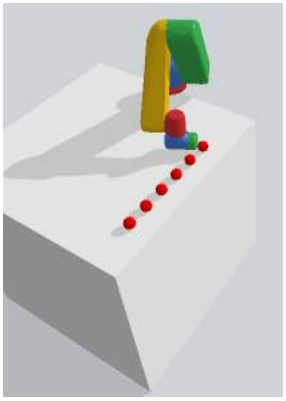
(a) Learning evolution (b) Network evolution

Fig. 4-16 Evaluation of the agent for *pick and place*

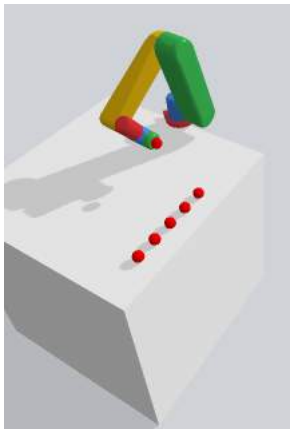
A training oriented to replicate the form of hexagon has also been conducted. The magnetic balls and pick and place one by one to get a hexagon. The bars couldn't be added because the orientation has to be taken into account and the material used for the study is not powerful enough for such a task.



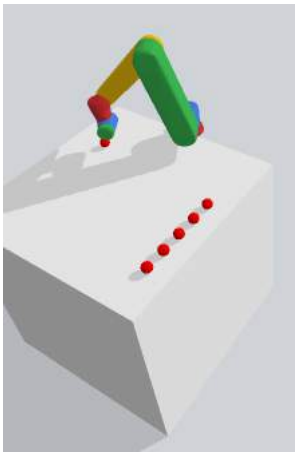
(a) Start



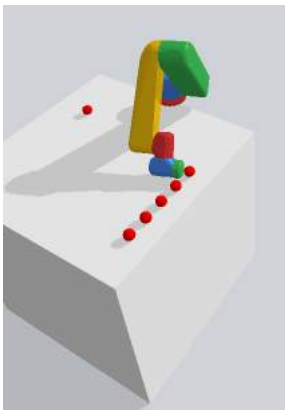
(b) Reach



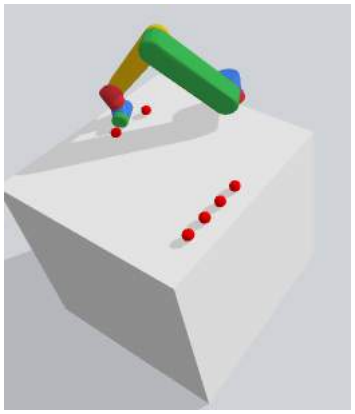
(c) Move



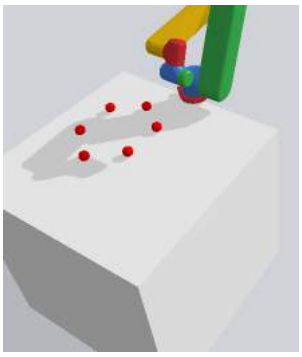
(d) Place



(e) Reach



(f) Place



(g) Final Assembly

Fig. 4-17 Assembly of hexagons with the magnetic balls

4.7 Summary

First, the software Pybullet has been chosen among other environments for its convenience for the experiments. Then the process of creating a RL algorithm was detailed. In third place, the Q-Learning policy was used for experiments. The results obtained with this policy for reaching a target were not conclusive. The number of steps is the number of modifications of the joint angle. This is what gave the efficiency of the robot. It could only reduce the number of steps by 10 compared to an untrained robot. Then the DQL showed better results. With this Policy, the robot could find the optimal path to the target with the minimum number of 3 steps. Finally, that technique was kept to make the $Pick\ and\ Place$ environment where the robot had to catch a ball with a magnetized end effector and place it in another desired position. In the end, the robot did it with an average of 5 steps.

5 Conclusion

5.1 Thesis Summary

This study aimed to find new techniques for robotic assembly in space. First a design of bars that allows easier construction of truss structure has been presented. This design could also reduce the payload for large structures such as telescopes. One of the key techniques is the use of magnets. Magnetized balls were used as joints between the elements and magnets integrated on the sides of the bars and the back of the mirror again ease the assembly.

Afterward, the forward and inverse kinematic were presented for a better understanding of the operation of the UR10. Then the autonomous path planning using reinforcement has been explained. Reinforcement Learning is a rather complex technique but allows to have optimal results for path planning. Two policies were detailed, Q Learning and Deep-Q-Learning. This second policy uses Neural Network to find the best action and is more adapted to a continuous environment such as robotic.

Finally, the simulation and experiments were detailed. The simulation was first made using Q-Learning but the results were not conclusive. Then the Deep-Q-Learning has been used to solve the problem of path planning. The experiments with this policy showed that RL with Neural Network is well adapted to path planning because the robot was able to reach the target with the optimal path after training. The robot was able in the end to pick with a magnetize end effector and place elements for a future assembly.

5.2 Discussion and Future Works

This study has shown how efficient could be the use of reinforcement learning for path planning. However, the assembly has not been fully made. Taking into consideration the rotation of the wrist and the orientation of the bars would require a very powerful computer. The training could take more than 3 days if the computer is not adapted. So the future work would be to have a fully assembled structure using the technique of machine learning with the magnetized end effector. Also, more policies could be compared as this field of study is rather new, more and more techniques are arising every year.

An important future work would also be to test the program on the real robot. Also, a camera could be used to detect the object in the space to have again a more adaptable manipulator for the assembly.

Bibliography

[1] D. Akin, S. Brook. A survey of serviceable spacecraft failures[C]//AIAA Space 2001 Conference and Exposition. Albuquerque, USA: AIAA, 2001: 1-8.

[2] J. Dorsey, W. Judith. Space Assembly of Large Structural System Architectures[C]// AIAA SPACE 2016. Long Beach, California, USA: AIAA, 2016: 1-11.

[3] S. Mohan. Operational Impact of Mass Property Update for On -Orbit Assembly[C]// SpaceOps 2006 Conference. Rome: AIAA, 2006: 1-8.

[4] P. Williams, J. Dempsey, D. Hamill, et al. Space Science and Technology Partnership Forum: Value Proposition, Strategic Framework, and Capability Needs for In-Space Assembly[C]//2018 AIAA SPACE and Astronautics Forum and Exposition. Orlando, FL, USA: AIAA, 2018.

[5] D. Piskorz, K. Jones. On-orbit assembly of space assets: A path to affordable and adaptable space infrastructure[J]. The Aerospace Corporation 2018, 2018.

[6] W. Whittake. Robotics For Assembly, Inspection, And Maintenance Of Space Macrofacilities[C]//AIAA Space 2000 Conference and Exposition. USA: AIAA, 2000.

[7] D. Barnhart, P. Wills, B. S. et al. Creating a sustainable assembly architecture for next-gen space: The Phoenix effect.[C]//30th Space Symposiumn. Colorado Springs: DARPA, 2014.

[8] I. D. Boyd, D. P. Reina S. Buenconsejo, B. Lal, et al. On-Orbit Manufacturing and Assembly of Spacecraft[J]. IDA SCIENCE & TECHNOLOGY POLICY INSTITUTE, 2017.

[9] M. D. Lallo. Experience with the Hubble Space Telescope: Twenty Years of an Archetype[J]. Opt. Eng. 51, 011011, 2011.

[10] G. H. Kitmacher. Reference Guide to the international Space Station[M]. NASA, Washington DC: NASA, 2010.

[11] M. Lopez-Morales, K. France, F. Ferraro, et al. Another Servicing Mission to Extend Hubble Space Telescope's Science past the Next Decade[J]. ArXiv: Instrumentation and Methods for Astrophysics, 2019.

[12] J. Coleshill, L. Oshinowo, R. Rembala, et al. Dextre: Improving maintenance operations on the International Space Station[J]. Acta Astronautica, 2009, 64: 869-874.

[13] M. Hiltz, C. Rice, K. Boyle, et al. Canadarm: 20 years of mission success through adaptation[J]. Canadian Space Agency, 2020.

[14] J. Watson, T. Collins, H. Bush. A history of astronaut construction of large space structures at NASA Langley Research Center[C]//2002 IEEE Aerospace Conference: vol. 7. Big Sky, MT, USA: IEEE, 2002: 7-7.

[15] M. Rognant, C. Cumer, J. Biannic, et al. Autonomous assembly of large structures in space: a technology review[C]//8TH European Conference for Aeronautics and Aerospace Sciences (EUCASS). Madrid, SPAIN: EUCASS, 2019.

[16] J. Garibotti, A. Cwiertny, R. Johnson. On orbit fabrication and assembly of large space structural subsystems[J]. Acta Astronautica, 1980, 7(7): 847-865.

[17] W. L. Heard, H. G. Bush, R. E. Wallsom, et al. A mobile work station concept for mechanically aided astronaut assembly of large space trusses[M]. Langley Technical Report Server, USA: NASA, 1983.

[18] B. Jenett, C. Gregg, D. Cellucci, et al. Design of multifunctional hierarchical space structures[C]//2017 IEEE Aerospace Conference. Big Sky, MT, USA: IEEE, 2017: 1-10.

[19] P. Pressel. Generic telescope truss[C]//Analysis of Optical Structures: vol. 1991. International Society for Optics: SPIE, 2015: 50-56.

[20] M. Mikulas, R. Pappa, J. Warren, et al. Telescoping Solar Array Concept for Achieving High Packaging Efficiency[C]//2nd AIAA Spacecraft Structures Conference. Kissimmee, Florida: AIAA, 2015.

[21] M. Rhodes, R. Will, M. A. Wise. A Telerobotic System for Automated Assembly of Large Space Structures[M]. Administration N.A.S., USA: CreateSpace Independent Publishing Platform, 2018.

[22] Z. Xue, J. Liu, C. Wu, et al. Review of in-space assembly technologies[J]. Chinese Journal of Aeronautics, 2020.

[23] J. P. Gardner, J. C. Mather, M. Clampin, et al. The James Webb Space Telescope[J]. Space Science Reviews, 2006, 123: 485-606.

[24] W. R. Doggett. Modular Assembly: An Efficient Approach for Creation and Maintenance of Persistent Space Assets[C]//2019 IEEE International Conference on Robotics and Automation. Hampton, Va. 23681 USA: Mechanics, 2019.

[25] E. L. Gralla. Strategies for launch and assembly of modular spacecraft[D]. 2006.

[26] S. Mohan, D. Miller, J. Budinoff. Assembly of a large modular optical telescope (ALMOST)[J]. SPIE, 2008.

[27] W. R. Doggett, J. Dorsey. State of the Profession Considerations: NASA Langley Research Center Capabilities and Technologies for Large Space Structures, In-Space Assembly and Modular Persistent Assets[J]. Bulletin of the AAS, 2019, 51(7).

[28] E. Medzmariashvili, N. Tsignadze, Z. Gviniashvili, et al. Ideology for Creation the Large Size Space Reflectory Autonomous Complex[C]//37th ESA (The European Space Agency) Antenna Workshop. Noordwijk, Netherland: ESTEC, 2016: 1-9.

[29] W. R. Doggett. Robotic assembly of truss structures for space systems and future research plans[C]//Proceedings, IEEE Aerospace Conference: vol. 7. Big Sky, MT, USA: IEEE, 2002: 7-7.

[30] Y. Bai, X. Yang. Novel Joint for Assembly of All-Composite Space Truss Structures: Conceptual Design and Preliminary Study[J]. Journal of Composites for Construction, 2013, 17.

[31] M. A. Roa, K. Nottensteiner, A. Wedler, et al. Robotic Technologies for In-Space Assembly Operations[C]//Advanced Space Technologies in Robotics and Automation (ASTRA). Leiden, The Netherlands: ASTRA, 2017.

[32] K. Nottensteiner, T. Bodenmueller, M. Kassecker, et al. A Complete Automated Chain for Flexible Assembly using Recognition, Planning and Sensor-Based Execution[C]//Proceedings of ISR 2016: 47st International Symposium on Robotics. Munich, Germany: VDE, 2016: 1-8.

[33] K. Albee. Toward optimal motion planning for dynamic robots : applications on-orbit[D]. 2019.

[34] M. Diftler, W. Doggett, J. Mehling, et al. Reconfiguration of EVA Modular Truss Assemblies using an Anthropomorphic Robot[J]. AIP Conference Proceedings, 2006, 813: 992-999.

[35] S I. Nishida, H. Hirabayashi, T. Yoshikawa. A New Space Robot End-Effector for On-Orbit Reflector Assembly[C]//2006 9th International Conference on Control, Automation, Robotics and Vision. Singapore: IEEE, 2006: 1-6.

[36] W. Bluethmann, R. Ambrose, M. Diftler, et al. Robonaut: A Robot Designed to Work with Humans in Space[J]. Autonomous robots, 2003, 14: 179-197.

[37] M. Diftler, J. Mehling, M. Abdallah, et al. Robonaut 2 - The first humanoid robot in space[C]//IEEE International Conference on Robotics and Automation. Shanghai, China: IEEE, 2011: 2178-2183.

[38] B. A. Corbin, A. Abdurrezak, L. P. Newell Gordon, et al. Global Trends in On Orbit Servicing, Assembly and Manufacturing (OSAM)[J]. IDA SCIENCE & TECHNOLOGY POLICY INSTITUTE, 2020.

[39] S. Patané, E. Joyce, M. Snyder, et al. Archinaut: In-Space Manufacturing and Assembly for Next-Generation Space Habitats[C]//AIAA SPACE and Astronautics Forum and Exposition. Orlando, FL: AIAA, 2017.

[40] C. Koch, M. Jankovic, S. Natarajan, et al. Underwater Demonstrator for Autonomous In-Orbit Assembly of Large Structures[C]//15th International Symposium on Artificial Intelligence, Robotics and Automation in Space (i-SAIRAS) 2020. Online: SAIRAS, 2020.

[41] M. Mori, H. Kagawa, Y. Saito. Summary of studies on space solar power systems of Japan Aerospace Exploration Agency (JAXA)[J]. Acta Astronautica, 2006, 59: 132-138.

[42] R. Will, M. Rhodes, W. R. Doggett, et al. An Automated Assembly System for Large Space Structures[M]//A. A. Desrochers. Intelligent Robotic Systems for Space Exploration. Boston, MA: Springer US, 1992: 39-110.

[43] W. Doggett. Robotic assembly of truss structures for space systems and future research plans[C]//IEEE Aerospace Conference. Big Sky, MT, USA: IEEE, 2002: 7-3589.

[44] R. Hoyt, J. Cushing, J. Slostad. SpiderFab: Process for On-Orbit Construction of Kilometer-Scale Apertures[J]., 8 July 2013.

[45] D. L. Chandler. Assembler robots make large structures from little pieces[J]. MIT News, 2019.

[46] Y. Dai, Z. Liu, Y. Qi, et al. Spatial cellular robot in orbital truss collision-free path planning[J]. Mechanical Sciences, 2020, 11: 233-250.

[47] B. Doerr, R. Linares. Motion Planning and Control for On-Orbit Assembly using LQR-RRT* and Nonlinear MPC.[J]. ArXiv: Robotics, 2020.

[48] C. Zhou, B. Huang, P. Fränti. A review of motion planning algorithms for intelligent robotics[J]. ArXiv: Robotics, 2021.

[49] A. Javaid. Understanding Dijkstra Algorithm[J]. SSRN Electronic Journal, 2013.

[50] L. Jaillet, J. Cortes, T. Simeon. Transition-based RRT for path planning in continuous cost spaces[C]//2008 IEEE/RSJ International Conference on Intelligent Robots and Systems. Nice, France: IEEE, 2008: 2145-2150.

[51] S. Lavalle, J. Kuffner. Rapidly-Exploring Random Trees: Progress and Prospects[J]. Algorithmic and computational robotics: New directions, 2000.

[52] F. Bounini, D. Gingras, H. Pollart, et al. Modified artificial potential field method for online path planning applications[J]. 2017 IEEE Intelligent Vehicles Symposium (IV), 2017: 180-185.

[53] O. Khatib. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots[J]. The International Journal of Robotics Research, 1986.

[54] M. Z. Azmi, T. Ito. Artificial Potential Field with Discrete Map Transformation for Feasible Indoor Path Planning[J]. Applied Sciences, 2020, 10: 87-89.

[55] T. Evgeniou, M. Pontil. Support Vector Machines: Theory and Applications[C]//Machine Learning and Its Applications, Advanced Lectures. Leibniz, Germany: Springer, 2001: 249-257.

[56] S. Hochreiter, J. Schmidhuber. Long Short-term Memory[J]. Neural computation, 1997, 9: 1735-80.

[57] M. Kalos, P. Whitlock. Monte Carlo Method[M]. Weinheim, Germany: Wiley VCH, 2008.

[58] C. Remi, M. Pontil. Efficient Selectivity and Backup Operators in Monte-Carlo Tree Search[C]//5th International Conference on Computer and Games. Turin, Italy: INRIA, 2006.

[59] R. Sutton, D. Mcallester, S. Singh, et al. Policy Gradient Methods for Reinforcement Learning with Function Approximation[J]. Adv. Neural Inf. Process. Syst, 2000, 12.

[60] C. Daskalakis, D. J. Foster, N. Golowich. Independent Policy Gradient Methods for Competitive Reinforcement Learning[J]. ArXiv, 2021.

[61] L. Wang, Q. Cai, Z. Yang, et al. Neural Policy Gradient Methods: Global Optimality and Rates of Convergence[J]. ArXiv, 2019.

[62] M. Ryu, Y. Chow, R. Anderson, et al. CAQL: Continuous Action Q-Learning[C]// International Conference on Learning Representations. Virtual Conference: ICLR, 2020.

[63] J. Xiong, Q. Wang, Z. Yang, et al. Parametrized Deep Q-Networks Learning: Reinforcement Learning with Discrete-Continuous Hybrid Action Space[J]. ArXiv, 2018.

[64] T. Lillicrap, J. J. Hunt, A. Pritzel, et al. Continuous control with deep reinforcement learning[J]. CoRR, 2019.

[65] A. Franceschetti, E. Tosello, N. Castaman, et al. Robotic Arm Control and Task Training through Deep Reinforcement Learning[J]. ArXiv, 2020.

[66] A. Franceschetti, E. Tosello, N. Castaman, et al. Robotic Arm Control and Task Training through Deep Reinforcement Learning[J]. ArXiv, 2020.

[67] X. Xing, D. E. Chang. Deep Reinforcement Learning Based Robot Arm Manipulation with Efficient Training Data through Simulation[C]//2019 19th International Conference on Control, Automation and Systems (ICCAS). Jeju, Korea.: ICCAS, 2019: 112-116.

[68] C. Liu. A Multitasking-Oriented Robot Arm Motion Planning Scheme Based on Deep Reinforcement Learning and Twin Synchro-Control.[J]. SensorS, 2020.

[69] S. Li, X. Wang, L. Hu, et al. Mobile robot path planning based on Q-learning algorithm*[C]//2019 WRC Symposium on Advanced Robotics and Automation (WRC SARA). Beijing, China: IEEE, 2019.

[70] M. Ji, L. Zhang, S. Wang. A Path Planning Approach Based on Q-learning for Robot Arm[C]//2019 3rd International Conference on Robotics and Automation Sciences (ICRAS). Wuhan, China: ICRAS, 2019: 15-19.

[71] A. Zeng, S. Song, J. Lee, et al. TossingBot: Learning to Throw Arbitrary Objects with Residual Physics[J]. IEEE Transactions on Robotics, 2020.

[72] M. Plappert, M. Andrychowicz, A. Ray, et al. Multi-Goal Reinforcement Learning: Challenging Robotics Environments and Request for Research[J]. ArXiv, 2018.

[73] M. Lucchi, F. Zindler, S. Mühlbacher-Karrer, et al. Robo-gym – An Open Source Toolkit for Distributed Deep Reinforcement Learning on Real and Simulated Robots[J]. ArXiv, 2020.

[74] K. P. Hawkins. Analytic Inverse Kinematics for the Universal Robots UR-5/UR-10 Arms[J]. SMARTech, December 2013.

[75] H. Van Hasselt, A. Guez, D. Silver. Deep Reinforcement Learning with Double Q-learning[J]. ArXiv, 2015.

[76] T. Kobayashi, W. E. L. Ilboudo. T-soft update of target network for deep reinforcement learning[J]. Neural Networks, 2021.

[77] W. Smart, L. Kaelbling. Effective Reinforcement Learning for Mobile Robots[J]. Proceedings - IEEE International Conference on Robotics and Automation, 2002, 4.

[78] A. Panov, K. Yakovlev, R. Suvorov. Grid Path Planning with Deep Reinforcement Learning: Preliminary Results[J]. Procedia Computer Science, 2018, 123: 347-353.

[79] A. H. Qureshi, M. J. Bency, M. C. Yip. Motion Planning Networks[C]//2019 International Conference on Robotics and Automation (ICRA). Montreal, QC, Canada: IEEE, 2019: 2118-2124.

[80] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Playing Atari with Deep Reinforcement Learning[J]. ArXiv, 2013.

[81] V. Mnih, K. Kavukcuoglu, D. Silver, et al. Human-level control through deep reinforcement learning[J]. Nature, 2015, 518: 529-33.

[82] S. R. Company. Smart Grasping Sandbox[EB/OL]. https://github.com/shadow-robot/smart_grasping_sandbox. (accessed: 10.20.2020).

[83] E. Coumans, Y. Bai. Pybullet[EB/OL]. https://pybullet.org/. (accessed: 01.05.2021).

[84] O. Biza, D. Wang, R. W. Platt, et al. Action Priors for Large Action Spaces in Robotics[J]. ArXiv, 2021.

[85] P. Aumjaud, D. McAuliffe, F. J. Rodríguez-Lera, et al. Reinforcement Learning Experiments and Benchmark for Solving Robotic Reaching Tasks[J]. Advances in Physical Agents II, 2020.

[86] G. Brockman, V. Cheung, L. Pettersson, et al. OpenAI Gym[J]. ArXiv, 2016.

[87] F. Chollet. Keras[EB/OL]. https://github.com/keras-team/keras. (accessed: 01.17.2021).

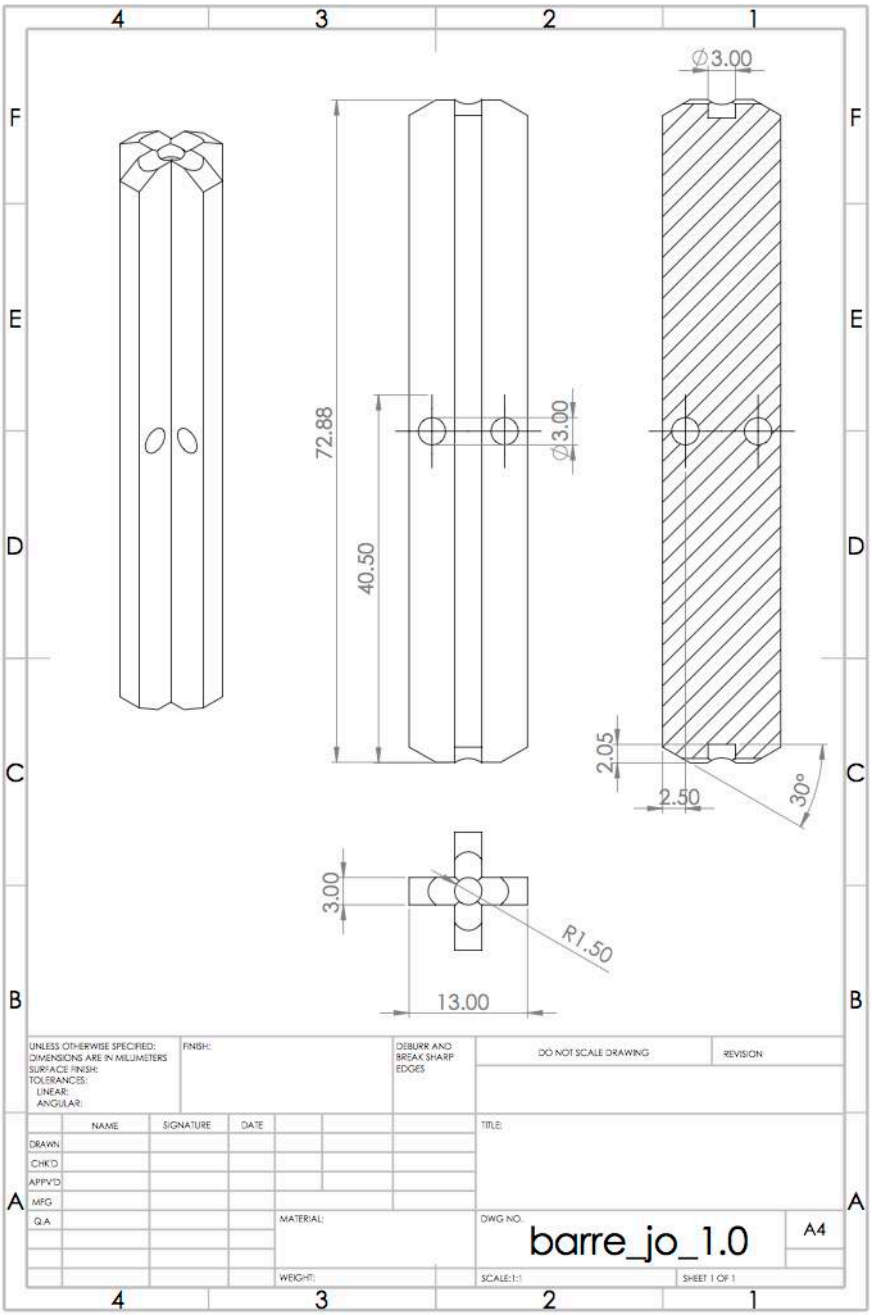
[88] M. Abadi, P. Barham, J. Chen, et al. TensorFlow: A system for large-scale machine learning[J]. ArXiv, 2016.

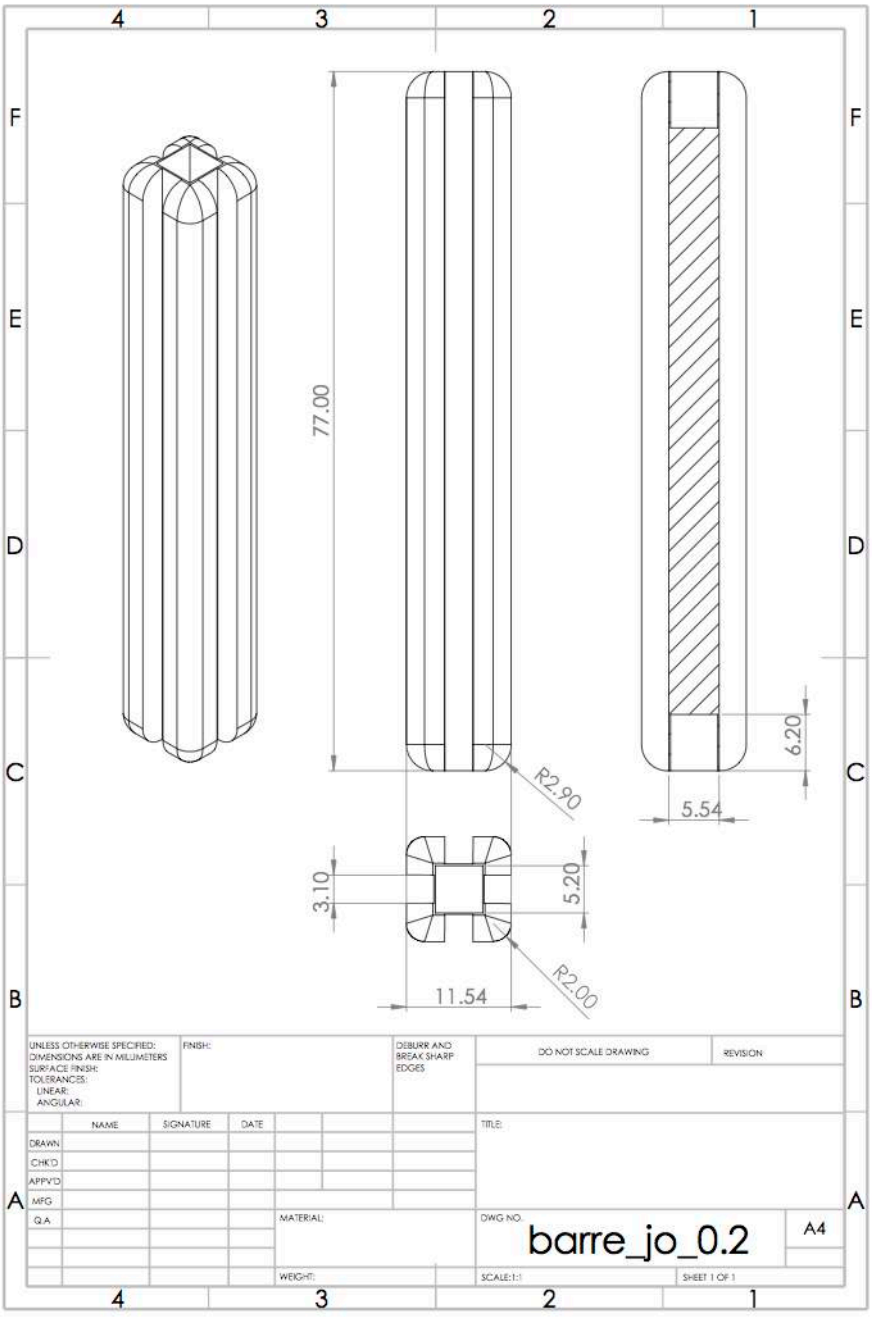
[89] W. Fedus, P. Ramachandran, R. Agarwal, et al. Revisiting Fundamentals of Experience Replay[J]. ArXiv, 2020.

[90] M. Brittain, J. Bertram, X. Yang, et al. Prioritized Sequence Experience Replay[J]. ArXiv, 2019.

Appendix

A.3 Bars plan




77.00
R2.90
6.20
5.54
3.10
5.20
R2.00
11.54
UNLESS OTHERWISE SPECIFIED:
DIMENSIONS ARE IN MILLIMETERS
SURFACE FINISH:
TOLERANCES:
LINEAR:
ANGULAR:
FINISH:
DEBURR AND BREAK SHARP EDGES
DO NOT SCALE DRAWING
REVISION
NAME
SIGNATURE
DATE
DRAWN
CHK'D
APPV'D
MFG
Q.A
TITLE:
MATERIAL:
DWG NO.
barre_jo_0.2
A4
WEIGHT:
SCALE:1:1
SHEET 1 OF 1

Acknowledgment

I would like to express my sincere gratitude to my supervisor, Prof associated Wang Ming Ming for allowing me to work under his supervision. Moreover, I thanks him for providing me continuous guidance, motivation, endless support, assistance, and feedback throughout the entire period of work. The conditions due to the pandemic led to a particular situation of working but he still managed to make me progress in my research and gave precious advice. The study wouldn't have been possible without his help and for that, I thank him very much.

Furthermore, I would like to thank the students of my supervisor who were present to give me support and advice during my thesis.

I finally thank my family and my friends especially Julien Mellet for the mental support, the precious guidance, and the help given in every stage of my research.

西北工业大学

学位论文知识产权声明书

本人完全了解学校有关保护知识产权的规定，即：研究生在校攻读学位期间论文工作的知识产权单位属于西北工业大学。学校有权保留并向国家有关部门或机构送交论文的复印件和电子版。本人允许论文被查阅和借阅。学校可以将本学位论文的全部或部分内容编入有关数据库进行检索，可以采用影印、缩印或扫描等复制手段保存和汇编本学位论文。同时本人保证，毕业后结合学位论文研究课题再撰写的文章一律注明作者单位为西北工业大学。

保密论文待解密后适用本声明。

学位论文作者签名： 

2021 年 07 月 12 日

指导教师签名： 

2021 年 07 月 15 日

- -

西北工业大学

学位论文原创性声明

秉承学校严谨的学风和优良的科学道德，本人郑重声明：所呈交的学位论文，是本人在导师的指导下进行研究工作所取得的成果。尽我所知，除文中已经注明引用的内容和致谢的地方外，本论文不包含任何其他个人或集体已经公开发表或撰写过的研究成果，不包含本人或其他已申请学位或其他用途使用过的成果。对本文的研究做出重要贡献的个人和集体，均已在文中以明确方式表明。

本人学位论文与资料若有不实，愿意承担一切相关的法律责任。

学位论文作者签名：_____

年　　月　　日