POLITECNICO
MILANO 1863

School of Industrial and Information Engineering
Master of Science in Space Engineering

A Q-learning based path planning approach for a space manipulator system

Author: **Nicolò Azzimonti**

Student ID: 944328

Advisor: Prof. Mauro Massari

Academic Year: 2020-2021

Abstract

Nowadays an increasingly number of space missions focus on the development of on-orbit servicing systems capable of accomplishing tasks such as repairing of satellites, removing of orbital debris, rendezvous and docking manoeuvres. Space manipulator systems (SMS), i.e. satellite-mounted manipulators, are expected to be a key technology for on-orbit servicing tasks.

Among the main challenges related to this technology, planning a suitable path to reach a target is probably the most demanding one. The existence of kinematic singularities and joint angles limits together with the need for keeping the target in sight of the chaser represent key issues that have to be taken into account; in addition, a reduced motion of the spacecraft base would be preferred in order to save on-board fuel. Moreover, from an extensive research on the current state of the art, it emerges that the current need is that of an autonomous SMS which is able to plan a trajectory from start to target, by adapting to different starting points.

Scope of this thesis is then to use a machine learning algorithm in order to train the SMS to generate autonomously a proper path of the end effector of the satellite-mounted manipulator. The selected approach is the Q-learning method, which is a simple, stable and reliable algorithm, frequently adopted for path planning.

The last part of the thesis eventually concentrates on the possible control logic to be implemented in order to follow the prescribed trajectory, both in full control and under-actuated control modes (to avoid controlling the base, hence saving on-board fuel).

To verify the implemented strategies, a complete simulation of the 3D behavior of the SMS has been carried out; the results show a good robustness of the path planning algorithm to different initial positions of the SMS end effector in the environment, together with the overcoming of the previously considered challenges. Furthermore, the implemented controllers prove to be efficient in tracking the states under control.

Keywords: Space manipulator system; Q-learning; Path-planning

Abstract in Italian

Al giorno d'oggi un numero sempre maggiore di missioni spaziali si sta concentrando sullo sviluppo di sistemi di on-orbit servicing in grado di svolgere compiti come la riparazione di satelliti, la rimozione di detriti orbitali, manovre di rendezvous e docking. I sistemi di manipolazione spaziale (SMS), ovvero i manipolatori montati su satellite, sono destinati ad essere una tecnologia chiave per le attività di manutenzione in orbita.
Tra le principali sfide legate a questa tecnologia, la pianificazione di una guida corretta per raggiungere il target è probabilmente la più impegnativa. L'esistenza di singolarità cinematiche e limiti meccanici degli angoli dei giunti, insieme alla necessità di mantenere il bersaglio nella visuale dell'inseguitore rappresentano limitazioni che devono essere prese in considerazione; per di più, un movimento ridotto della base sarebbe preferibile, in quanto permetterebbe di risparmiare propellente. Inoltre, da un'ampia ricerca sullo stato dell'arte, emerge che l'esigenza attuale è quella di un SMS autonomo che sia in grado di pianificare una traiettoria adattandosi a diverse posizioni di partenza.
Scopo di questa tesi è quindi quello di utilizzare un algoritmo di machine learning per addestrare l'SMS a generare autonomamente un'opportuna traiettoria dell'end effector del manipolatore. L'approccio scelto è il metodo Q-learning, che è un algoritmo semplice, stabile e affidabile, frequentemente adottato per scopi di guida.
L'ultima parte della tesi si concentra infine sulla possibile logica di controllo da implementare per seguire la traiettoria prescritta, sia in modalità di controllo completo che in modalità di controllo sottoattuato (al fine di evitare il controllo della base e risparmiare carburante).
Per verificare le strategie implementate è stata effettuata una simulazione completa del comportamento 3D dell'SMS; i risultati mostrano una buona robustezza dell'algoritmo di guida a diverse posizioni iniziali dell'end effector dell'SMS, insieme al superamento delle sfide precedentemente considerate. Inoltre, le tecniche di controllo implementate si dimostrano efficienti nel seguire gli stati sotto controllo.

Contents

List of Figures

List of Tables

1 | Introduction

Space manipulator systems (SMS) are considered as one of the most promising technologies for future space activities, thanks to their important role in various on-orbit servicing missions.
In this thesis the space manipulator technology is presented, studied and simulated.
The first chapter aims at introducing the SMS, focusing on their main applications and historical background. Following, a summary of the current state of the art for the technology is presented. Finally, the aim of the thesis is stated.

1.1. Manipulators in space

Nowadays robotic manipulators are extensively used.
The fabrication industry exploits manipulators to handle heavy objects and dangerous materials, as in welding processes. In the automotive industry most of the assembly phase is now performed using robotic systems. On the other side the surgical field pushed for the development of robotic arms for extremely precise medical operations.
Considering the advantages of manipulators and the results achieved with their use, the application of robotic arms in the space environment is a very interesting possibility.

As a general terminology, a SMS refers to a multi degree of freedom robotic arm mounted on a spacecraft that can be used for orbital operations. In particular, on orbit servicing (OOS) is a class of mission in which a servicer (chaser) spacecraft performs servicing tasks on a client (target) spacecraft. The space frontier offers a great variety of on-orbit tasks that can be performed by a SMS: from the repair of damaged structures, to the refuelling of a satellite (using the manipulator as a fuel pump); from the rendezvous and docking manoeuvers to the manipulation and de-orbiting of space debris.

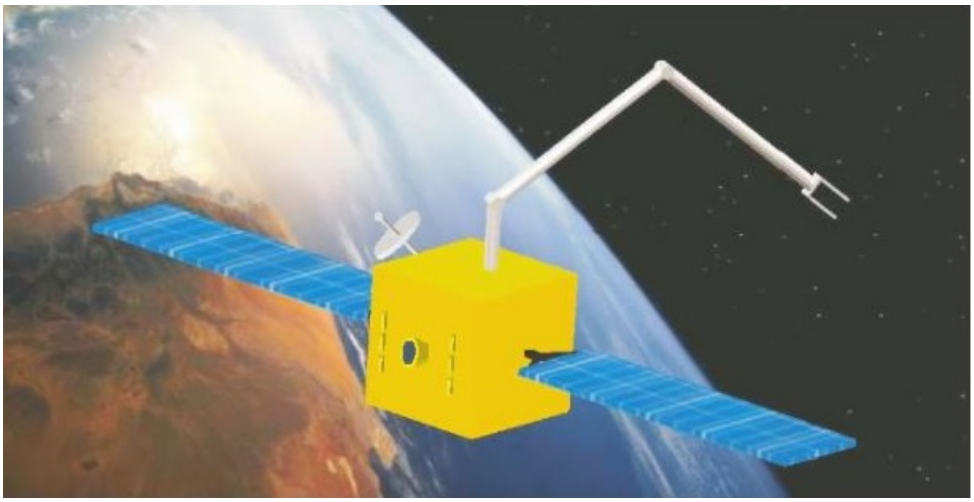


Figure 1.1: Graphical representation of a space manipulator system

One of the most attractive applications is the rendezvous and docking task. Using a robotic arm could improve the reliability of the operation: in this way the two docking spacecrafts could enter in contact in a more controlled and soft way, avoiding the risks involved in such complex operations.

An extremely attractive idea is to use the manipulator to repair the spacecraft itself and other target spacecrafts. The typical scenario could be that regarding a satellites constellation: the idea is that of placing on each orbit of the constellation a satellite equipped with tools able to perform accurate analysis of the damages occurred on other satellites. This feature could allow to save money for the maintenance of the satellites in the constellation. At the same time this maintenance capability could be applied on the spacecraft itself: during long interplanetary missions the capability of the spacecraft to perform analysis on its status could reveal crucial for the success of the mission.

Active debris removal is a relatively new application for space manipulators.
The need for debris reduction has become urgent: in LEO alone, there are an estimated 150 million fragments of human-created junk. These orbits are close to the Kessler limit, i.e. the point beyond which the debris population becomes self-perpetuating and grows uncontrollably. According to NASA, this limit is expected to be reached by 2055 [53].
To minimise debris in LEO, it is recommended their removal as the only effective solution: the most controllable approach consists in capturing the debris and manoeuvring it to a lower orbit (at LEO) or graveyard orbit (at GEO). This mission can be potentially carried out by an on-orbit servicer: from a SMS to a spacecraft mounting robotic tentacles, a harpoon or a deployable net. These have been the commonest proposals for debris removal.

1.2. Relevant space manipulators missions

In the recent space history there have been several technology demonstrator missions that witnessed the potentialities of this kind of technology.

Space manipulators have played a significant role on the ISS, which has installed on it three manipulator systems: the Canadian Mobile Servicing System (MSS), the Japanese Experiment Module Remote Manipulator System (JEMRMS), and the European Robotic Arm (ERA). These robotic manipulators and their predecessor, the Shuttle RMS (SRMS), have resulted essential for some human servicing tasks like providing mobile but stable footings for astronauts and, latterly, more sophisticated manipulation tasks, primarily orbital replacement unit (ORU) exchange on the ISS.

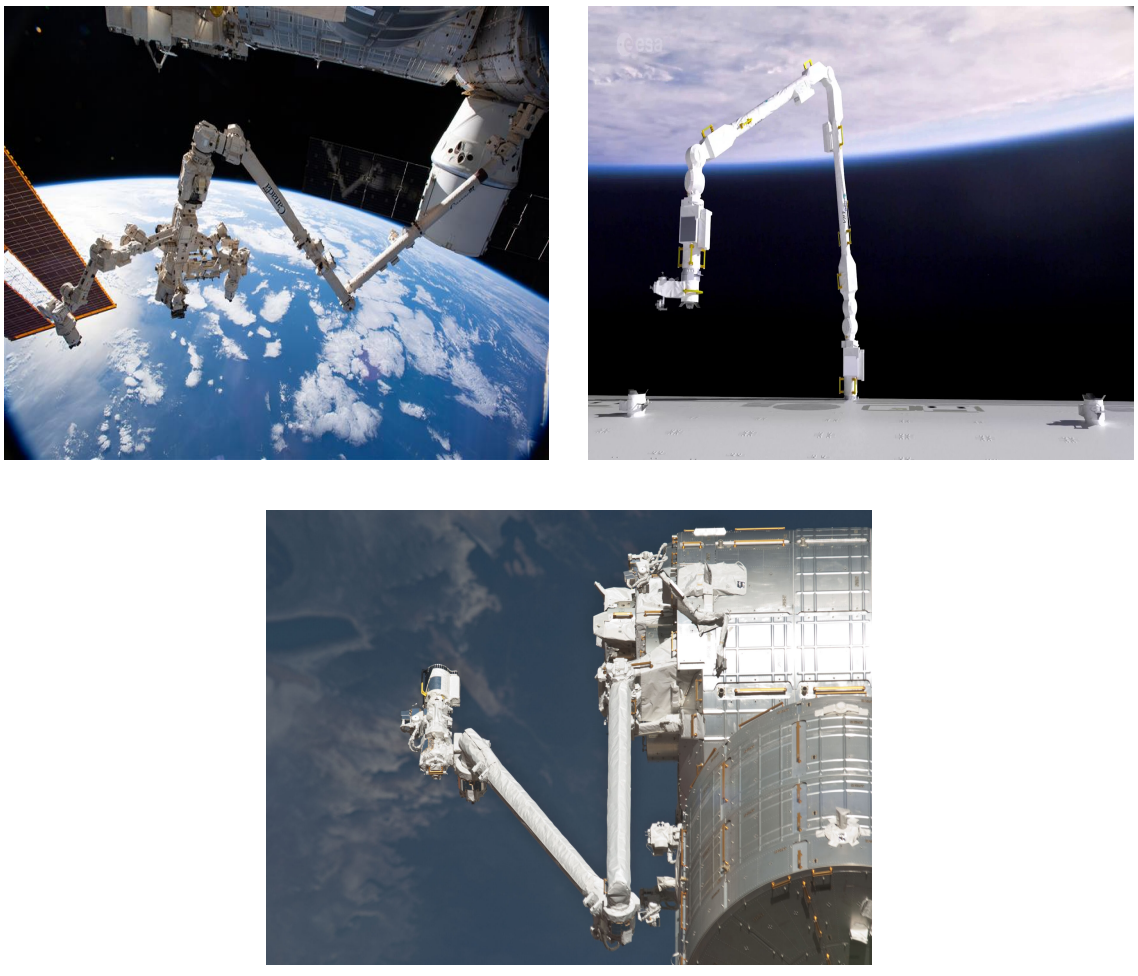


Figure 1.2: ISS manipulators: MMS on the top left, ERA on the top right, JEMRMS on the bottom

The Japanese Experimental Test Satellite, ETS-VII (1997), successfully tested a 6 DOF 2 m long robotic manipulator to demonstrate the feasibility of a rendezvous and docking between a chaser (Hikoboshi) and a target (Orihime). Furthermore, ETS-VII successfully demonstrated the feasibility of other on-orbit tasks like ORU exchange including bolt fastening and target berthing, ground teleoperation, handling of flexible object, mating/demating of electrical connectors, visual inspection of on-board equipment, etc.

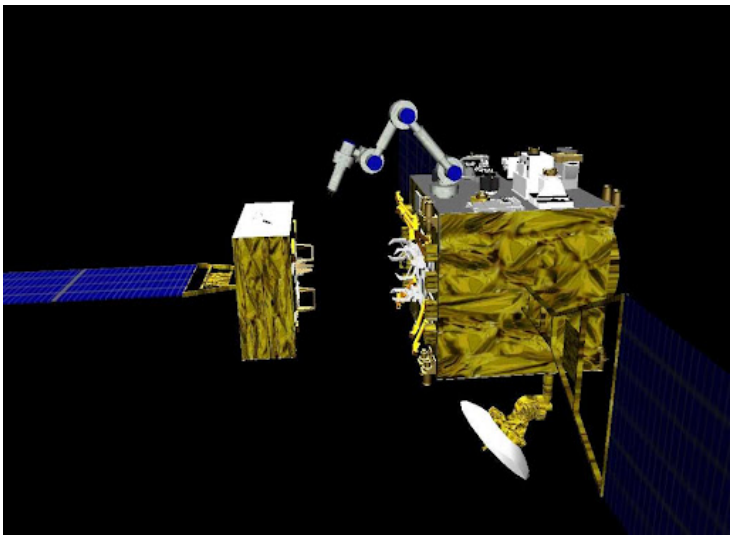


Figure 1.3: ETS-VII: computer graphics simulation for the target capture by the manipulator

Orbital Express (2007) aimed at demonstrating several cost-effective robotic servicing solutions including rendezvous, soft capture and mating, proximity operations, fluid transfer (specifically, hydrazine monopropellant), and ORU transfer. The system consisted of two spacecraft: the ASTRO servicing satellite, and a prototype modular next-generation serviceable satellite, NEXTSat. The Orbital Express Demonstration Manipulator System (OEDMS) was the mission's integrated robotics solution and consisted of a 6-DOF revolute joint robotic arm.

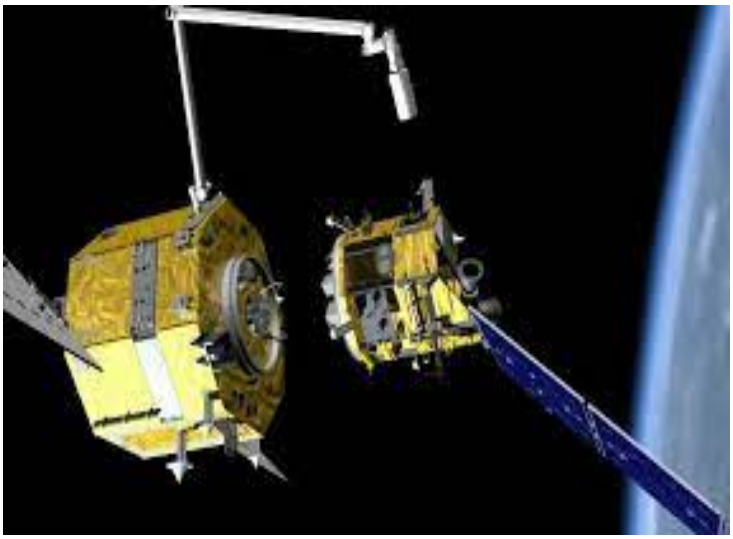


Figure 1.4: Orbital Express: ASTRO approaching NextSat

The ClearSpace-1 mission is an ESA Space Debris Removal mission led by Swiss company ClearSpace SA. The mission has as objective to demonstrate the feasibility and advantages of Active Debris Removal by removing a Vega payload adapter (Vespa Upper Part) from orbit. The mission, which will be launched in 2025, will be the first ever spacecraft to deorbit two non-functional satellites consecutively. ClearSpace-1 will use ESA-developed robotic arm technology to capture the Vespa, that consists of a four-armed space manipulator.

Figure 1.5: Clearspace-1 reaches Vespa

Finally, another mission led by DARPA aims at developing technologies that would enable cooperative inspection and servicing in geosynchronous Earth orbit (GEO). It will be based on the the use of a space robot, the Robotic Servicing of Geosynchronous Satellites (RSGS), for refuelling a target spacecraft in GEO. The launch of the mission is targeted for 2023.

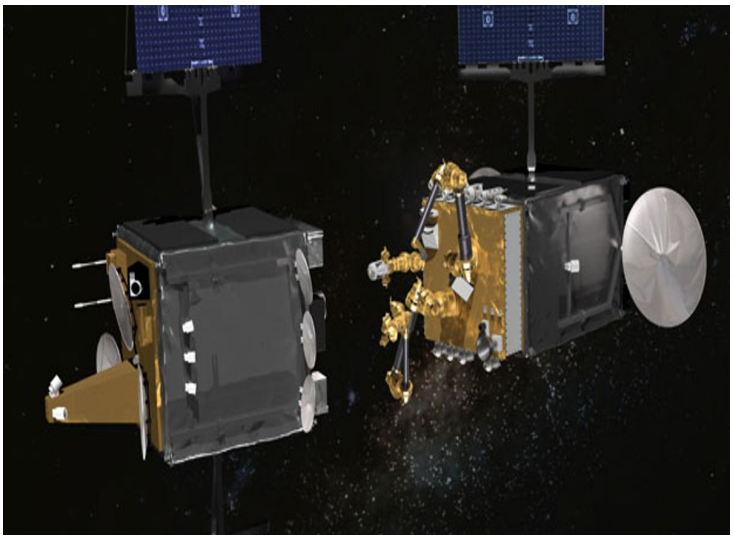


Figure 1.6: The RSGS spacecraft (right) will be able to approach, dock with, and service satellites in geosynchronous orbit

This concludes the historical background of this technology. Now that the SMS has been presented, an analysis of the current state of the art is pursued in order to better understand challenges and limits of this technology.

1.3. State of the art

The field of spacecraft manipulator systems has been studied in recent years; first studies date back to the early '90s, in particular at Massacchussets Institute of Technology (especially by S. Dubowsky, E. Papadopoulos and Z. Vafa), Stanford University (at the Aerospace Robotics Laboratory) and at Tohoku University (in particular K. Yoshida). Although research has been performed, the potential of space robots is still underexploited due to the challenges and limits they encounter on orbit.

Indeed, the space environment poses unique challenges to space manipulator systems survivability; not only microgravity but also extremely high or low temperatures, high vacuum or high pressure and ionizing radiations are critical issues that must be taken into account in order to enable a practical application.

Moreover an accurate modelling of the kinematics and dynamics of these systems is essential in order to perform the mission correctly; unmodelled dynamics, preliminary assumptions and possible errors in the plant may cause inaccuracies in the prediction of the behavior of the system.

Furthermore, to accomplish a specific task, a control system is necessary. This controller must generate torques and forces, by facing the very limited fuel storage present onboard the spacecraft.

Another important aspect as far as SMS missions are concerned is the one related to the guidance task. Planning suitable path is fundamental in order for space manipulators to carry out on-orbit service missions. The guidance problem is a topic that is already hard for ground based robots, and that is significantly more complicated in space. The complication comes from the fact that the robot is not mounted on an inertially fixed base, but rather on the satellite that can move in response to the manipulator motion, which in turn is affected by the base movement. This dynamical coupling obviously affects what one must command the robot to do in order to get the manipulator to reach a desired position in space.
When dealing with path planning, the existence of the singularities is another issue that has to be taken into account, since enormous joint velocities are induced when such singularities are met.
Moreover, the generation of the path has obviously to take into consideration the mechanical joint angles limits that characterise the robotic arm itself. Indeed, typically, the revolute joints composing the manipulator are not allowed to rotate of arbitrary angles, but are usually limited within a certain angular range that must not be overcome.
An additional constraint is then represented by the need for maintaining a certain orientation that allows the base to be always "looking" at the target, so that the information on target position can effectively be sensed.
Furthermore, the current trend, even if it's not a restrictive aspect, is that of planning a trajectory that guarantees a reduced motion of the spacecraft base; this would allow for a save in both on-board power and in fuel consumption as well, especially when a system of highly-consuming thrusters is used for the purpose.

In the following subsection the topic of path planning is detailed with the main conducted researches.

1.3.1. Main path planning methods

Many reasearches have been conducted in order to obtain a good solution for manipulator path planning. Here are reported the main methods that have been proposed in literature, both for ground based and space manipulators: from polynomial path planning methods to machine learning techniques, passing from solutions like Artificial Potential Field (APF), Probabilistic Road Maps (PRM), Rapidly exploring Random Tree (RRT) to cite some.

Sheng Nan Gai et al. presented a path planning algorithm for a 6-DOF industrial robot based on the artificial potential field method [54]. It is based on the definition of a potential function whose minimum corresponds to the desired position in space. However, this method can become trapped in the local minima of the potential field and fail to find a path or find a non-optimal path.

L.Kavraki et al. presented a path planning algorithm for robots based on Probabilistic road maps [63]. This method solves the problem of determining a path between a starting configuration of the robot and a goal configuration, by connecting different configurations with graphs. Nevertheless, this approach is time-consuming and requires re-planning when the initial position changes.

S.M.Lavalle [69] proposed a dynamic path planning method for robotic manipulator based on a rapidly exploring random tree algorithm. This method is based on generating trajectories by randomly building a space-filling tree. However, this method has many problems, including non-optimal paths. Additionally, like the PRM solution, this method is time-consuming and requires re-planning when the initial position changes.

Specifically related to space manipulators, S. Dubowsky and M. A. Torres [67] addressed path planning of space manipulator to minimize the disturbance to the space base.
Papadopoulos and Dubowsky have developed a method to plan a singularity-free path for a space manipulator.

Certainly, one of the trending topic nowadays is to try and exploit the potentialities of machine learning techniques.
S. Mohammad [25] offers an example of the use of supervised learning for motion of robotic arms; the aim is to learn discrete robot motions from a set of demonstrations. A a Gaussian Mixture Model is adopted in order to generalize the behavior of the system under consideration. Though the method proves the ability to generalize motions, this application to robot control suffers from the difficulty of ensuring stability towards the target.

Seyed Sina Mirrazavi Salehia [59] presents an approach for softly catching a flying object.

It uses a linear parameter varying system to model a trajectory that intercepts the target exactly at the desired predefined point. A GMM-based method is proposed for accurately approximating and modeling the parameters of LPV systems. However, the proposed learning algorithm is not convex, i.e., the performance of the system is dependent to the initialization.

In [60] Mengyu Ji et al. propose a path planning method based on Q-learning for a robot arm. Though the solution is efficient and has ability to plan the path for the robot arm, the motion of each joint needs to be planned an this dramatically complicates the design when a high number of DOF is considered.

Taiguo Li [31] introduces a Q-Learning autonomous learning algorithm to solve the problem of the path planning of a space manipulator in an unknown environment. The results show that the algorithm has the advantages of simple calculation, strong self-learning ability and adaptability to different environments. However, the proposed solution considers a too simple 2D environment, which limits the motions of the manipulator on a plane.

After a brief recap of the main researches, the following section introduces the solution proposed in this work, together with the motivation that has brought to it.

1.4. Motivation and proposed solution

From previous section, it is evident that the current need is that of an autonomous SMS which is able to plan an end effector trajectory from start to target by both adapting to different starting points and by overcoming the main challenges (singularity avoidance, joint and attitude limits, etc.). The most efficient way to introduce a certain autonomy in a system is that of adopting a machine learning technique. The advantages for using this algorithms is evident if we consider that machine learning makes a system more robust by allowing system decisions to be adjusted automatically.

Scope of this work is then to try and use a ML algorithm in order to train the SMS to generate autonomously a proper path of the end effector of the satellite-mounted manipulator. At the end of the training process, the goal is to have a system that, independently of the initial position of the end effector, results able to derive in autonomy a trajectory that overcomes the main challenges previously presented. This means that the space manipulator is trained to maintain a certain base attitude to guarantee that the target is always on sight of the space robotic chaser. In addition, during the motion the movements of the joints are guaranteed to stay within the mechanical joint angular limits. Moreover

the problem of singularity avoidance is overcome in order to guarantee a proper functioning of the SMS. Finally, the solution tries to promote the generation of a path that limits the displacement of the base of the SMS.
The machine learning algorithm adopted in this thesis is catalogued inside the reinforcement learning algorithms. These methods are very efficient when a system has to be trained to effectively take autonomous decisions: their rewards/punishment logic guarantees a certain reliability in the sense that, once trained, the system has "learnt the lesson" and the chances of making bad decisions are very small.
Moreover a simplified RL method, as Q-learning, that is the method adopted in this work, reduces the complexity due to using a simple environment, states, and actions.
More details on machine learning techniques and their use in the field of manipulator path planning are summed up in chapter 5.

Finally, the last part of the thesis deals with the investigation on the possible control logic to be implemented in order to follow the prescribed trajectory. Both full control and underactuated control are assessed, in order to verify the possibility to avoid controlling the translation of the base, hence allowing for a smaller consumption of fuel.

1.5. Structure of the thesis

The structure of the thesis is built up on 7 main chapters

- **Model description**
  The chapter illustrates the block scheme that has been implemented in order to study the behavior of the space robotic system, with an explanation of each of the blocks involved.

- **Kinematics**
  The chapter defines the kinematic transformation from one reference frame to another and hence from joint space (end effector space) to task space (base and manipulator space) and viceversa.

- **Dynamics**
  The chapter deals with defining the dynamical response of the system to external and internal forces. The equation of motion is derived, together with the computation of the main dynamical matrices.

- **Path planning**
  The chapter aims at showing the developed path planning algorithm. A first introduction to machine learning techniques is made in order to assess the theoretical

basis on which the algorithm is build up. Then the application to the case in exam is presented.

- **Control**
  The chapter deals with the implementation of the control logic selected for the purpose of commanding the system to follow the desired end effector trajectory. Both full control and underactuated control are assessed.

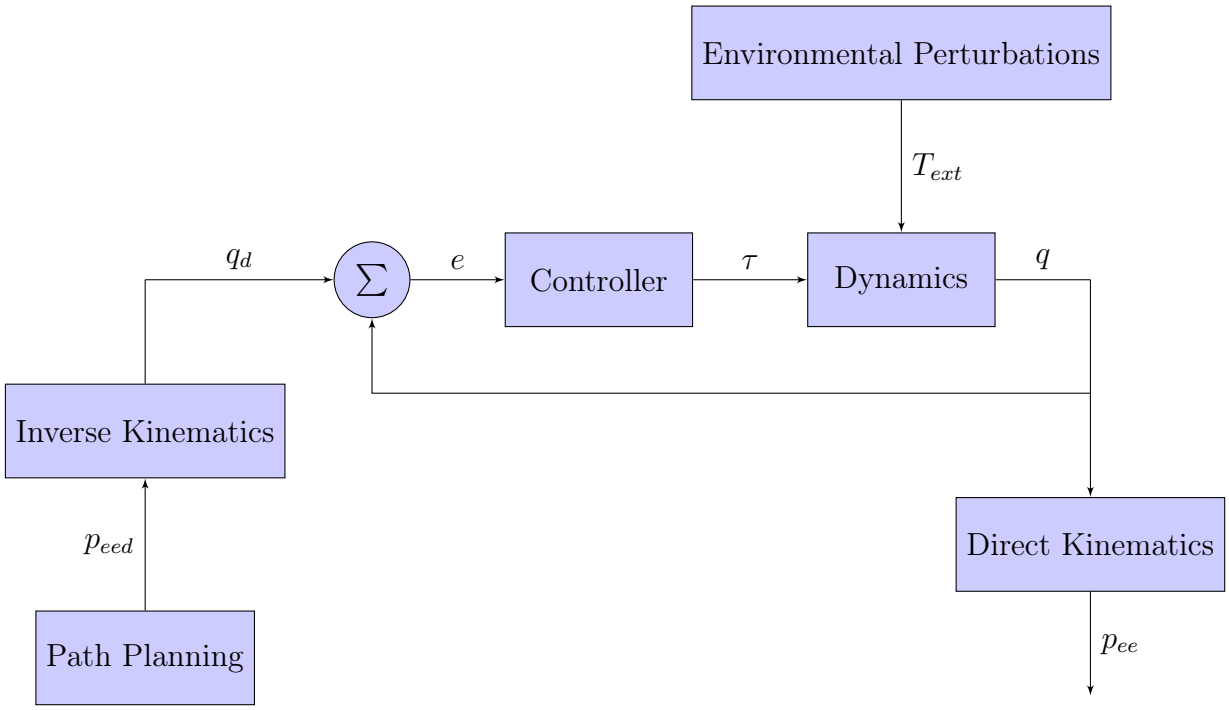
- **Results**
  The chapter initially presents the software used for the simulation of the SMS. Then, it introduces the numerical results obtained with the use of the developed path planning algorithm, analysing both the algorithm and its application. Finally the selected control approach is tested.

- **Conclusion and future work**
  At the end of the thesis, a summary of the relevant results obtained are reported, together with the possible future developments.

2 | Model Description

Usually, when dealing with dynamical systems in which control is involved, a block scheme is set up. Each element of the dynamical system is represented with a block that is the symbolic representation of the transfer function of that element. Indeed, it is easier to derive the transfer function of the control elements separately, instead of determining a general complex transfer function. Moreover, this representation helps in visualising the input-output correlations between the modules that rule the behavior of the entire system.

The idea on the basis of this work was then to split the SMS' behavior into the different modules that allow to study and then simulate all the functionalities of the system itself: from the dynamics and kinematics to the control task, from the generation of the reference to follow to the consideration of the disturbances affecting the system response.
The block scheme implemented in this thesis is reported in the following block diagram. This brief chapter aims at presenting a detailed explanation for each of these blocks.



- **Dynamics**
  The dynamics block contains the equation of motion governing the behavior of the space manipulator system (including both base and manipulator) when external forces and torques (mainly in the form of environmental disturbances and control actions) are introduced.
  It is possible to express the dynamics both in joint space (i.e. in terms of base pose and manipulator joint angles) and in task space (i.e. in terms of the end effector pose). In this thesis the equation of motion is derived in joint space.
  The forces and moments are taken as input to the dynamics block, which outputs the actual state vector containing joint space quantities.
  Of predominant importance for the equation of motion is the proper computation of two matrices that take into account the inertia and Coriolis forces that characterise the motion of the SMS, i.e. the Mass and Coriolis matrices, whose expression are showed later on in the work in chapter 4.

- **Direct Kinematics**
  The direct kinematics block contains the transformation rule necessary to perform the passage from the position of a point of the system in a local frame to the same position in the inertial frame. Since this transformation is ruled by some geometrical considerations, considering that point to be the end effector, this block also permits the passage from joint space to task space, i.e. it allows to express the end effector pose in function of manipulator joints angles and spacecraft base pose. At the end of the day, it takes as input the state vector coming from the dynamics block and outputs the end effector pose.

- **Path planning**
  Path planning block implements the designed method for the generation of the trajectory to follow in order to reach the target. As already stated in section 1.1, the idea of this thesis is to investigate a trajectory generation technique based upon a machine learning algorithm. Thus, the block outputs the required end effector path.

- **Inverse kinematics**
  The inverse kinematics block acts in the opposite direction with respect to the direct kinematics one, in the sense that it allows to pass from the task space to the corresponding joint space. It takes as input the desired end effector pose coming from the path planning generator and outputs the corresponding state configuration to be given to the control system.

- **Control**
  The control block aims at implementing the control logic required to make the system behave as desired, i.e. to command the end effector to follow the prescribed trajectory coming out of the path planning algorithm. The block receives as input the error between the prescribed and actual state vector coming from the dynamical block and outputs the required control torques and forces.

- **Environmental perturbations**
  This block computes the main environmental perturbations affecting the behavior of the system. In this work, the disturbing actions are considered to be just affecting the attitude of the base, i.e. the outputs of the block are torques coming from the aerodynamics, from the solar radiation pressure, from the magnetic field influence and from the gravity gradient. Eventually, after a simulation, one can see that the actual effects caused by the introduction of these disturbances is negligible.

For sake of completeness, it is remarked that a real block scheme should contain two extra blocks: a sensor block, that accounts for the measurements performed by sensors in order to get the system's states and an actuator block, that considers the devices that produce the actions required for the accomplishment of the control task. A sensor measurement is usually affected by errors that make the sensed states different from the actual ones, while the actuators, especially those involved in tracking tasks, are affected by saturation problems that limit the maximum control action applicable to the system.
The scope of this work is to generate a reference trajectory and test the system response to this input in order to verify the correctness of the methods introduced at theoretical level; in this case the assumption of perfect measurements and ideal actuators can be made and, thus, the two previous blocks can be neglected in the analysis.

In the following chapters all the blocks composing the scheme are treated in detail.

3 | Kinematics

In order to express the position of a point of the space manipulator system, the derivation of the system kinematics is essential.

The first part of this chapter is dedicated to the derivation of the direct kinematics equation that allows the passage from task space quantities to the corresponding joint space quantities. First of all the Denavit-Harteberg convention is decribed as it simplifies a lot the direct kinematic equations. Then the mathematical solution is detailed.
The chapter ends with the derivation of the inverse kinematics, which, instead, allows the passage from the task space to the joint space.

3.1. Denavit-Hartenberg convention

An efficient way to express the kinematics of a manipulator is to use the Denavit-Hartenberg convention. It is now detailed as reported in [4].

With reference to Figure 3.1, let axis i denote the axis of the joint connecting link $i-1$ to link i; the so-called Denavit–Hartenberg convention (DH) is adopted to define a link frame i, as follows:

- Choose axis z_i along the axis of Joint $i+1$
- Locate the origin O_i at the intersection of axis z_i with the common normal to axes z_{i-1} and z_i. Also, locate O_i at the intersection of the common normal with axis z_{i-1}.
- Choose axis x_i along the common normal to axes z_{i-1} and z_i with direction from joint i to joint $i+1$.
- Choose axis y_i so as to complete a right-handed frame

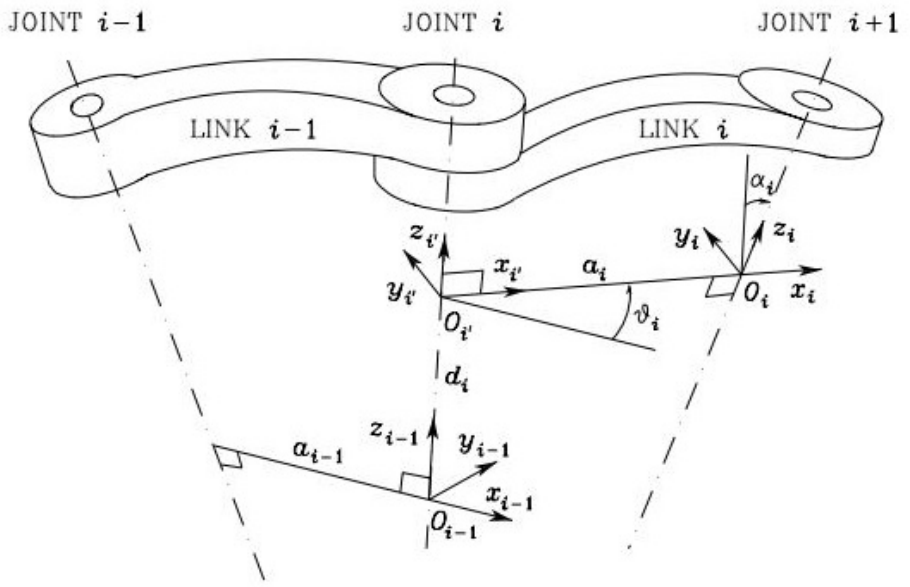


Figure 3.1: Denavit-Hartenberg convention

Once the link frames have been established, the position and orientation of Frame i with respect to Frame $i-1$ are completely specified by the following Denavit-Hartenberg parameters:

- a, distance between O_i and $O_{i'}$
- α, angle between axes z_{i-1} and z_i about axis x_i to be taken positive when rotation is made counter-clockwise
- d, coordinate of O_i along z_{i-1},
- θ, angle between axes x_{i-1} and x_i about axis z_{i-1} to be taken positive when rotation is made counter-clockwise.

Assuming all the joints as revolute joints, a, α and d are constant parameters and depend only on the geometry of connection between consecutive joints, while θ is varying in time. These parameters are sufficient to fully describe the kinematics of the space robot.

3.2. Direct kinematics

With this section the mathematical derivation of the direct kinematics equation is addressed, by exploiting the previously defined Denavit Hartenberg convention.
In literature, there exist two methods to compute the kinematics of a space manipylator system, namely, the Barycentric Vector Approach (BVA) and the Direct Path Method (DPM) [36].
The approach implemented in the thesis is the Direct Path Method. This method provides low computational cost and can be easily implemented in a symbolic environment.

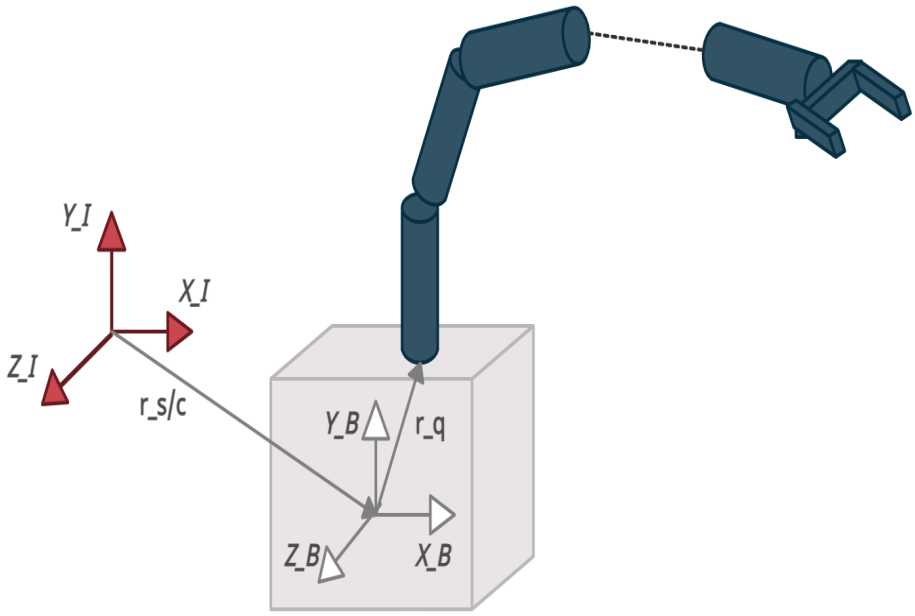


Figure 3.2: Schematic representation of a space manipulator system

Given the position of a generic point in the i^{th} joint frame, the position of the same point in the $(i-1)^{th}$ joint frame can be expressed as in Equation 3.1.

$$\begin{Bmatrix} \mathbf{r}_{i-1} \\ 1 \end{Bmatrix} = \boldsymbol{A}_i^{i-1}(\theta_i) \begin{Bmatrix} \mathbf{r}_i \\ 1 \end{Bmatrix} \tag{3.1}$$

where

$$\boldsymbol{A}_i^{i-1}(\theta_i) = \begin{bmatrix} \boldsymbol{R}_i^{i-1} & \boldsymbol{p}_i^{i-1} \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} = \begin{bmatrix} c_{\vartheta_i} & -s_{\vartheta_i}c_{\alpha_i} & s_{\vartheta_i}s_{\alpha_i} & a_i c_{\vartheta_i} \\ s_{\vartheta_i} & c_{\vartheta_i}c_{\alpha_i} & -c_{\vartheta_i}s_{\alpha_i} & a_i s_{\vartheta_i} \\ 0 & s_{\alpha_i} & c_{\alpha_i} & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.2}$$

This matrix is composed by a 3x3 matrix $\boldsymbol{R}_i^{i-1}$ that represents the rotation from the i^{th} joint frame to the $(i-1)^{th}$ one and by a 3x1 vector $\boldsymbol{p}_i^{i-1}$ that represents the position of the i^{th} joint with respect to the $(i-1)^{th}$ joint frame. It can be noticed the the definition of the DH parameters is sufficient to compute this matrix.
It is then possible to obtain the position of a generic point, and hence of each joint, in an inertial frame $\boldsymbol{r}_I$ by the consecutive rotation rule, i.e. through a matrix multiplication as in Equation 3.4.

$$\begin{Bmatrix} \mathbf{r}_I \\ 1 \end{Bmatrix} = \boldsymbol{A}_i^I(\theta_i) \begin{Bmatrix} \mathbf{r}_i \\ 1 \end{Bmatrix} \tag{3.3}$$

$$\boldsymbol{A}_i^I = \boldsymbol{A}_0^I \boldsymbol{A}_1^0 \ldots\ldots \boldsymbol{A}_i^{i-1} = \begin{bmatrix} \boldsymbol{R}_i^I & \boldsymbol{p}_i \\ \mathbf{0}_{1x3} & 1 \end{bmatrix} \tag{3.4}$$

Through Equation 3.3, the position of the end-effector of the manipulator can be found and the transformation matrix $\boldsymbol{A}_{ee}^I$ can be found through the product in Equation 3.5

$$\boldsymbol{A}_{ee}^I = \boldsymbol{A}_0^I \boldsymbol{A}_1^0 \ldots\ldots \boldsymbol{A}_{ee}^n \tag{3.5}$$

where $\boldsymbol{A}_{ee}^n$ is the matrix representing the orientation of the end effector frame with respect to the last joint frame.

Analogously, through geometric considerations, the position of the end effector can be written explicitely as in the following Equation 3.6

$$\mathbf{r}_{ee} = \mathbf{r}_{s/c} + \mathbf{R}_0^I \, \mathbf{r}_q + \sum_{i=1}^{n} \mathbf{R}_{i-1}^I \mathbf{p}_i^{i-1} \tag{3.6}$$

where $\mathbf{r}_{s/c}$ is the position of the base centre of mass in the inertial frame, $\mathbf{r}_q$ is the position of the manipulator's first joint with respect to the base centre of mass, as in Figure 3.2.

By derivating the previous expression it is possible to obtain an expression that leads

to the velocity of the end effector starting from the joint space velocities $\dot{\boldsymbol{q}}$. The overall result is expressed in the following Equation 3.7

$$\mathbf{v}_{ee} = \mathbf{J}\dot{\boldsymbol{q}}, \tag{3.7}$$

where $\boldsymbol{J}$ is called Jacobian of the system.

In previous equation

$$\mathbf{J} = \begin{bmatrix} \mathbf{I} & (\mathbf{r}^{\times}_{ee\_s})^T & \hat{\mathbf{k}}_1 \times (\mathbf{r}_{ee} - \mathbf{r}_1) & \cdots & \hat{\mathbf{k}}_n \times (\mathbf{r}_{ee} - \mathbf{r}_n) \\ \mathbf{0} & \mathbf{I} & \hat{\mathbf{k}}_1 & \cdots & \hat{\mathbf{k}}_n \end{bmatrix} \tag{3.8}$$

where $\mathbf{r}_{ee\_s} = \mathbf{r}_{ee} - \mathbf{r}_{s/c}$ and the symbol $^{\times}$ denotes the skew-symmetric matrix built up on vector $\boldsymbol{r}_{ee\_s}$, i.e.:

$$[\boldsymbol{r}_{ee\_s} \wedge] = \begin{bmatrix} 0 & -r_{ee\_sz} & r_{ee\_sy} \\ r_{ee\_sz} & 0 & -r_{ee\_sx} \\ -r_{ee\_sy} & r_{ee\_sx} & 0 \end{bmatrix} \tag{3.9}$$

The rotation axis of each joint of the manipulator in the inertial frame $\hat{\boldsymbol{k}}_i$ is found as in Equation 3.10, where n is the number of DoF (joints) of the manipulator

$$\hat{\boldsymbol{k}}_i = \boldsymbol{A}_0^I \boldsymbol{R}_1^0 .... \boldsymbol{R}_i^{i-1} \begin{Bmatrix} 0 \\ 0 \\ 1 \end{Bmatrix} \quad 1 \leq i \leq n \tag{3.10}$$

3.3. Inverse kinematics

As opposed to the direct kinematics, the inverse kinematics allows the passage from a certain pose of the end effector in task space to the corresponding quantities in joint space. The reason why the inverse kinematics procedure is important is that generally the reference motion that the end effector has to follow is expressed in the task space. Anyway, in order to control the base and the manipulator it is necessary to map the information in task space into the joint space. Indeed, usually controllers are built in a way that they take the required joint space quantities as inputs and outputs the corresponding control action.

Let's start from the direct kinematics equation

$$\mathbf{v}_{ee} = \mathbf{J}\dot{\boldsymbol{q}} \tag{3.11}$$

The most adopted approach for solving the inverse kinematics of a space manipulator is that of using the Moore-Penrose pseudoinverse of the system Jacobian

$$\mathbf{J}^* = \mathbf{J}^T(\mathbf{J}\mathbf{J}^T)^{-1} \tag{3.12}$$

.

The inverse kinematic equation is hence given by Equation 3.13

$$\dot{\boldsymbol{q}} = \mathbf{J}^*\mathbf{v}_{ee} \tag{3.13}$$

Since the Jacobian is, in general, a function of the configuration $\boldsymbol{q}$ those configurations at which

$$det(\boldsymbol{J}\boldsymbol{J}^{\boldsymbol{T}}) = 0 \tag{3.14}$$

and the pseudo-inverse cannot be computed are termed kinematic singularities. These conditions have to be avoided since in the neighbourhood of a singularity, small velocities in the task space may cause large velocities in the joint space.

In order to pass from a desired end effector pose to the correspondent joint configuration an iterative Newton method can be exploited. For each iteration

$$\boldsymbol{q}_{k+1} = \boldsymbol{q}_k + \boldsymbol{J}^*\boldsymbol{e} \tag{3.15}$$

Here $\boldsymbol{q}_k$ is an initial guess, while $\boldsymbol{e}$ is the error between the desired end effector pose and the one obtained with the direct kinematics.

4 | Dynamics

The derivation of the dynamic model of the space manipulator system plays an important role for the simulation of motion. Simulating manipulator motion allows control strategies and motion planning techniques to be tested without the need to use a physically available system.
This chapter is dedicated to the derivation of the equation of motion expressing the dynamics of the space manipulator system, together with the computation procedure of the main dynamical matrices that rule the system behavior.

4.1. Equation of motion

There are two well-established methods for deriving the equation of motion of an open-chain multi-body system, like the space manipulator: the Lagrange method and the Newton-Euler method.
The Newton-Euler method is based on the description of the reciprocal interactions in terms of momentum and forces between each link of the manipulator and between the manipulator itself and the base. This method is not suitable for simulation, as it requires the elimination of internal forces and moments that do not affect the motion of the system.
On the other hand, the Lagrange-Euler method is based on the consideration of the kinetic and potential energy of the system, whose values are expressed with the use of generalised coordinates. This is a more systematic method which makes it simpler than the Newton-Euler method; anyway it requires the computation of complex derivatives that increase the computational demand.
These two methods are equivalent as they result in the same solution to the equation of motion, but they differ when it comes to their approach and computation burden.
In this work the Lagrangian approach has been carried out due to its suitability for simulation, that makes it the most used approach when modeling a dynamical system.

Before deriving the equation of motion, some common assumptions have been previously considered:

- The chaser body and manipulator links are assumed to be rigid, thus no flexibility effects are accounted for.
- The effects of the orbital mechanics on the system dynamics are ignored, because they are insignificant compared to the control forces and torques and since the time and distance scale of robotic motions are assumed to be relatively small compared to the orbital period [41][52].
- Due the small presence of micro-gravity in orbit the potential energy is considered null [2].

Following the Lagrangian formalism, the total energy of the system is just made up of the kinetic one, as follows

$$\mathcal{L} = \mathcal{T} = \frac{1}{2} \sum_{i=0}^{N} \left(\boldsymbol{\omega}_i^T I_i \boldsymbol{\omega}_i + m_i \dot{\boldsymbol{r}}_i^T \dot{\boldsymbol{r}}_i \right) \tag{4.1}$$

where $\boldsymbol{r}_i$ is the distance of the i^{th} body centre of mass while $\boldsymbol{\omega}_i$ are the angular velocities of both manipulator's joints and satellite base.

Solving the Lagrangian Equation 4.2, where $\boldsymbol{T}_{nc}$ is the vector collecting the non conservative forces acting on the system and $\boldsymbol{q}$ is the generalized coordinate,

$$\frac{\partial}{\partial t} \left(\frac{\partial \mathcal{L}}{\partial \dot{\boldsymbol{q}}} \right) - \frac{\partial \mathcal{L}}{\partial \boldsymbol{q}} = \boldsymbol{T}_{nc} \tag{4.2}$$

after some algebraic passages, the equation of motion of a space manipulator can be derived. It is expressed as reported in Equation 4.3

$$\mathbf{M}(\mathbf{q})\, \ddot{\mathbf{q}} + \mathbf{C}(\mathbf{q}, \dot{\mathbf{q}})\, \dot{\mathbf{q}} = \boldsymbol{\tau} + \mathbf{T}_{s/c} + \mathbf{T}_{ee} \tag{4.3}$$

In this equation, the dynamics of the SMS are presented to be highly nonlinear, coupled, time varying and configuration dependent.

In Equation 4.3

- $\dot{\mathbf{q}}$ is the $(6+n)x1$ state vector (where n indicates the number of DoF of the manipulator) collecting the linear velocity of the spacecraft base $\mathbf{v}_{s/c}$, its angular velocity $\boldsymbol{\omega}_{s/c}$ and the manipulator joint velocities $\dot{\boldsymbol{\theta}}$.

$$\dot{\mathbf{q}} = \begin{Bmatrix} \mathbf{v}_{s/c} \\ \boldsymbol{\omega}_{s/c} \\ \dot{\boldsymbol{\theta}} \end{Bmatrix} \tag{4.4}$$

- $\boldsymbol{M}$ is the $(6+n)x(6+n)$ symmetric positive definite mass matrix of the system. This matrix can be split into blocks collecting the contribution of the base and of the manipulator, as follows

$$\boldsymbol{M} = \begin{bmatrix} \boldsymbol{H}_0 & \boldsymbol{H}_{0m} \\ \boldsymbol{H}_{0m}^T & \boldsymbol{H}_m \end{bmatrix} \tag{4.5}$$

  $\boldsymbol{H}_0$ is the satellite mass matrix; $\boldsymbol{H}_{0m}$ is the coupling term; $\boldsymbol{H}_m$ is the manipulator mass matrix

- $\mathbf{C}$ is the $(6+n)x(6+n)$ Coriolis matrix, collecting the contribution of centrifugal and coriolis forces and that can be split too, as follows

$$\boldsymbol{C} = \begin{bmatrix} \boldsymbol{C}_0 & \boldsymbol{C}_{0m} \\ \boldsymbol{C}_{0m}^T & \boldsymbol{C}_m \end{bmatrix} \tag{4.6}$$

  Once again, $\boldsymbol{C}_0$ is the satellite coriolis matrix; $\boldsymbol{C}_{0m}$ is the coupling term; $\boldsymbol{C}_m$ is the manipulator coriolis matrix

  The computation of each term of $\boldsymbol{M}$ and $\boldsymbol{C}$ matrices is explained in section 4.2 and section 4.3

- $\mathbf{T}_{s/c}$ is the $(6+n)x1$ external perturbation acting on the spacecraft:

$$\mathbf{T}_{s/c} = \begin{Bmatrix} \mathbf{F}_{s/c} \\ \mathbf{M}_{s/c} \\ \mathbf{0} \end{Bmatrix} \tag{4.7}$$

  where $\mathbf{F}_{s/c}$ is the perturbing force and is assumed to be null, i.e. no perturbation is affecting the orbit of the chaser, due to the preliminary assumption, while $\mathbf{M}_{s/c}$ is

the torque affecting the attitude of the spacecraft.

- $\mathbf{T}_{ee}$ is the $(6+n)x1$ external perturbation acting on the end effector:

$$\mathbf{T}_{ee} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \mathbf{J} \end{bmatrix} \begin{Bmatrix} \mathbf{F}_{ee} \\ \mathbf{M}_{ee} \end{Bmatrix} \tag{4.8}$$

  In this work T_{ee} has been assumed to be null.

- $\boldsymbol{\tau}$ is the $(6+n)x1$ control action required to obtain a desired behavior of the space manipulator system:

$$\boldsymbol{\tau} = \begin{Bmatrix} \boldsymbol{\tau}_{s/c} \\ \mathbf{Q}_{s/c} \\ \boldsymbol{\tau}_{ee} \end{Bmatrix} \tag{4.9}$$

  When dealing with a full control, all the components are different from 0. On the contrary, when an underactuated system is considered, as will be explained in chapter 6, one or more of these components are set to be null.

4.2. Mass matrix computation

In this thesis, the correct computation of the mass and coriolis matrices is essential not just for a more precise determination of the evolution of the states in time, but also for the implementation of the control logics that are based upon the dynamic model, as the computed control torque, presented in section 4.2.

As already stated, mass matrix $\mathbf{M}$ can be expressed as in Equation 4.5. The computation of each block is now detailed.

- $\boldsymbol{H}_0$ is the $6x6$ satellite matrix, that can be expressed as follows

$$\mathbf{H}_0 = \begin{bmatrix} \mathbf{A} & \mathbf{B} \\ \mathbf{B}^T & \mathbf{E} \end{bmatrix} \tag{4.10}$$

  The blocks in previous matrix are the following ones:

$$\mathbf{A} = \left(m_s + \sum_{i=1}^{n} m_i \right) \mathbf{I} \tag{4.11}$$

where m_s is the satellite mass, m_i is the mass of the i^{th} link of the manipulator and $\mathbf{I}$ is the identity matrix

$$\mathbf{B} = \left(m_s + \sum_{i=1}^{n} m_i \right) \mathbf{r}_{s\_g}^{\times} \tag{4.12}$$

where

$$\boldsymbol{r}_{s\_g} = \boldsymbol{r}_g - \boldsymbol{r}_{s/c} \tag{4.13}$$

with the vector $\boldsymbol{r}_g$ denoting the position of the center of mass of the space robot system, and the symbol $^{\times}$ denotes the skew-symmetric matrix of $\boldsymbol{r}_{s\_g}$.

$$\mathbf{E} = \mathbf{I}_s + \sum_{i=1}^{n} \left(\mathbf{I}_i + m_i (\mathbf{r}_{i\_s}^{\times})^T \mathbf{r}_{i\_s}^{\times} \right) \tag{4.14}$$

where $\boldsymbol{I}_s$ is the satellite inertia matrix , $\boldsymbol{I}_i$ is the inertia matrix of the i_{th} link of the manipulator and

$$\boldsymbol{r}_{i\_s} = \boldsymbol{r}_{ic} - \boldsymbol{r}_{s/c} \tag{4.15}$$

with $\boldsymbol{r}_{ic}$ representing the vector starting from the origin of the base-satellite reference frame and pointing towards the i_{th} link's centre of mass.

- Matrix $\boldsymbol{H}_{0m}$ is the $6xn$ matrix that introduces the dynamic coupling terms between the manipulator and the base satellite in terms of mass influence. Its expression is given by

$$\mathbf{H}_{0m} = \begin{bmatrix} \mathbf{D} \\ \mathbf{F} \end{bmatrix} \tag{4.16}$$

where

$$\mathbf{D} = \sum_{i=1}^{n} m_i \mathbf{J}_{Ti} \tag{4.17}$$

where $\boldsymbol{J}_{Ti}$ is the traslational part of the Jacobian , expressed as in Equation 4.18

$$\mathbf{J}_{Ti} = \left[\hat{\boldsymbol{k}}_1^{\times}\left(\boldsymbol{r}_i - p1\right), \hat{\boldsymbol{k}}_2^{\times}\left(\boldsymbol{r}_i - \boldsymbol{p}2\right), \cdots, \hat{\boldsymbol{k}}_i^{\times}\left(\boldsymbol{r}_i - \boldsymbol{p}i\right), \mathbf{0}_{3xN-i}\right] \quad \forall 1 \leq i \leq n \tag{4.18}$$

where $\hat{\boldsymbol{k}}_i$ is the unit vector indicating joint axis direction of the link i, $\boldsymbol{r}_i$ denotes the position of the mass center of the i^{th} link, $\boldsymbol{p}_i$ is the position vector of joint i.

$$\mathbf{F} = \sum_{i=1}^{n} \left(\mathbf{I}_i \mathbf{J}_{Ri} + m_i \mathbf{r}_{i\_s}^{\times} \mathbf{J}_{Ti}\right) \tag{4.19}$$

where $\boldsymbol{J}_{Ri}$ is the rotational part of the Jacobian , expressed as follows

$$\mathbf{J}_{Ri} = \left[\hat{\boldsymbol{k}}_1, \cdots, \hat{\boldsymbol{k}}_i, \mathbf{0}_{3xN-1}\right] \quad \forall 1 \leq i \leq n \tag{4.20}$$

- Regarding the manipulator inertia matrix $\boldsymbol{H}_m$, it contains the kinetic energy contribution, in both rotational and linear means, when considering just the manipulator; it has the same expression as the one for ground-based manipulators, as follows

$$\mathbf{H}_m = \sum_{i=1}^{n} \left(\mathbf{J}_{Ri}^T \mathbf{I}_i \mathbf{J}_{Ri} + m_i \mathbf{J}_{Ti}^T \mathbf{J}_{Ti}\right) \tag{4.21}$$

4.3. Coriolis matrix computation

The Coriolis matrix elements can be derived directly from the mass matrix elements as reported in the following Equation 4.22

$$\mathbf{C}_{ij} = \sum_{k=1}^{n} \left(\frac{\partial}{\partial q_k} m_{ij} - \frac{1}{2} \frac{\partial}{\partial q_i} m_{jk} \right) \tag{4.22}$$

In a first simulation, $\boldsymbol{C}$ matrix was calculated in this way. Anyway, this approach requires complex calculations for the partial derivatives that drastically increments the computational time.
In order to overcome this issue, many researches have been conducted. Among these, the method that makes use of the decoupled natural orthogonal complement matrix has been considered [5]. This allows to obtain an expression of the $\boldsymbol{C}$ matrix in which no partial derivatives are present.
According to this method, the $\mathbf{C}$ matrix is computed as in Equation 4.23

$$\mathbf{C} = \boldsymbol{\mathcal{N}}^T (\boldsymbol{\mathcal{M}} \dot{\boldsymbol{\mathcal{N}}} + \dot{\boldsymbol{\mathcal{M}}} \boldsymbol{\mathcal{N}}) \tag{4.23}$$

In previous equation $\boldsymbol{\mathcal{M}}$ is a $(6 * (n+1))x(6 * (n+1))$ matrix defined as

$$\boldsymbol{\mathcal{M}} = \text{diag} \left([\boldsymbol{M}_0, \boldsymbol{M}_1, \dots \boldsymbol{M}_N]^T \right) \tag{4.24}$$

where

$$\boldsymbol{M}_i = \begin{bmatrix} \boldsymbol{I}_i & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \boldsymbol{I}_{3x3} \end{bmatrix} \tag{4.25}$$

with $\boldsymbol{I}_i$ the i^{th} inertia matrix of the system.

The derivative of $\boldsymbol{\mathcal{M}}$ is then computed as follows

$$\dot{\boldsymbol{M}}_i = \begin{bmatrix} \boldsymbol{\omega}_i^{\times} \boldsymbol{I}_i & \mathbf{0}_{3x3} \\ \mathbf{0}_{3x3} & \mathbf{0}_{3x3} \end{bmatrix} \tag{4.26}$$

where $\boldsymbol{\omega}_i$ is the i^{th} body angular velocity.

Matrix $\boldsymbol{\mathcal{N}}$ is a $(6 * (n+1))x(6+n)$ matrix defined as the folowing product

$$\boldsymbol{\mathcal{N}} = \boldsymbol{\mathcal{N}}_l \boldsymbol{\mathcal{N}}_d \tag{4.27}$$

between a lower block triangular matrix and a diagonal matrix that are expressed as follows:

$$\boldsymbol{\mathcal{N}}_l = \begin{bmatrix} \boldsymbol{I}_{6,6} & \boldsymbol{0}_{6,6} & \cdots & \boldsymbol{0}_{6,6} \\ \boldsymbol{B}_{10} & \boldsymbol{I}_{6,6} & \cdots & \boldsymbol{0}_{6,6} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{B}_{N0} & \boldsymbol{B}_{N1} & \cdots & \boldsymbol{I}_{6,6} \end{bmatrix} \tag{4.28}$$

$$\boldsymbol{\mathcal{N}}_d = \begin{bmatrix} \boldsymbol{P}_0 & \boldsymbol{0}_{6,1} & \cdots & \boldsymbol{0}_{6,1} \\ \boldsymbol{0}_{6,6} & \boldsymbol{P}_1 & \cdots & \boldsymbol{0}_{6,1} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{0}_{6,6} & \boldsymbol{0}_{6,1} & \cdots & \boldsymbol{P}_N \end{bmatrix} \tag{4.29}$$

In previous equations the twist propagation matrices $\boldsymbol{B}_{i,j}$ and twist propagation vectors $\boldsymbol{P}_i$ appear. These quantities can be computed as follows

$$\boldsymbol{B}_{i,0} = \begin{bmatrix} \boldsymbol{I}_{3,3} & \boldsymbol{0}_{3,3} \\ \left(\boldsymbol{r}_{s/c} - \boldsymbol{p}_i\right)^{\times} & \boldsymbol{I}_{3,3} \end{bmatrix} \quad \forall i = 1 \dots n \tag{4.30}$$

$$\boldsymbol{B}_{i,j} = \begin{bmatrix} \boldsymbol{I}_{3,3} & \boldsymbol{0}_{3,3} \\ \left(\boldsymbol{p}_j - \boldsymbol{p}_i\right)^{\times} & \boldsymbol{I}_{3,3} \end{bmatrix} \quad \forall j = 1 \dots n, \forall i = 1 \dots n \tag{4.31}$$

$$\boldsymbol{P}_0 = \begin{bmatrix} \boldsymbol{A}_0^i & \boldsymbol{0}_{3,3} \\ \boldsymbol{0}_{3,3} & \boldsymbol{I}_{3,3} \end{bmatrix} \tag{4.32}$$

$$\boldsymbol{P}_i = \begin{bmatrix} \hat{\boldsymbol{k}}_i \\ \hat{\boldsymbol{k}}_i \wedge \boldsymbol{a}_i \end{bmatrix} \tag{4.33}$$

As a result, the method described in this section make the resulting equations of motion, and hence the simulation of the mechanical system, easier and computationally cheaper.

4.4. External perturbations

For sake of completeness, a brief explanation of the main environmental distrubances affecting the attitude of a spacecraft are here reported.
In space the spacecraft attitude is mainly influenced by the following environmental disturbances:

- **Gravity gradient torque.**
  The gravitational force is generating an unbalanced torque around the centre of mass of the spacecraft, whose effect is dependent on the inertia properties and on the attitude of the spacecraft itself
- **Solar radiation pressure torque.**
  The energy of the photons hitting each surface of the spacecraft produces a disturbance torque that depends upon the solar pressure, the properties of the surface and the angle between the surface itself and the Sun.
- **Magnetic field torque.**
  The magnetic disturbance is given by the interaction between the residual magnetic induction present onboard the spacecraft, mainly due to parasitic currents, and the Earth's magnetic field. This interaction produces a torque that is responsible for disturbing the attitude of the spacecraft.
- **Aerodynamic torque.**
  The aerodynamic force acts on each surface of the spacecraft, generating a torque. It depends upon the air density and upon the relative velocity of the spacecraft itself, both in magnitude and direction.

At the end of the day, supposing the system on a LEO, the typical intensity is in the order of $10^{-5}Nm$, fact that makes these disturbances barely affect the dynamics of the system, reason why they have been neglected from the analysis.

5 | Path Planning

This chapter introduces and describes the path planning task for a space manipulator system.
When the goal is that of reaching a specific position, a certain trajectory has to be tracked. The trajectory tracking problem consists of two steps, i.e. reference path planning and tracking control. The first task is treated in this chapter, the second one will be treated in chapter 6, which is dedicated to the control approach.

Path planning, in the case under investigation, refers to the process whose goal is to generate a trajectory that brings the end effector of the manipulator to a desired target position. Additional constraints can then be fulfilled, related to singularity and joint limits avoidance, fuel consumption, etc.
As already stated in the first chapter, the idea in this work is to investigate the potentiality of machine learning (reinforcement Q-learning in this research) to create an autonomous SMS that, once trained, is able to derive autonomously a proper trajectory from start to target.
First, an introduction to machine learning and its application on manipulator path planning is presented. Later on in the chapter, the developed method is described.

5.1. Machine Learning and its subdivisions

Nowadays machine learning is a widely used approach; from online shops to social media; from the financial field to the medical research; from voice recognition to self driving vehicles, this method proves to be an efficient, easy and safe approach.
Machine learning is a subfield of artificial intelligence which was defined in the 1950s by AI pioneer Arthur Samuel as “the field of study that gives computers the ability to learn without explicitly being programmed”.
In other words, it is a scientific discipline concerned with the design and development of algorithms that allow a system to evolve autonomous behaviors, in order to perform a task or find the optimal solution to a complex problem.

Machine learning is generally divided into subcategories, each of which adopts a different approach from the others. Here, the two main subfields of particular interest for robotics applications are presented: these are supervised learning and reinforcement learning

1. **Supervised learning**
   Supervised learning (SL) is a machine learning mechanism that first finds a generalized mapping between inputs and outputs coming from a given dataset, and then, based on this mapping, makes predictions to inputs contained in a different dataset. In this context, the set of examples with known labels is called the training set and the new set is called the test set. After a suitable number of examples have been given, the algorithm learns to generalize input-output relations.
   There are two types of problem: classification and regression.
   Classification problems ask the algorithm to recognize and categorize an object in a set of objects, i.e. identify the input data as a member of a particular class or group. An example is the image classification (like image recognition).
   Regression problems, instead, consist in predicting an output based upon a set of current datas, by understanding the relation between two or more variables. An example is predicting the price of a house based on its squared metres.

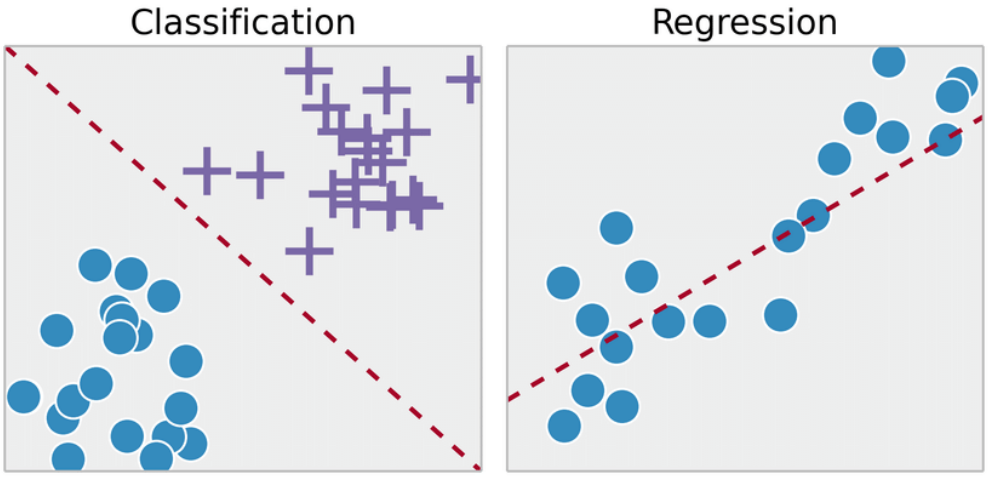


Figure 5.1: Supervised learning: classification vs regression

2. **Reinforcement learning**
   The reinforcement learning (RL) is a learning method in which the system to train interacts with its environment by performing actions and thus changing its state; consequently it is given punishments or rewards, depending on the action taken.
   This approach enables an agent to learn a mapping from states to actions by trial and error so that the expected cumulative reward in the future is maximized.
   Moreover, the model can correct the errors that occurred during the training process. Once an error is corrected by the model, the chances of occurring the same error are very less. Indeed, the peculiarity of reinforcement learning algorithms is that they maintain a balance between exploration and exploitation. Exploration is the process of trying different actions to see if the results are better than what has been tried before. Exploitation, instead, is the process of performing the actions that have worked best in the past.
   This learning approach is often used for example for videogames and in the robotic field (self-driving cars, autonomous rovers e.g.).

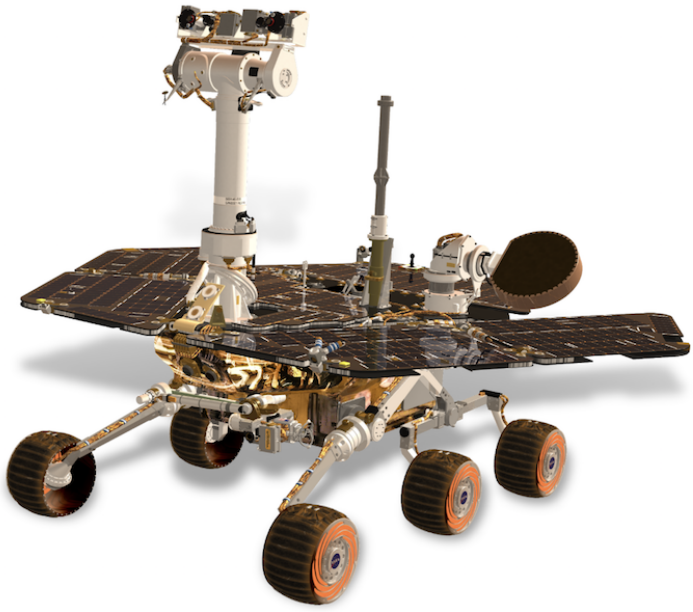


Figure 5.2: Mars Exploration Rover: its autonomy is based on reinforcement learning

From this brief description of the two approaches, it is evident that they differs in many aspects, thus presenting their own advantages and drawbacks.
Reinforcement learning differs from supervised learning since in supervised learning the training data contains the answer to a certain problem, so the model is trained with the correct answer; in reinforcement learning, instead, there is no training dataset and hence no answer, so the reinforcement agent decides what action to take on the basis of the rewards/punishments it gets, i.e. it learns from its experience.
In supervised learning, moreover, a huge amount of data is required to train the system to generalize an input-output formula, whereas in reinforcement learning the system or learning agent itself creates data on its own by interacting with the environment.
The absence of a dataset is a huge advantage of reinforcement learning techniques if compared with supervised ones. In supervise learning, indeed, when training the classifier, an as-complete-as-possible dataset, with a lot of good examples, is needed to make a generalization; otherwise, the accuracy of the model will be very poor and the output may be a wrong classification/regression. On the other side, reinforcement learning does not need datas to collect and categorise; anyway it needs a high number of interactions between the agent and the environment to reach the task.
Speaking about drawbacks, for both methods, usually, the training process needs a lot of computation time, especially if the data set or the possible states-actions couples are in large numbers. This issue represents a problem for the machine's efficiency.

5.2. Machine learning for manipulator path planning

Here are summed up the main strategies to adopt machine learning techniques for manipulator path planning.

- **Supervised learning**
  In the field of robotic path planning, a lot of researches have concentrated on the application of supervised learning algorithms. The idea is that of generating optimized trajectories by solving a system of equations that contain the equation of motion of the dynamical system under consideration and some optimal features that the trajectory is desired to have, like a control torque that is lower than a certain upper bound, a specific joint angles evolution, etc. After the generation of the training dataset, an approach, like the one exploiting Gaussian Mixture model, Neural networks, Decision Tree is selected in order to obtain the generalization of the input-output law.

- **Reinforcement learning**
  Robots are a natural application domain for reinforcement learning.
  In particular, this method is very suitable for autonomous manipulator path planning. In the case of manipulators, the input for the training process is the current state, which generally includes the angles of the joints or the position of the end effector; the output, instead, is a sequence of actions maximising the cumulated sum of rewards that the manipulator is receiving throghout the entire process. These rewards are established in order to plan a trajectory that guarantees the satisfaction of some criteria. Many criteria can be accounted for: fuel consumption, actuators saturation, joints position limits, singularities and obstacle avoidance are some of them.
  There are different available RL algorithms that can be used and that differs from the data collection approaches: Q-learning, Deep reinforcement learning, Actor-critic learning are the most explored ones.

Having analysed the current trends and advantages/drawbacks of the two approaches, this thesis investigates the potentiality of reinforcement learning for path planning of a SMS. The great potentialities of this method, especially applied in robotic field, have brought to this decision, together with the challenge deriving from the fact that it is still a widely unexplored field if the purpose is that of SMS trajectory generation.
In this work, the RL approach has the task of making the SMS learn how to plan its path through a direct interaction with the environment: as it moves, it will get rewards/punishments depending on the satisfaction of some constraints (singularity avoidance, etc.), so that at the end of the training process it will be able to take autonomously effective path-planning decisions.

Hence, a deeper description of RL is pursued.

5.3. Reinforcement learning

As shown in Figure 5.3, the main entities in a RL problem are always two: the agent and the environment.

The agent is the entity that has the aim of choosing which action to carry out in every instant of time: for the sake of this thesis, the agent would be represented by the space manipulator system.

The environment instead is the space in which the agent is moving and with which it is interacting, thus obtaining rewards and punishment. It may include the presence of obstacles that has to be avoided by the agent; in this thesis, due to the assumed vicinity of the target, an obstacle-free environment is considered.

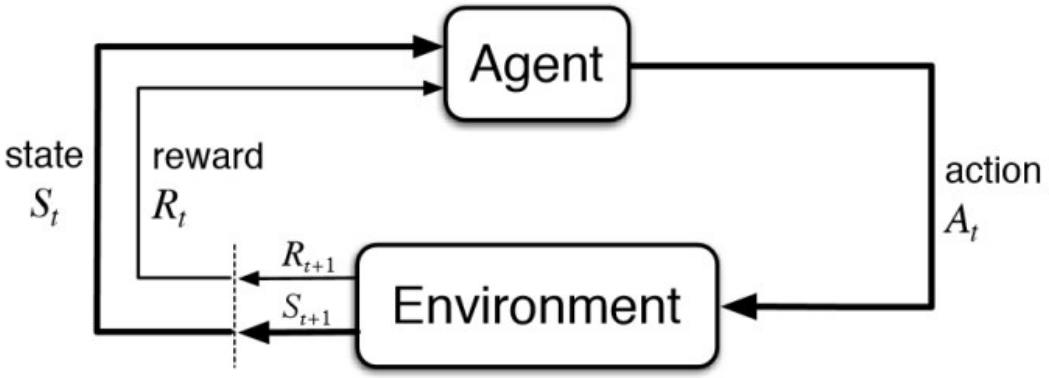


Figure 5.3: RL general scheme

Before starting the training, the agent does not know what are the correct actions to take because he does not have any knowledge of how the environment will respond. The training is therefore that process through which the agent comes to determine which action to take every time that it comes to a determined state. If the algorithm is set up correctly, after the training the agent will be able to perform the best action, given its current state, i.e. it will take actions according to an optimal policy.

Another element that is always present in RL process is the reward. Goal of the agent is to maximize the sum of the rewards obtained in time: we talk about the action-value function of a state-action couple Q(s,a) as the sum of total rewards that an agent can expect to accumulate in the future, starting from state s and taking action a. The reward assignment becomes therefore fundamental in this kind of problems: it must be connected to the physical objective that one wants to reach.

The world in which the system 'lives' is assumed to be a Markov decision process. Working with a Markovian system is an ideal condition for RL problems: the choice of the correct action at time t will depend only on the state of the system at time t and the state of the system at the next instant of time $t + 1$ will depend only on the state at time t and the action taken.

Among the different reinforcement learning methods, the Q-learning approach has been selected, due to its simplicity and well-known theoretical knowledge. Moreover, the Q

reinforcement learning method is very suitable for autonomous manipulator path planning and some researchers are directing in this way [61].
Hence, the following section introduces the features of this algorithm that will represent the theoretical base essential to apply it to the SMS under consideration.

5.4. Q-learning

The Q-learning algorithm is one of the most widely used RL algorithms. It is based on continuous update of the action-value function to reach the optimal policy. Once the training is over, the agent acts in this way: from a generic state s it takes the action a that it considers best, in the sense that it is the one for which the Q(s,a) is the largest.

5.4.1. Policy update

There are many ways to update the Q(s,a) function.
In this thesis the temporal difference approach has been selected. The big advantages of temporal difference method, compared to other methods like Dynamic Programming, are that to update the policy it is enough to wait for a single temporal step and in general it tends to be more efficient. Moreover, it exploits the Markov property, which makes it more effective in Markov environments.

Since the system is trained to modify its policy based on rewards and/or punishments it receives, it is a common practice to perform this updating procedure as a sum of the immediate reward resulting from a particular action in a particular state (Q(s,a)) plus a term called temporal difference error that compares the current Q-Value to the reward gained plus the maximal option available to the agent during the next state.
At each time step of the training, with the environment in state s, an action a is taken, the value of Q(s,a) is updated by a formula derived from the Bellman equation

$$Q'(s,a) = Q(s,a) + \alpha \left[r + \gamma \max_{a'} \left(Q\left(s', a'\right) \right) - Q(s,a) \right] \tag{5.1}$$

where

- Q(s,a) is the action value function for current state and action a
- α is the learning rate
- γ is the discount factor
- r is the reward associated to the choice of action a in state s

- $\max_{a'}(Q(s', a'))$ is the maximum value among the Q values of the future state

Changing parameters such as γ, α or the ratio of exploration to exploitation makes the final learning outcomes change significantly and hence influence the performance of the algorithm.

The discount factor γ $(0 \leq \gamma \leq 1)$ is a parameter that allows to select the importance given to future rewards by the agent. If $\gamma = 0$, the agent will only learn from actions that produce an immediate reward. If $\gamma = 1$, the agent will evaluate each of its actions based on the total sum of all of its future rewards. In other words, the closer is γ to 1, the more importance is given to the future rewards; on the contrary, the closer is γ to 0, the less relevant are the future rewards for the agent.

The learning rate α $(0 \leq \alpha \leq 1)$ is instead a parameter that indicates how fast the agent learns. The higher the learning rate the more the Q(s,a) changes if the quantity $[r + \gamma \max_{a'}(Q(s', a')) - Q(s, a)]$ is different from zero. Consequently, using a high α allows one to get close to the value of Q convergence faster. On the other hand, a high α may not always be the most effective choice, since in some cases the agent needs new information to be acquired.

5.4.2. Tabular Q-learning

The algorithm that was used in this thesis work is tabular Q-learning. In this case the values of each Q(s,a) are contained within a matrix called the Q-table. Each row of the matrix corresponds to a particular state, while each column corresponds to a particular action; each cell of the matrix contains the value of Q(s,a).
Before starting the training of the agent, the latter does not know yet the optimal policy. The initial value of each Q(s,a) present within the Q-table is theoretically arbitrary. Clearly the choice of this initial value can influence the training and indirectly the relationship between exploration and exploitation. A common choice is to select null initial values.
At each time instant t the value of the corresponding Q of the state s and action a is updated with the TD method.

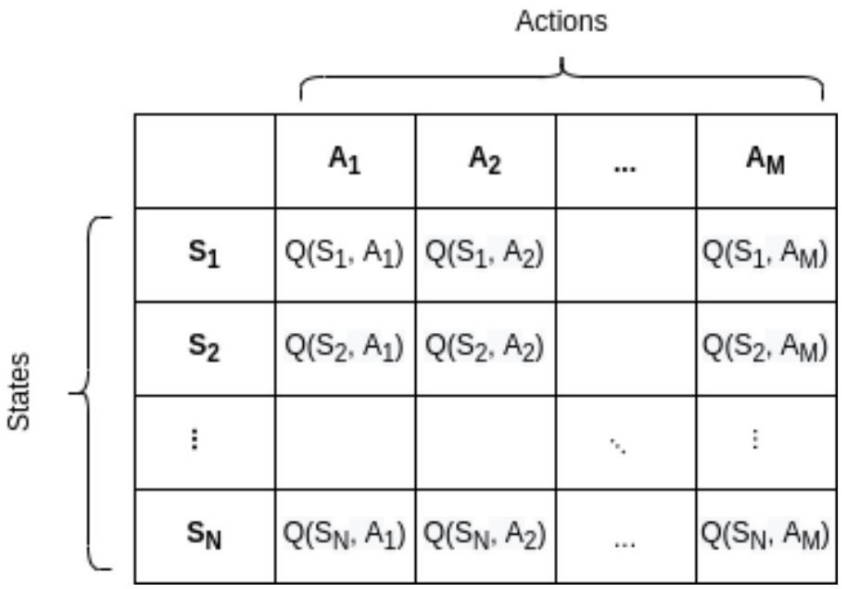


Figure 5.4: Q-table representation

This kind of Q-learning algorithm is based on the use discrete variables both for the states and the actions.
The state is usually discretized according to a discretization grid. A specific state corresponds to the discretized values of a certain physical quantity (the end effector position in this case).
Actions, too, must be discretized in the same way: commonly they are divided into the possible movements that are allowed to the system (right/left, up/down, etc).

5.4.3. Exploration and exploitation

As already stated at the beginning of the chapter, there are two modalities of choice of the action by the agent for every instant of time: exploration and exploitation.
In exploration, the agent explores the environment, in the sense that it randomly chooses one of the discretized actions, without considering the rewards that it could get by selecting an action rather than another.
In exploitation, the agent exploits what it has learnt up to then, in the sense that at state s the agent chooses the action a which it considers to be the best, i.e. the one for which there is the maximum value of Q(s,a).
As previously said, before starting the training, the agent does not know which are the best actions to take and the Q-table is initialized so that all values of Q(s,a) are equal to each other. In this case, exploration and exploitation can be considered synonyms.

During training, instead, there is a need to accurately balance the relationship between exploration and exploitation.

In fact, to always act in pure exploitation during all the training would lead to bad results because the agent would tend to take the same actions: once in a generic state s, the action to which corresponds the maximum value of Q(s,a) would be an action which has already been taken previously. Consequently, different actions, which could have led to better results would not be considered, since the system is just exploiting and is not exploring.

The other extreme condition of pure exploration is dangerous too: the agent would always take a random action in every state it is. In this way it is very unlikely that the agent turns out to be trained at the end of the simulation, since it is never exploiting what it has learnt.

The general effect of increased exploitation is the increase in the number of positive experiences, i.e. the number of episodes in which a higher final return is obtained compared to the previous episode [46]. However, it is also very important to have variations in the quality of such experiences and this happens only if exploration is sufficiently high. In general it is therefore necessary to carefully balance the relationship between exploration and exploitation.

5.4.4. ϵgreedy algorithm

One of the simplest and most widely used methods for balancing exploration and exploitation is the ϵ-greedy method. This method is based on selecting a parameter ϵ ($0 \leq \epsilon \leq 1$) on which the choice between exploration and exploitation is made: in each instant of time the agent takes a random action, i.e. explores, with a probability equal to $1 - \epsilon$, while exploits the lesson learnt with probability equal to ϵ. The parameter ϵ is therefore directly related to to exploitation: the higher it is, the greater the exploitation and vice versa.

A very simple and often effective choice could be leaving the parameter ϵ constant throughout the training: in this way the same amount of exploring, and hence exploiting, episodes is maintained during the whole training. Anyway, this solution does not always guarantee the best results.
It can be more effective, in fact, to use an increasing ϵ during the training in order to decrease the ratio between exploration and exploitation as the number of the training episode increases. Intuitively, indeed, at the beginning of the training, when the agent has performed few interactions with the environment, a high level of exploration would allow the agent to try and evaluate new actions; on the contrary, when the end of the training is approached, it is more appropriate to perform actions already explored in order

to generate episodes with high cumulative rewards, while maintaining at least a minimum of exploration to ensure the possibility of policy improvements.
The approach of increasing the ratio between exploration and exploitation obtained can be pursued by simply selecting ϵ as a function increasing with time, like a polynomial (linear, quadratic, cubic) function.

5.5. Q-learning approach for the space manipulator

After a detailed description of the Q learning theoretical basis, the current section presents the selected approach that applies the tabular Q-learning method to the path planning problem related to the SMS under consideration.
In this algorithm the states, i.e. the end effector positions, are discretised according to a 3D grid, composed by cubic cells. The agent can move from one cell of the grid to the other by selecting among 6 elementary actions. Everytime the end effector moves, the variables of the system are observed, in order to verify the fulfillment of some imposed constraints.
A more detailed description of the training process carried out by the algorithm is now conducted.

5.5.1. Training phase

Environment model

As already stated, the environment in which the agent moves is modeled as a 3D cubic grid, with cubic cells. Each action of the agent translates into a movement from a cell to another, depending on the action itself.
The selection of a 2D grid is usually preferred when mobile robots are concerned, especially because their motion can be modelled in a plane; for manipulators, instead, the selection of a 2D grid would limit the possible movements of the end effector, even though it would guarantee a lower compuatational demand. It has to be noticed, indeed, that an increase in the dimensions of the environment would increment the time it takes for the training phase.

Actions

A set of 6 elementary actions in a 3D space was considered: upward, downward, rightward, leftward, frontward and backwards: from its current cell, the end effector of the SMS moves towards the cell that corresponds to the action taken.
The selection of these actions goes in favor of a higher simplicity and is in compliance with the typical researches and applications that adopt the Q-learning algorithm. Obviously, the addition of extra-actions goes in favour of a higher completeness, although representing a computational limitation.

Algorithm 5.1 Pseudocode for actions implementation

if $action = upward \ \& \ cell < environmentz$ **then**
  $cell+ = [0, 0, 1]$
else if $action = downward \ \& \ cell > 0$ **then**
  $cell- = [0, 0, 1]$
else if $action = rightward \ \& \ cell > 0$ **then**
  $cell- = [0, 1, 0]$
else if $action = leftward \ \& \ cell < environmenty$ **then**
  $cell+ = [0, 1, 0]$
else if $action = frontward \ \& \ cell < environmentx$ **then**
  $cell+ = [1, 0, 0]$
else if $action = backward \ \& \ cell > 0$ **then**
  $cell- = [1, 0, 0]$
end if

Rewards

Everytime the agent takes an action, it receives a reward, depending on the fulfillment of some criteria: if the criteria are respected the system will receive a higher reward, if they are not the system will be punished. These constraints are selected on the basis of the challenges that a designer needs to face when dealing with a SMS, as explained in section 1.1. The selected conditions are:

- **Joint limits avoidance**
  Obviously, the robotic system must be able to reach the location not only in the simulation, but also in reality; hence a proper consideration of the mechanical limits of the joint angles must be made. Each joint has been consider to have a range of possible rotation of $\pm 90°$.

- **Singularity avoidance**
  As explained in section 3.3, a singularity is a condition in which a robotic system doesn't work properly. When dealing with path planning, singularity conditions must be avoided: this can be achieved by checking that the Jacobian matrix of the system fulfills the condition expressed by Equation 3.14

- **Target always in sight**
  In order to perform the mission correctly, the chaser base must constantly have the target in its sight, since this allows a proper functioning of the sensors installed in the base whose aim is that of detecting the position of the target. A range constraint of $\pm 60°$ has been considered. This means that the angle between the direction of the target and the direction of the spacecraft face looking at the target must be inside that range.

- **Small translation of the base**
  As already stated in section 1.1, it would be good, as far as fuel consumption is concerned, to promote those trajectories that allow for a lower displacement of the base, so that the fuel of the thrusters that are dedicated to the control of this displacement can be saved. An upper limit of 50% of the total displacement of the end effector has been considered.

It is then specified that the fulfillment of the first three constraints is essential to carry out the mission. If the agent finds itself in a position in which these conditions are not respected, the corresponding state is labeled as a terminal state: the current training simulation is stopped and a new one is performed, so that the system understands that it cannot pass from there.
On the other side, the last condition on the translation of the base is just an advantageous system behavior: if missed the mission can still be completed and the states in which this condition is not respected are not labeled as terminal one. Obviously the system will receive a higher reward for movements that allow for small base translation.

Action selection

As already stated, in order to select the action to take at each step of the algorithm the ϵ-greedy algorithm was implemented. In particular, for the sake of properly balancing the ratio between exploration and exploitation an increasing variation of the ϵ parameter over the training episodes has been considered, according to the following linear behavior:

$$\epsilon = \epsilon_i + \frac{(\epsilon_f - \epsilon_i)}{N_{ep}} N_{ep} \tag{5.2}$$

where ϵ_i and ϵ_f are the initial and final values of epsilon, taken equal to 0.1 and 0.9 respectively, while N_{ep} is the total number of episodes used for the training of the model, whose required value obviously increases with an increment in the dimensions of the environment and in the total number of actions considered.
The action selection procedure is summarised in Algorithm 5.2.

Algorithm 5.2 Pseudocode for action selection

Take a random value n, with
  $0 \leq n \leq 1$
if $n < \epsilon$ **then**
  $a \mid Q(s,a) = \max_a(Q(s,a))$
else
  $a =$ random action
end if

Q-Table design and updating procedure

After the introduction of states and actions, the Q-table can be created. In the algorithm developed in this thesis, it is represented by a 4 dimensional array, each element of which contains the value of the Q function corresponding to a certain state (3D position) and action couple, Q(s,a).
In every episode the Q-table is updated with the temporal difference method, as explained in Equation 5.1 and showed in Algorithm 5.3

Algorithm 5.3 Pseudocode for Q-table updating

1: **Take** action a:
   $s \longrightarrow s'$
2: **Get** r for the couple (s, a)
3: **Compute** the temporal difference error:
   $TD = r + \gamma \max_{a'} (Q(s', a')) - Q(s, a)$
4: **Update** the Q table:
   $Q'(s, a) = Q(s, a) + \alpha \cdot TD$

After many experiments, the parameters of the proposed Q-learning algorithm are set as follows: $\alpha = 0.9$ and $\gamma = 0.9$. In this way, with the selected α the algorithm would allow for a fast learning process; while the choice of γ allows to give more importance to the future rewards, like it is usually done in this applications [31]. Different tests have been executed in order to select the best parameters; these results are reported in the final chapter of this work.

Steps of the training process

Now that the main features and parameters have been introduced, it is possible to start the training algorithm. It consists in the following steps:

1. **Select an action**
   The system selects an action depending upon the value of ϵ parameter corresponding to the number of the episode currently running. In the beginning, ϵ is small, meaning that the system is more prone to explore the environment, while in the end ϵ becomes larger, thus allowing the system to take the best possible action with a higher probability.

2. **Assign Reward**
   Depending on the action selected at previous step, the system moves to the next state. Here the physical quantities under exam (singularity condition, angle with respect to the target, displacement of the base and joint angles) are checked: if the state is a terminal one, the episode is finished and a new one begins; otherwise a proper reward is given, on the basis of the satisfaction of those requirements presented in the previous subsection.

3. **Update the Q-Table**
   Once the reward is assigned, the Q-table is updated: the cell corresponding to the combination (previous location, selected action) is filled with the value coming from

the updating procedure described in subsubsection 5.5.1.

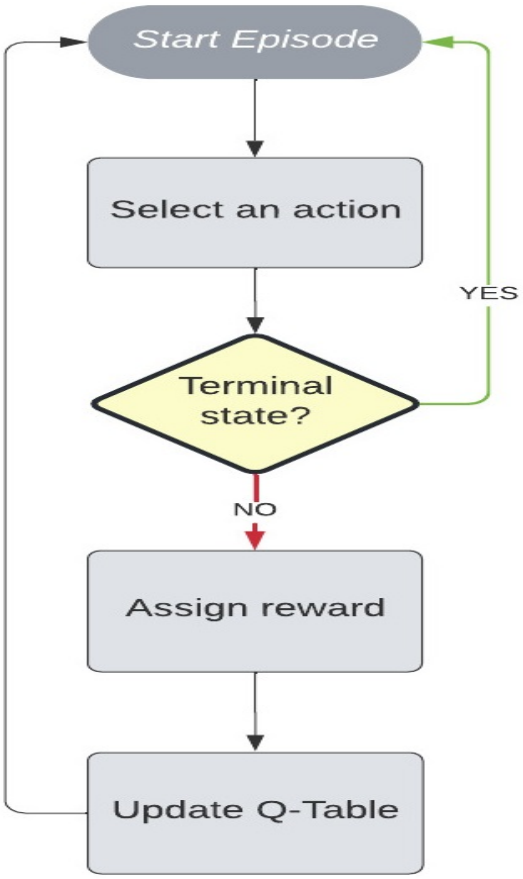


Figure 5.5: Steps of the algorithm

The previous steps are run for a sufficiently high number of episodes in order to train the system. The process as a whole takes the name of training process.

5.5.2. Application phase

At the end of this training process, the system is able to face the application phase, i.e. to take autonomous decisions in order to start from an initial state in the environment and move autonomously towards the target through a proper trajectory that satisfies the cryteria with which it has been trained.

6 | Control

This chapter presents the main features of controlling a space manipulator.
First, an introduction to the different controller types and control modes is presented.
Then, full control mode development is detailed with a technique called computed control torque.
Finally an underactuated control technique is implemented in order to assess the possibility of avoiding the control of the linear displacement of the base, fact that translates in a lower consumption of thrusters fuel.

6.1. Controller types

Trajectories are usually specified in the task space in terms of a desired path of the end-effector, while control actions are tipically performed in the joint space. The main goal of the joint space control is to design a feedback controller such that the actual state vector $\boldsymbol{q}(t)$ track the desired state $\boldsymbol{q}_d(t)$ as closely as possible.
Joint space controllers are generally divided into two groups:

- Classic control
- Model-based control

The classical control approach relies on classical techniques that can be implemented using a PID controller. Classical control has the advantage of being very simple to implement. However, this approach does not consider the effects of the system dynamics. In other words, in applications that concern robotic system like SMS which is a highly nonlinear dynamical system with multiple coupling variables, dynamic and coupling effects play such an important role that ignoring them in controller design will cause robot performance to deteriorate, especially if high precision is required and an accurate control is needed.

The Model-based control solves this problem by combining an inner feedback loop designed in order to eliminate non-linearities effects with a simple PID outer loop for classical control. Among the control approaches reported in the literature, typical methods include inverse dynamic control, the computed torque control, and the passivity based control

method.

Before entering into the details of the control logic implementation, it is necessary to consider the different possible operative modes characterising the control of space manipulator systems.

6.2. Control modes

When dealing with a space manipulator, no matter what the controller type is, there are different operative modes through which the system can fulfill its objective. Most researches have concentrated on mainly three modes [56], that differ for the number of state components that are involved in the control:

- **Arm control**
  The control in this mode involves only the arm of the space manipulator, while the spacecraft base is let free to float; from here the system takes the name of free floating system. Free-floating mode leaves the attitude uncontrolled during the operation of the robotic arm. However, leaving the attitude of the spacecraft tumbled is unsafe and not ideal for the power system and the sensors used for determining its attitude. The free-floating robot, although fuel efficient, presents some limitations that make other modes preferred when dealing with this kind of technology.
- **Full control**
  In this mode both the arm and the spacecraft base translation and attitude are controlled. A system that is controlled in this way takes the name of free-flying system. The free-flying mode couples the Attitude and Orbital Control System (AOCS) of the spacecraft with the robotic arm controller. This mode is preferred when the base attitude must be maintained during arm motion and with external disturbances, though it is obviously less efficient in terms of fuel consumption.
- **Partial base control**
  In this mode the arm is fully controlled, but the spacecraft base is controlled only in attitude, not in translation. Like the first mode, it involves the consideration of an underactuated system, i.e. a system whose number of actuators is smaller than the number of controllable states. Anyway, differently from the free-floating system, this third mode blends the advantages of lower fuel and power consumption with the ones of the free-flying systems, i.e. its capability of maintaining a desired attitude, that is an essential condition, especially in case of SMS.

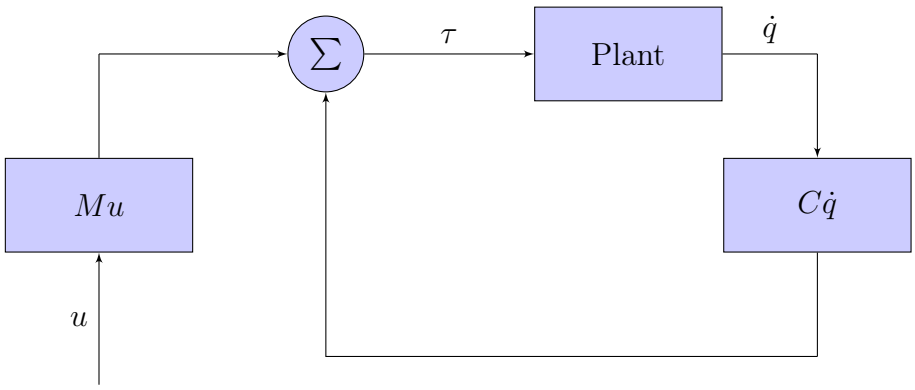
While for free-floating systems there's only one controller devoted to the control of the space manipulator arm, for the other two modes the existing control strategies can be classified into 2 types [19].
One type, called pose-fixed mode, consists in treating the base and the manipulator separately: firstly the position and attitude of the spacecraft bus are fixed using the thruster and reaction wheels to compensate any forces or torques resulted from the motions of the manipulator, and then applying the controllers designed for controlling the manipulators. The other mode is to control the motions of both the spacecraft base and the manipulator simultaneously by considering the dynamic coupling effects.
Although both approaches have been assessed theoretically, researches are moving towards the latter mode, in order to obtain a more compact and efficient design. This thesis goes in this direction.

As far as the control modes are concerned, in this work the full control and the partial base control have been considered. In both cases model based control are assessed, specifically the computed control torque is chosen for full actuation and a possible underactuated control, based upon the computed control torque idea is then presented.

6.3. Computed control torque

The computed control torque is one of the most common model based feedback linearization control techniques that have been studied and tested on non linear dynamical systems [4]. Its expression requires the complete knowledge of the system properties, in particular the knowledge of the actual Mass and Coriolis matrices $\boldsymbol{M}$ and $\boldsymbol{C}$. The forces/torques to apply to the system are given by the relation expressed in Equation 6.1

$$\boldsymbol{\tau} = \boldsymbol{M}\boldsymbol{u} + \boldsymbol{C}\dot{\boldsymbol{q}} \tag{6.1}$$



The control law consists of an inner nonlinear compensation loop represented by the term $\boldsymbol{C\dot{q}}$, and an outer loop with an equivalent control signal $\mathbf{u}$, as can be seen in the above diagram.

If the control actions in Equation 6.1 are considered, the equation of motion turns out to be the one in Equation 6.2:

$$\boldsymbol{M\ddot{q} + C\dot{q} = Mu + C\dot{q}} \tag{6.2}$$

By simplifying the previous expression, it follows that:

$$\boldsymbol{\ddot{q} = u} \tag{6.3}$$

Equation 6.3 represents a simple double integrator dynamic system. By following this law, the stability properties of the system are completely determined by the choice of the control action $\mathbf{u}$.

The outer-loop control $\mathbf{u}$ can be selected as a classical proportional–derivative (PD) feedback control law, whose expression can be given by Equation 6.4

$$\boldsymbol{u = \ddot{q}_d + K_D\dot{e} + K_Pe} \tag{6.4}$$

where $\mathbf{K}_D$ and $\mathbf{K}_P$ are proper gain matrices, derivative and proportional respectively, while $\mathbf{e}$ is the tracking error defined as

$$\boldsymbol{e = q_d - q} \tag{6.5}$$

By substituting Equation 6.4 in Equation 6.3, the resulting linear error dynamics turns out to be the following one

$$\ddot{\boldsymbol{e}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} = \boldsymbol{0} \tag{6.6}$$

With this equation, now the tracking performance of the control system depends upon ensuring an asymptotic convergence towards zero of the error dynamics that is described by the ordinary differential equations system of the second order (Equation 6.6).
According to linear system theory, convergence of the tracking error to zero is guaranteed by a proper selection of the gain matrices. A good choice for the selection of $\boldsymbol{K}_D$ and $\boldsymbol{K}_P$ is to take them as diagonal matrices: in this way it is guaranteed the implementation of a decoupled/independent joint control: each DOF, in this configuration, is controlled independently from the others.
There are many ways to get a proper expression for these matrices: from classical ziegler-nicholson scheme, to pole-placement and LQR techniques, to a trial and error process. Due to the scope of the thesis, that is mainly focused on obtaining a control system only capable of correctly tracking the desired trajectory, a trial and error process is sufficient. Typical value to start from can be found for example in [58].

Returning to the expression of the control action, by substituting Equation 6.4 in Equation 6.1, the commanded forces and torques on the system are given as follows

$$\boldsymbol{\tau} = \boldsymbol{M}(\ddot{\boldsymbol{q}}_d + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e}) + \boldsymbol{C}\dot{\boldsymbol{q}} \tag{6.7}$$

The computed control torque method is extremely attractive because it permits to work with a linearized system, hence designing the control with classical well studied techniques.

On the other side it requires to calculate step by step the dynamic matrices $\boldsymbol{M}$ and $\boldsymbol{C}$. This could be extremely challenging in real-time applications since their derivation is one of the main causes of computational time increase.
Moreover, the application of the computed control torque implies the deep knowledge of the parameters of the entire system in order to have the most possible accurate computation of $\boldsymbol{M}$ and $\boldsymbol{C}$. The perfect elimination of the non linearities can be done only in the computer simulation, where the matrices involved in the dynamics and in the control law are the same. In the practical case the computed matrices contain errors and uncertainties with respect to the real matrices. Indeed, this control method is affected by possible differences from the nominal condition, for example due to the presence of unmodeled dynamics (complex friction phenomena, transmission elasticity, orbital dynamics

consideration), etc.

With this section, the full control approach through computed control torque has been presented and analysed.
The following section, instead, starts introducing the underactuated control in order to assess a possible solution of the third control approach, i.e. partial base control.

6.4. Underactuated systems

The control of underactuated systems, i.e. systems with fewer actuators than degrees-of-freedom, is an open and interesting problem in controls; although there are a number of special cases where underactuated systems have been controlled, there are relatively few general principles, which make this field still widely unexplored.
When the underactuated approach is decided, the common solution is that of rearranging the equations governing the behavior of the system under consideration, in order to separate what is controlled from what is not.

Let's sart from the equation of motion of the space manipulator,

$$\mathbf{M}\,\ddot{\mathbf{q}} + \mathbf{C}\,\dot{\mathbf{q}} = \boldsymbol{\tau} \tag{6.8}$$

In order to account for the underactuation we have to consider a system with n degrees of freedom driven by m actuators, characterised by m<n.
Given that the system has more control signals than actuators, $\boldsymbol{\tau}$ has only m nonzero components as follows

$$\boldsymbol{\tau} = \begin{Bmatrix} \boldsymbol{F} \\ \mathbf{0}_{(n-m)x1} \end{Bmatrix} \tag{6.9}$$

Following the split in the control vector, the mathematical model of the system can be divided into two auxiliary dynamics, namely actuated and unactuated systems. Correspondingly, the states are split too, as follows

$$\boldsymbol{q} = \begin{Bmatrix} \boldsymbol{q}_a \\ \boldsymbol{q}_u \end{Bmatrix} \tag{6.10}$$

Equation 6.10 contains the actuated states, collected in the vector $\boldsymbol{q}_a$, and the unactuated states, collected in the vector $\boldsymbol{q}_u$:

$$\boldsymbol{q}_a = \begin{Bmatrix} \boldsymbol{q}_1 \\ \vdots \\ \boldsymbol{q}_m \end{Bmatrix}, \quad \boldsymbol{q}_u = \begin{Bmatrix} \boldsymbol{q}_{m+1} \\ \vdots \\ \boldsymbol{q}_n \end{Bmatrix} \tag{6.11}$$

Similarly, the mass matrix $\boldsymbol{M}$ and Coriolis matrix $\boldsymbol{C}$ are reorganized as in Equation 6.12 and in Equation 6.13

$$\mathbf{M}(\mathbf{q}) = \begin{bmatrix} \mathbf{M}_{aa}(\mathbf{q}) & \mathbf{M}_{au}(\mathbf{q}) \\ \mathbf{M}_{ua}(\mathbf{q}) & \mathbf{M}_{uu}(\mathbf{q}) \end{bmatrix} \tag{6.12}$$

$$\mathbf{C}(\mathbf{q}, \dot{\mathbf{q}}) = \begin{bmatrix} \mathbf{C}_{aa}(\mathbf{q}, \dot{\mathbf{q}}) & \mathbf{C}_{au}(\mathbf{q}, \dot{\mathbf{q}}) \\ \mathbf{C}_{ua}(\mathbf{q}, \dot{\mathbf{q}}) & \mathbf{C}_{uu}(\mathbf{q}, \dot{\mathbf{q}}) \end{bmatrix} \tag{6.13}$$

By splitting Equation 6.8 and substituting the prevoius equations, the resulting dynamic is expressed with the following system

$$\begin{cases} \mathbf{M}_{aa}\ddot{\mathbf{q}}_a + \mathbf{M}_{au}\ddot{\mathbf{q}}_u + \mathbf{C}_{aa}\dot{\mathbf{q}}_a + \mathbf{C}_{au}\dot{\mathbf{q}}_u = \mathbf{F} \\ \mathbf{M}_{ua}\ddot{\mathbf{q}}_a + \mathbf{M}_{uu}\ddot{\mathbf{q}}_u + \mathbf{C}_{ua}\dot{\mathbf{q}}_a + \mathbf{C}_{uu}\dot{\mathbf{q}}_u = \mathbf{0} \end{cases} \tag{6.14}$$

It must be noted that the mass matrix $\boldsymbol{M}$ is always guaranteed to be symmetric and positive definite, thus invertible; the same feature is guaranteed for the matrices $\mathbf{M}_{aa}$ and $\mathbf{M}_{uu}$, according to Sylvester's criterion [57].
Due to this, the unactuated states $\boldsymbol{q}_u$ can be determined as follows

$$\ddot{\mathbf{q}}_u = -\mathbf{M}_{uu}^{-1}\left\{\mathbf{M}_{ua}\ddot{\mathbf{q}}_a + \mathbf{C}_{ua}\dot{\mathbf{q}}_a + \mathbf{C}_{uu}\dot{\mathbf{q}}_u\right\} \tag{6.15}$$

Substituting Equation 6.15 into the first of Equation 6.14, it leads to

$$\overline{\mathbf{M}}\ddot{\mathbf{q}}_a + \overline{\mathbf{C}}_1\dot{\mathbf{q}}_a + \overline{\mathbf{C}}_2\dot{\mathbf{q}}_u = \mathbf{F} \tag{6.16}$$

where

$$\begin{aligned} \overline{\mathbf{M}} &= \mathbf{M}_{aa} - \mathbf{M}_{au}\mathbf{M}_{uu}^{-1}\mathbf{M}_{ua} \\ \overline{\mathbf{C}}_1 &= \mathbf{C}_{aa} - \mathbf{M}_{au}\mathbf{M}_{uu}^{-1}\mathbf{C}_{ua} \\ \overline{\mathbf{C}}_2 &= \mathbf{C}_{au} - \mathbf{M}_{au}\mathbf{M}_{uu}^{-1}\mathbf{C}_{uu} \end{aligned} \tag{6.17}$$

Equation 6.16 can be seen as the new equation of motion representing the dynamics of the actuated states of the system. It has a similar shape as Equation 6.8, but with the introduction of an additional term that accounts for the coupling influence of the underactuated states over the actuated ones.

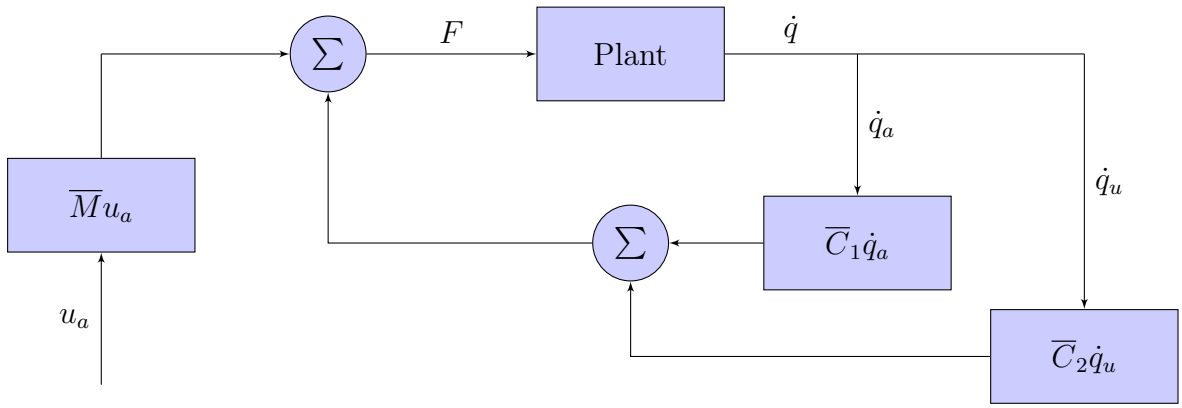
6.5. Designed underactuated control

For fully actuated systems, the trajectory tracking problem is now reasonably well understood [55]. Instead, for underactuated systems trajectory tracking is still an active research topic.
Fully actuated systems are dramatically easier to control than underactuated systems. The key observation is that, for fully-actuated systems with known dynamics it is possible to use feedback to effectively change a nonlinear control problem into a linear control problem. The field of linear control is incredibly advanced, and there are many well-known solutions for controlling linear systems.
The underactuated systems are not feedback linearizable. Therefore, unlike fully actuated systems, the control designer has no choice but to deal with the nonlinear dynamics of the plant in the control design. This dramatically complicates feedback controller design.

Anyway, Equation 6.16 allows to make some similar considerations to what has been done in section 6.3 with the computed control torque: as for fully actuated systems, the idea adopted in this thesis is that of controlling the system with an equivalent linearization form. The goal is then to reach a condition like Equation 6.3 in which the non linearities in Equation 6.16 are canceled out and all is reduced to a simple double integrator dynamical system. The aim is then the determination of a control logic that brings the actuated states to their desired values, without controlling the underactuated ones.

In analogy with Equation 6.1, the non zero part of the control action is selected as follows

$$\boldsymbol{F} = \overline{\boldsymbol{M}}\boldsymbol{u} + \overline{\boldsymbol{C}}_1\dot{\boldsymbol{q}}_a + \overline{\boldsymbol{C}}_2\dot{\boldsymbol{q}}_u \tag{6.18}$$



If Equation 6.18 is substituted in Equation 6.16, after a simplification the following condition for the actuated states is reached

$$\ddot{\boldsymbol{q}}_{\boldsymbol{a}} = \boldsymbol{u}_a \tag{6.19}$$

This is once again a double integrator dynamical system governing in this case only the behavior of the actuated states.
The equivalent control input $\boldsymbol{u}_a$ is taken as a PD controller

$$\boldsymbol{u}_a = \ddot{\boldsymbol{q}}_{\boldsymbol{ad}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} \tag{6.20}$$

where $\boldsymbol{e} = \boldsymbol{q}_a - \boldsymbol{q}_{ad}$ is the error over the actuated states. The dynamics of this error is given as follows

$$\ddot{\boldsymbol{e}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e} = \boldsymbol{0} \tag{6.21}$$

and is guaranteed to converge to zero if the gain matrices are properly selected.
Once again the selection of these matrices can be carried out through a trial and error process and they can be taken as diagonal matrices in order to set an independent joint control.

Finally, substituting Equation 6.20 in Equation 6.18,the control signals $\boldsymbol{F}$ become

$$\mathbf{F} = \overline{\mathbf{M}}(\ddot{\boldsymbol{q}}_{\boldsymbol{ad}} + \boldsymbol{K}_D\dot{\boldsymbol{e}} + \boldsymbol{K}_P\boldsymbol{e}) + \overline{\mathbf{C}}_1\dot{\mathbf{q}}_a + \overline{\mathbf{C}}_2\dot{\mathbf{q}}_u \tag{6.22}$$

The same consideration done for the full actuation is still valid for the proposed solution: this approach is affected by dynamic modeling errors, by unmodeled dynamics and it requires a step by step computation of the dynamical matrices.

Note that the unactuated states have not been considered in the control law, so they are not guaranteed to reach the desired condition. Nowadays researchers are trying and find a solution to account for these states too with a control law that does not produce an action to control them but exploits the reciprocal influence between actuated and unuctuated states to make both of them approximately converge towards their prescribed target solution.

This approach has now to be implemented in the space manipulator system under consideration.

6.6. Underactuated space manipulator

Due to the fact that we are not controlling the traslation of the base, the second operative mode presented in section 6.2 leads to an underactuated system.
In this case, following the structure shown in previous section, the state vector can be split and reorganized as in Equation 6.23, considering that the only actuators are the attitude control system (ACS) and the joint motors, responsible for the control of the base orientation and joint angles respectively.

$$\boldsymbol{q}_a = \begin{Bmatrix} \boldsymbol{\alpha} \\ \boldsymbol{\theta} \end{Bmatrix}, \qquad \boldsymbol{q}_u = \boldsymbol{r}_{s/c} \tag{6.23}$$

Consequently, the mass and coriolis matrices have to be rearranged and written as follows

$$\begin{aligned} \mathbf{M}(\mathbf{q}) &= \begin{bmatrix} \mathbf{M}_\alpha(\mathbf{q}) & \mathbf{M}_{\alpha\theta}(\mathbf{q}) & \mathbf{M}_{\alpha r}(\mathbf{q}) \\ \mathbf{M}_{\theta\alpha}(\mathbf{q}) & \mathbf{M}_\theta(\mathbf{q}) & \mathbf{M}_{\theta r}(\mathbf{q}) \\ \mathbf{M}_{r\alpha}(\mathbf{q}) & \mathbf{M}_{r\theta}(\mathbf{q}) & \mathbf{M}_r(\mathbf{q}) \end{bmatrix} \\ \mathbf{C}(\mathbf{q}) &= \begin{bmatrix} \mathbf{C}_\alpha(\mathbf{q}) & \mathbf{C}_{\alpha\theta}(\mathbf{q}) & \mathbf{C}_{\alpha r}(\mathbf{q}) \\ \mathbf{C}_{\theta\alpha}(\mathbf{q}) & \mathbf{C}_\theta(\mathbf{q}) & \mathbf{C}_{\theta r}(\mathbf{q}) \\ \mathbf{C}_{r\alpha}(\mathbf{q}) & \mathbf{C}_{r\theta}(\mathbf{q}) & \mathbf{C}_r(\mathbf{q}) \end{bmatrix} \end{aligned} \tag{6.24}$$

From previous expression,

$$\begin{aligned}
\mathbf{M}_{aa}(\mathbf{q}) &= \begin{bmatrix} \mathbf{M}_{\alpha}(\mathbf{q}) & \mathbf{M}_{\alpha\theta}(\mathbf{q}) \\ \mathbf{M}_{\theta\alpha}(\mathbf{q}) & \mathbf{M}_{\theta}(\mathbf{q}) \end{bmatrix} \\
\mathbf{M}_{uu}(\mathbf{q}) &= \mathbf{M}_{r}(\mathbf{q}) \\
\mathbf{M}_{au}(\mathbf{q}) &= \begin{bmatrix} \mathbf{M}_{\alpha r}(\mathbf{q}) \\ \mathbf{M}_{\theta r}(\mathbf{q}) \end{bmatrix}
\end{aligned} \tag{6.25}$$

Through Equation 6.25 it is possible to calculate matrices $\overline{\mathbf{M}}$, $\overline{\mathbf{C}}_1$, $\overline{\mathbf{C}}_2$ like in Equation 6.17 and consequently the control actions as in Equation 6.22.
These actions are usually given by coupling the joint motors with reaction/inertia wheels or control moment gyroscopes: these control moment devices are the most efficient solution when trajectory tracking is addressed.

7 | Simulation results

In this chapter are reported the results concerning both the simulation of the system and the machine learning algorithm implemented. First, the characteristics of the chosen space manipulator system (concerning masses, inertias, Denavit-Hartenberg parameters) are introduced. Then, the softwares used for the accomplishment of the task are presented. After that, the results related to algorithm analysis and trajectory generation are shown. Finally the selected control approach is tested.

7.1. Space manipulator features

The space manipulator base is selected to be a $1m$ cube satellite, on which a $3m$ long robotic arm with 6 DOF is mounted.
The selected inertia properties are reported in Table 7.2 for both base and manipulator.

	Mass [kg]	**I_x[kg m^2]**	**I_y[kg m^2]**	**I_z[kg m^2]**
Base	1000	166.67	166.67	166.67
Link 1	20	0.2	1.87	1.87
Link 1	15	0.1	1.25	1.25
Link 3	15	0.1	1.25	1.25
Link 4	15	0.1	1.25	1.25
Link 5	15	0.1	1.25	1.25
Link 6	20	0.2	1.87	1.87

Table 7.1: Satellite and manipulator dynamic properties

The parameters of the satellite-mounted manipulator are taken as reported in the following Table 7.2.

	a [m]	$\boldsymbol{\alpha}[\boldsymbol{deg}]$	**d [m]**
Link 1	0.7	$\pi/2$	0
Link 2	0.4	0	0
Link 3	0.4	$\pi/2$	0
Link 4	0.4	$\pi/2$	0
Link 5	0.4	0	0
Link 6	0.7	0	0

Table 7.2: Satellite and manipulator dynamic properties

The position of the first joint in the body frame is taken as

$$\boldsymbol{r}_q = \begin{Bmatrix} 0.5 \\ 0 \\ 0 \end{Bmatrix} m \tag{7.1}$$

7.2. Simulation environment

As already stated in chapter 2, the simulation of the system is carried out by implementing and relating each single dynamical block. In this thesis, the chosen environment for the simulation has been MATLAB and Simulink, as can be seen in Figure 7.1 where the block scheme is presented. Simulink environment is very useful when the goal is the simulation of nonlinear dynamical systems, since it is very easy to read as it is like a block diagram and since the simulation is done over time, so state of the whole system in each time-slot is available.

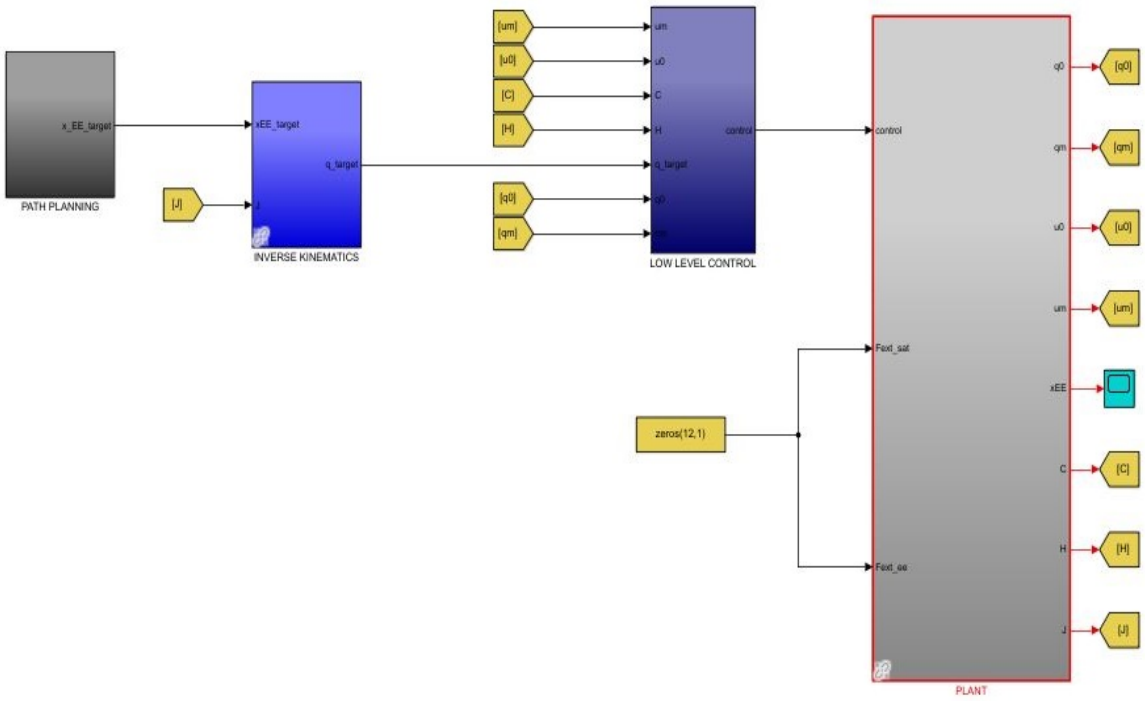


Figure 7.1: Block scheme implemented in Simulink. Note that the block named "PLANT" contains the dynamics and direct kinematics blocks.

The path planning algorithm has instead been implemented with python language in Microsoft Visual Studio Code environment. Python is a very simple programming language, which makes it appropriate for machine learning. Moreover, the Python programming language best fits machine learning due to its ability to run on multiple platforms, such as Windows, Linux and macOS, and due to its popularity in the programming community. Furthermore, Python frameworks and libraries offer a reliable environment that reduces software development time significantly.

The cooperation between MATLAB and Python codes is then essential for the accomplishment of phase 2 of the algorithm (see chapter 5). Indeed, in order to receive the rewards corresponding to the current location, a connection between Visual Studio Code and MATLAB is established through the *matlab.engine* library. In this way a simultaneous running of the python learning algorithm and of MATLAB is performed: MATLAB receives the current location of the end effector and gives back the correspondent rewards, by checking that the system dynamical features fulfill the requirements in subsubsection 5.5.1.

7.3. Q-learning training

In this section the results coming from the Q-learning training method are presented. The goal is to make an analysis of the python algorithm in order to assess the effectiveness of the method and its computational burden.

7.3.1. Algorithm analysis

In order to verify the effectiveness of the algorithm, a common way is to visualize the trend of the cumulated sum of rewards, i.e. of the sum of the rewards obtained per each episode; this trend shall be increasing with the increasing number of episode, if the algorithm is correctly implemented.
For the selected α and γ values the cumulated sum of rewards is reported in the following Figure 7.2.

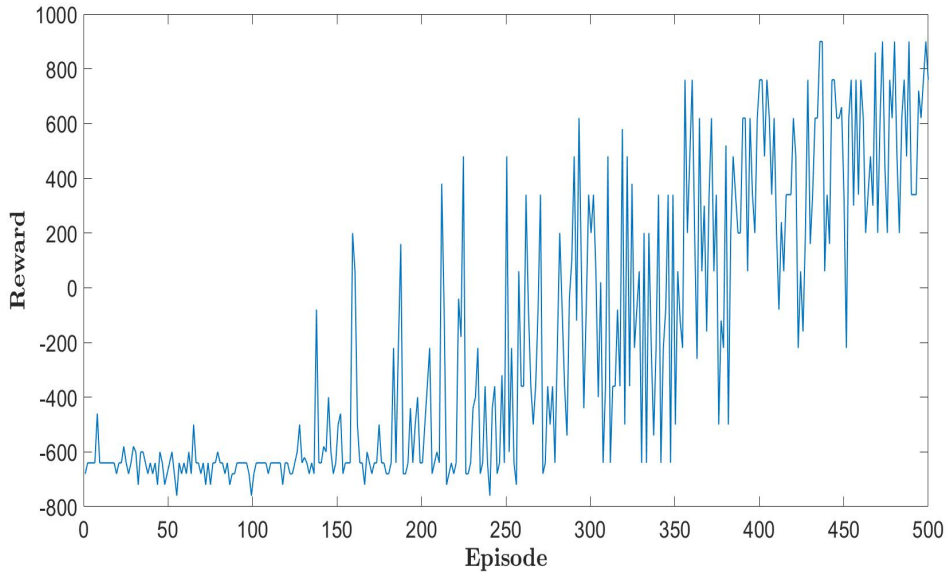


Figure 7.2: Cumulated sum of rewards with $\alpha = 0.9$ and $\gamma = 0.9$

We can see that for the first episodes the rewards are low, since the agent is still exploring the environment. From episode 150 the interaction with the environment starts producing higher rewards for the agent, up to when in the end, the trained agent keeps on performing actions that guarantee the highest possible reward, thanks also the the exploitation fetaure that is guaranteed by the increasing value of ϵ parameter.

It is interesting to compare the previous trend with the one that would be obtained with different values of α and γ. The comparison is depicted in Figure 7.3

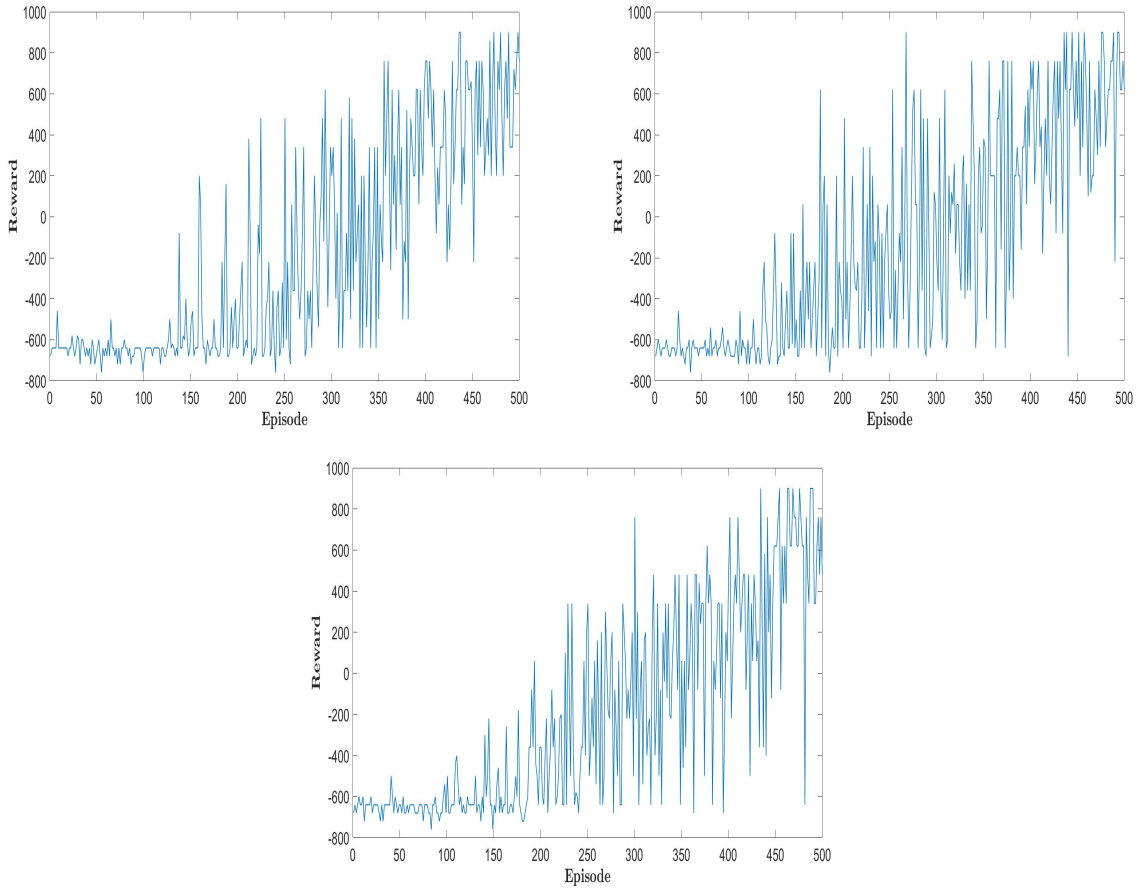


Figure 7.3: Comparison between cumulated sum of rewards trends for different values of α and γ: on the top-left $\alpha = 0.9$ and $\gamma = 0.9$; on the top-right $\alpha = 0.5$ and $\gamma = 0.8$; on the bottom $\alpha = 0.2$ and $\gamma = 0.5$

It can be seen that for $\alpha = 0.5$ and $\gamma = 0.8$ the agent keeps on taking wrong actions even in the final episodes, due to the lower learning rate that delays learning phase. As a result the algorithm is not working for all the cells in the environment, meaning that starting from some locations in the environment the agent is not abe to derive its path.
Even worse results are obtained for $\alpha = 0.2$ and $\gamma = 0.5$ case, where the first positive rewards are obtained after a larger number of episodes and even in the final phase the agent is still not well trained. This translates into a higher decrease in the coverage percentage.

Here below, Table 7.3, are reported the percentage of "working" cells for the same number of episodes, but for different parameters values.

	Working cells [%]
$\alpha = 0.9, \gamma = 0.9$	100
$\alpha = 0.5, \gamma = 0.8$	95
$\alpha = 0.2, \gamma = 0.5$	87

Table 7.3: Algorithm effectiveness for different α and γ

As reported, the $\alpha = 0.9, \gamma = 0.9$ is the only case guaranteeing the total coverage. In order to obtain the same result also for the other cases the number of episodes has to be increased, thus increasing the training time.

7.3.2. Computational time

An analysis on the computational demand of the training process has been conducted. Obviously, this shows an increase with the increase in the dimensions of the environment, as shown in Figure 7.4.

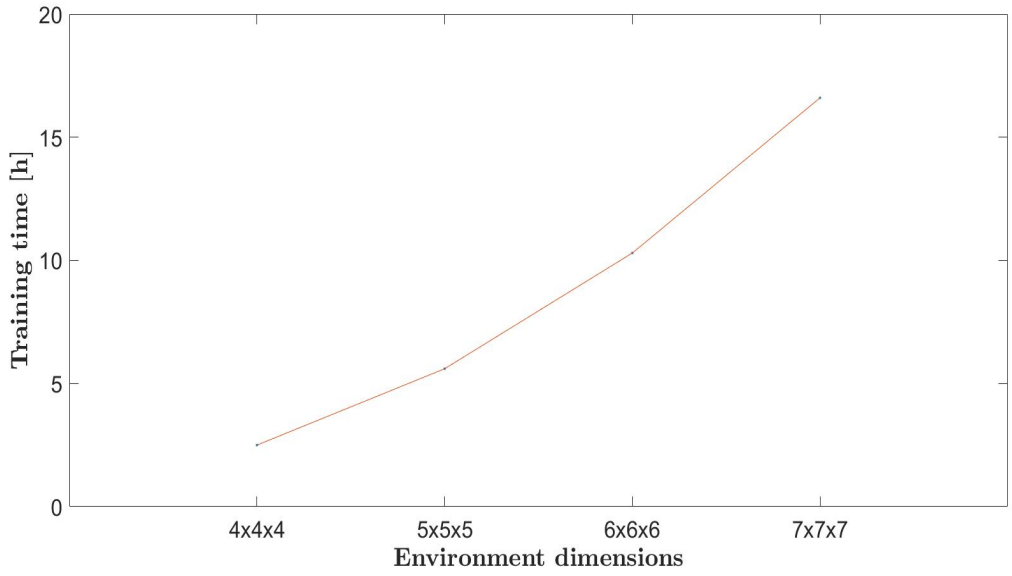


Figure 7.4: Computational time depending on environmental dimensions

The high cost in terms of time required for the training process is mainly due to two reasons: the first one is related to the intrinsic feature of reinforcement learning methods, that, as already stated in chapter 5, require a high amount of time to efficiently train the agent, since it needs a lot of interactions with the environment in order to learn the lesson. The second reason is imputable to the time required to the Matlab process needed to assign a reward for each action taken by the agent.
At the end of the day, since this high computational demand is limited to the training phase only, it does not represent an issue. Indeed, once the system is trained, the applicative phase can be carried out in almost real time.

Next section is hence dedicated to the applicative phase.

7.4. Applicative phase

Once the training algorithm has been analysed under the computational point of view, it is then possible to check the subsequent applicative phase.
Assuming that the target position is known and arbitrarily equal to $p_t = [1.5, 0.7, -1]m$ and the starting position is $p_{ee} = [1.4, 0.9, -0.8]m$, the system is able to derive autonomously a proper end effector trajectory from start to target. This trajectory is reported in Figure 7.5 in 3D space and and in Figure 7.6 over time.

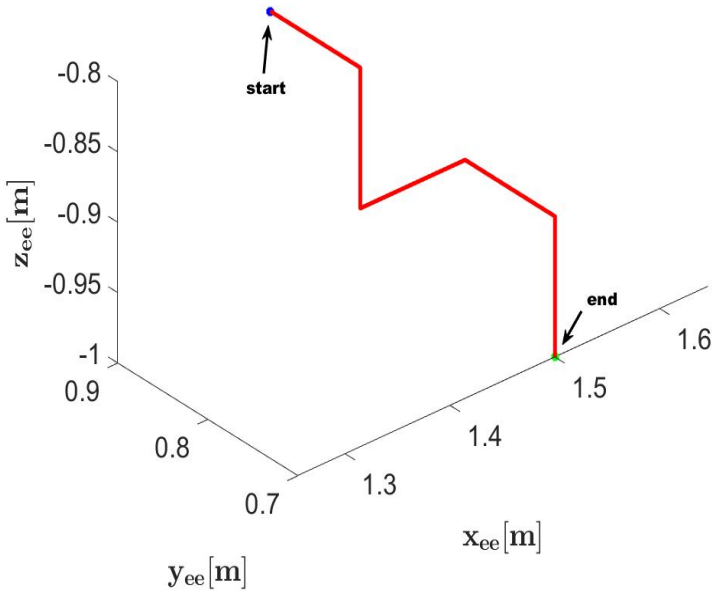


Figure 7.5: End effector trajectory in 3D space: the blue point is the starting position, the green point is the target

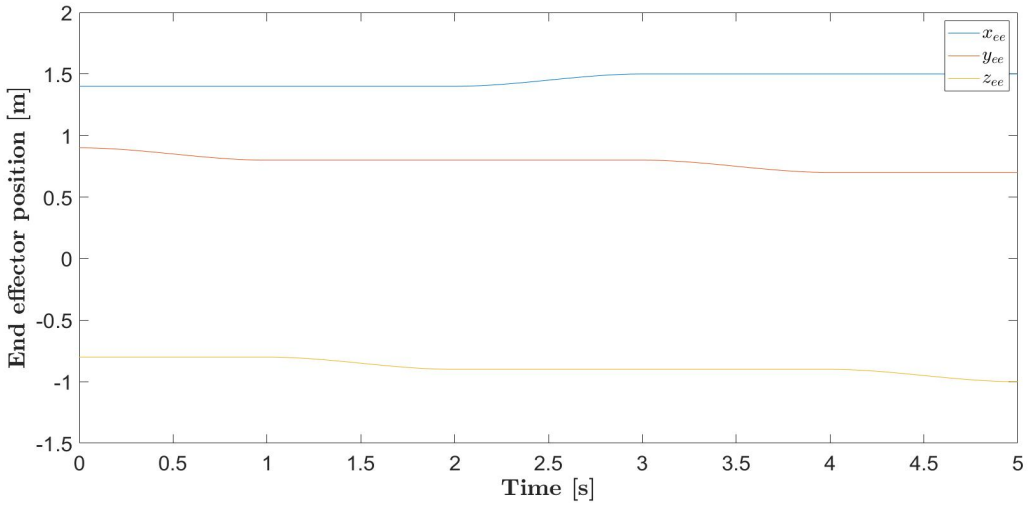


Figure 7.6: End effector trajectory over time

In order to verify the feasibility of the trajectory followed by the agent, the physical quantities of interest are reported in the following figures and analysed.

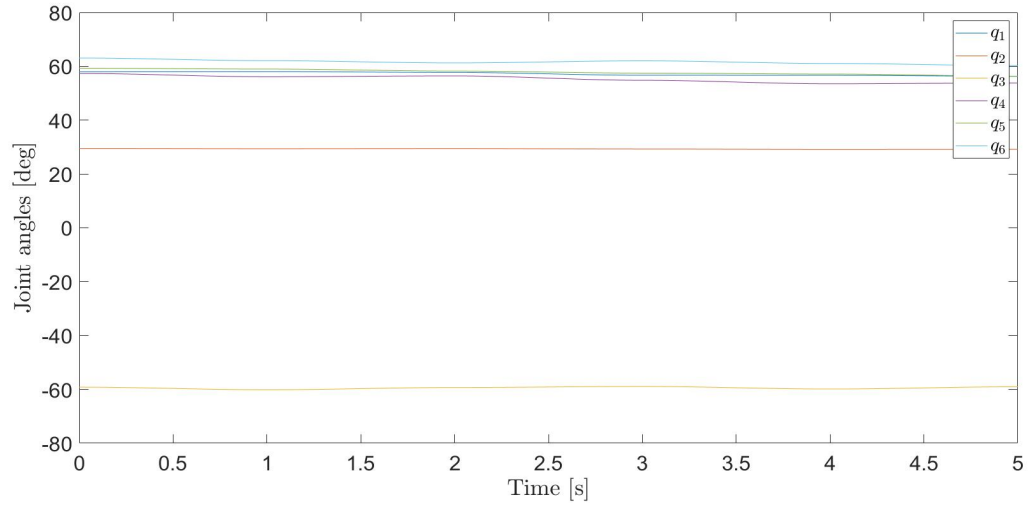


Figure 7.7: Joint angles

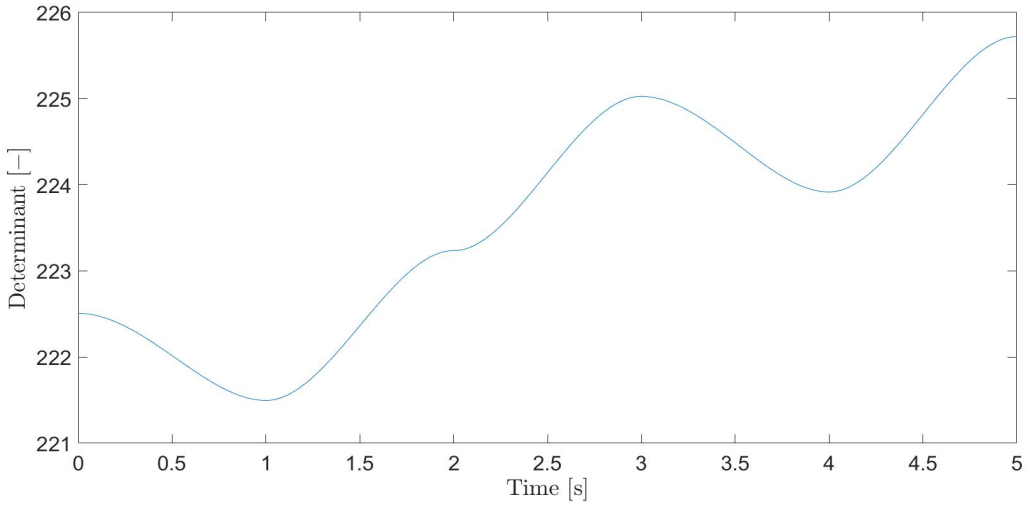


Figure 7.8: Singularity condition

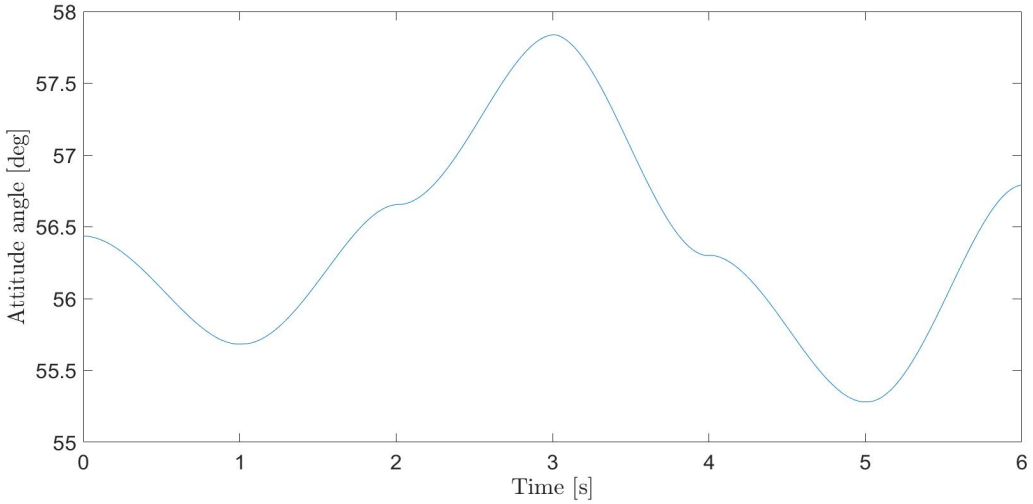


Figure 7.9: Angle with respect to the target

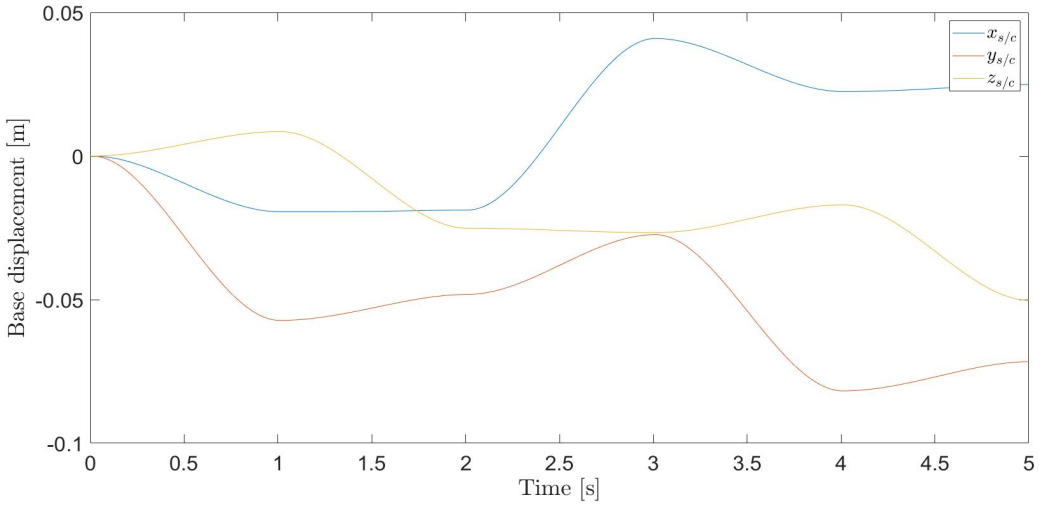


Figure 7.10: Base displacement

As can be seen from previous figure, all the constraints imposed in the algorithm are satisfied, in particular those regarding the terminal states.
From the joint behavior it can be seen that joint angles are always inside the prescribed mechanical limits stated in chapter 5
From the second plot the condition for singularity avoidance is fulfilled since $det(JJ^T)$ is always different from 0.
From third plot the attitude angle between the base and the target is always inside the prescribed range.
From last plot, the base displacement is overall lower than half the total displacement of the end effector.

Different initial condition

Starting from a different initial position $p_{ee} = [1.2, 0.9, -1.1]m$, the algorithm is still capable of determining a trajectory autonomously, as can be seen from Figure 7.11 and Figure 7.12.

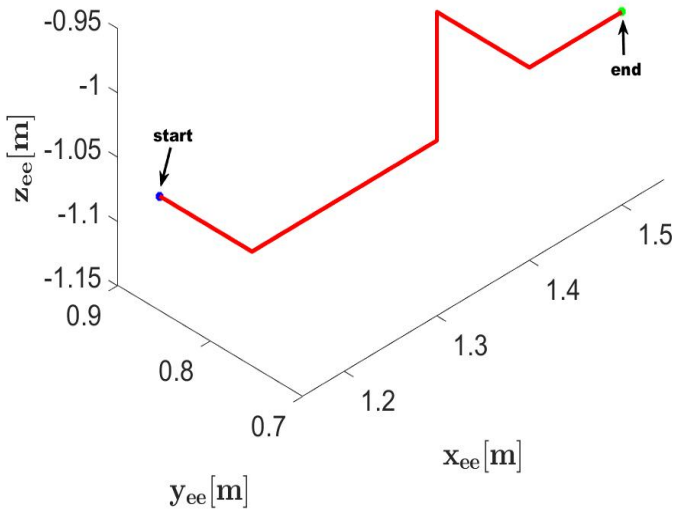


Figure 7.11: End effector trajectory in 3D space: the blue point is the starting position, the green point is the target

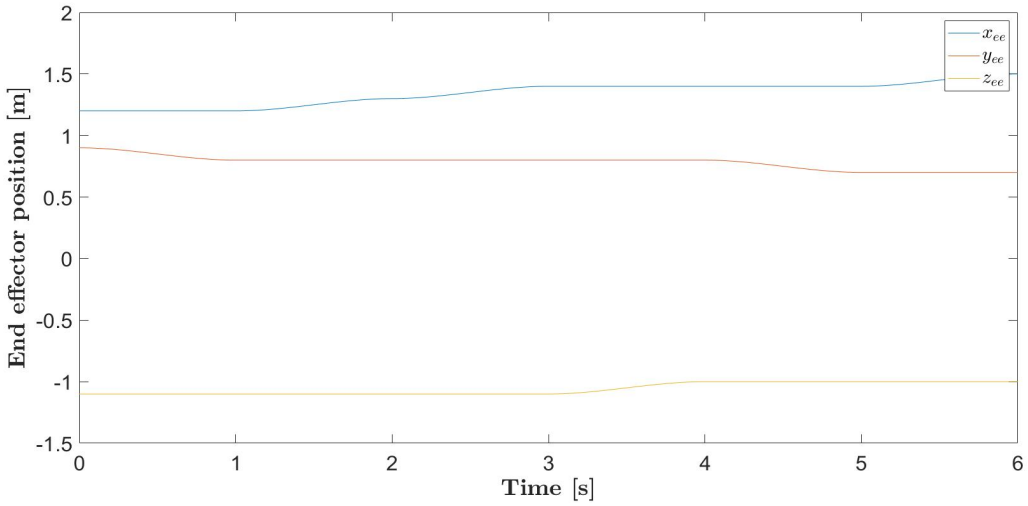


Figure 7.12: End effector trajectory over time

Together with the trajectory, the quantities of interest are reported in Figure 7.13. Once again, the satisfaction of the cryteria exposed in chapter 5 is fulfilled.

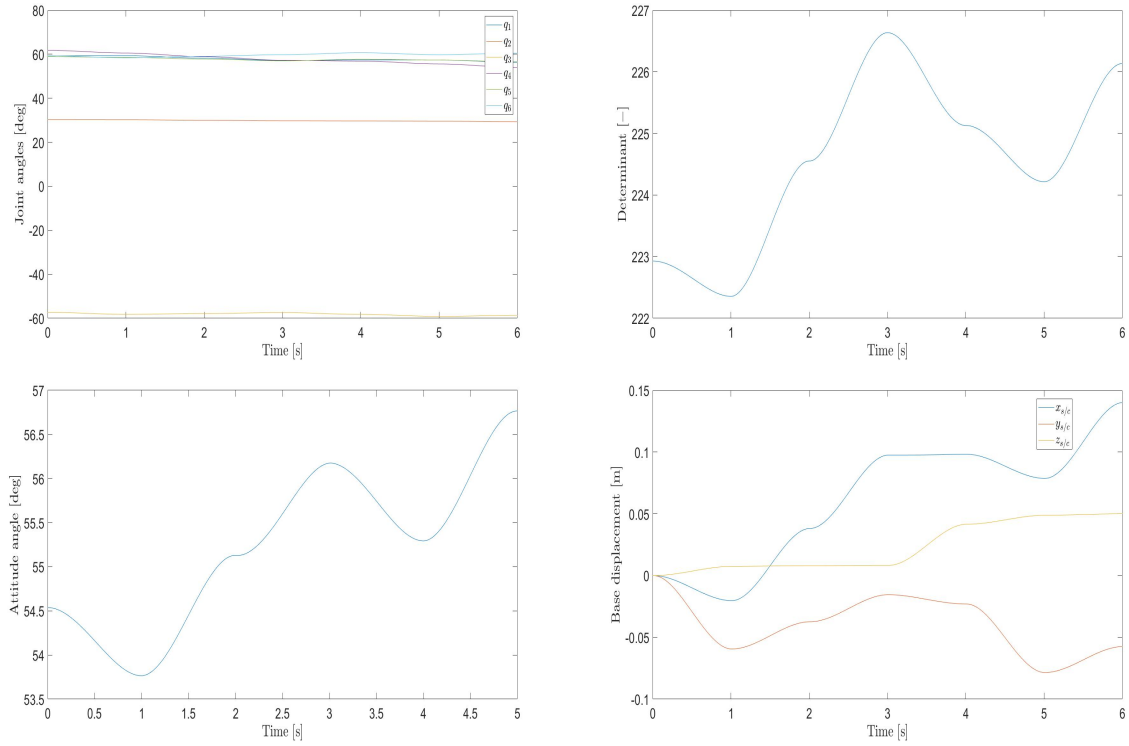


Figure 7.13: Main physical quantities: on the upper left joint angles, on the upper right the singularity condition, on the bottom left the angle with respect to the target, on the bottom right the base displacement

7.5. Control

In this section, the designed control approaches results are shown, both for the full actuation and for the underactuation, in order to test their trajectory tracking effectiveness.

7.5.1. Full control

For the full control, the computed control torque presented in chapter 6 has been tested. First of all the gain matrices ruling the asymptotic convergence of the tracking error are taken after a trial and error process as diagonal matrices with diagonal values equal to

$$\boldsymbol{k}_{Pi} = 1 \tag{7.2}$$

$$\boldsymbol{k}_{Di} = 10 \tag{7.3}$$

The test of the controller is here conducted on the first trajectory in section 7.4. The full control result regarding the end effector position is reported in the following Figure 7.14 and Figure 7.15

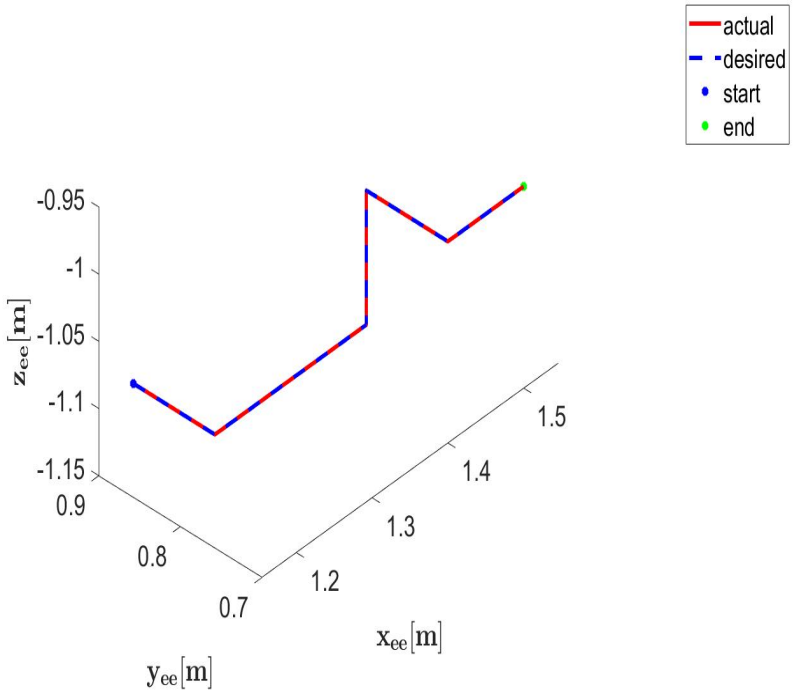


Figure 7.14: Computed control torque of end effector 3D trajectory

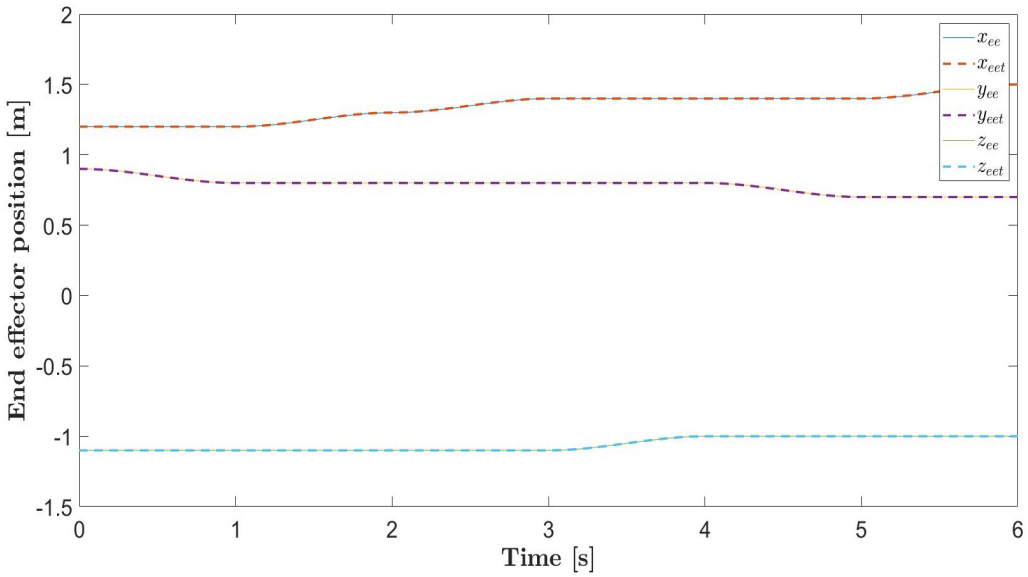


Figure 7.15: Computed control torque of end effector in time

As can be seen, the implemented control strategy allows for an efficient trajectory tracking for each of the three components of the end effector position. Indeed, the maximum tracking error concerning the end effector trajectory is in the order of $10^{-4}\,m$ as shown in Figure 7.16

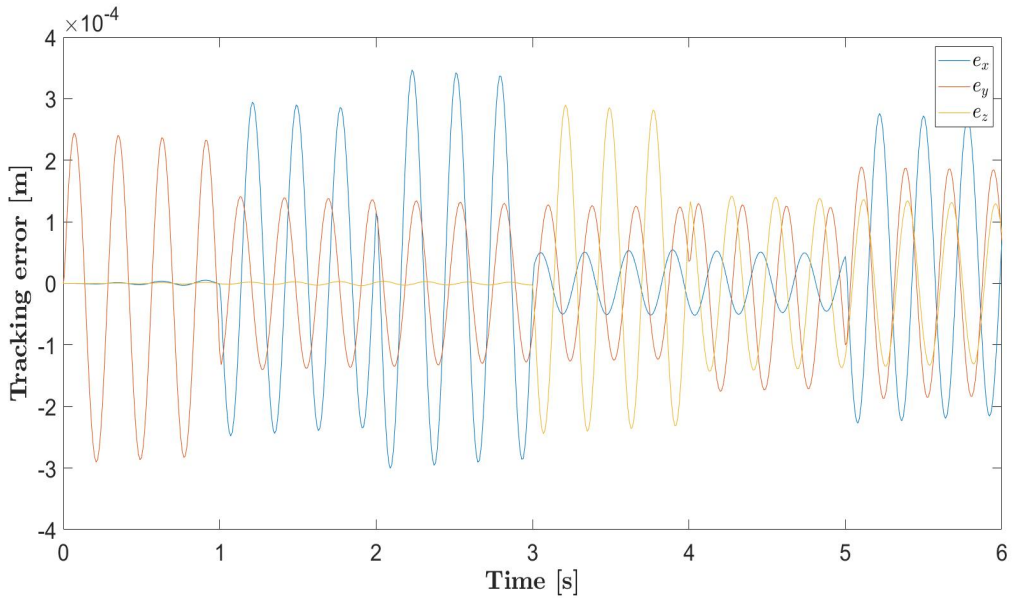


Figure 7.16: Tracking error

7.5.2. Underactuated control

In this subsection the designed underactuated control system is tested.

Once again, first of all the gain matrices are taken, after a trial and erro process, as diagonal matrices with diagonal values equal to

$$\boldsymbol{k}_{Pi} = 10 \tag{7.4}$$

$$\boldsymbol{k}_{Di} = 100 \tag{7.5}$$

Then, the test is conducted on the same trajectory as for the full actuation.
As can be seen from Figure 7.17 Figure 7.18 the control is just partially following the desired end effector trajectory.

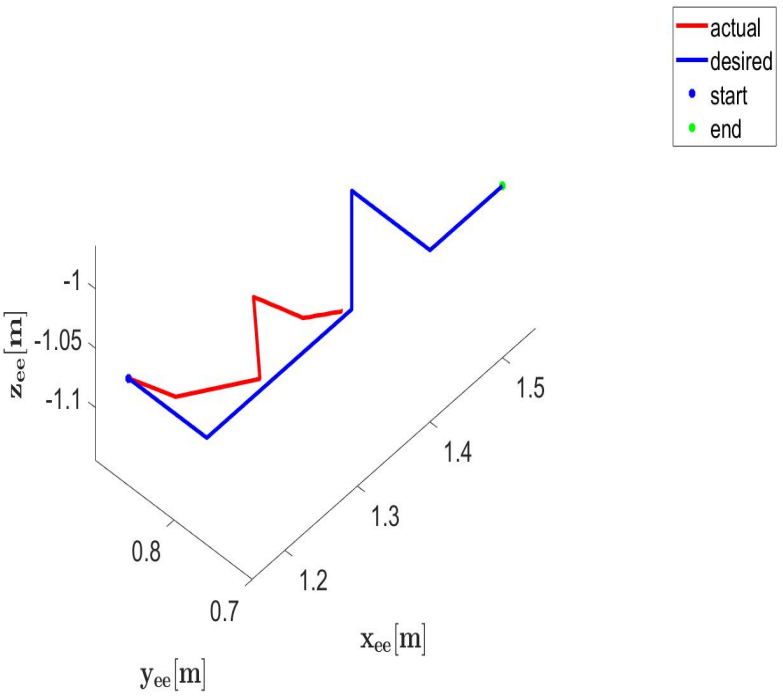


Figure 7.17: Underactuated control for the 3D end effector position

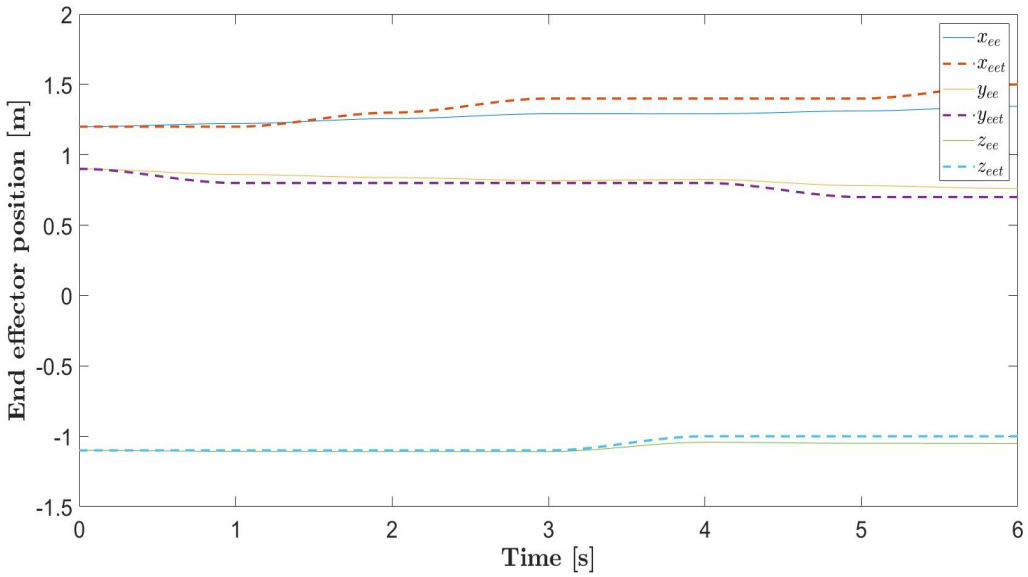


Figure 7.18: Underactuated control for the end effector position in time

This behavior is due to the fact that the system is not controlling all the states.

Anyway, it is interesting to analyse how the implemented underactuated control system accomplishes its task of controlling the actuated states, as demonstrated by Figure 7.19, where the desired joint behavior is tracked efficiently.

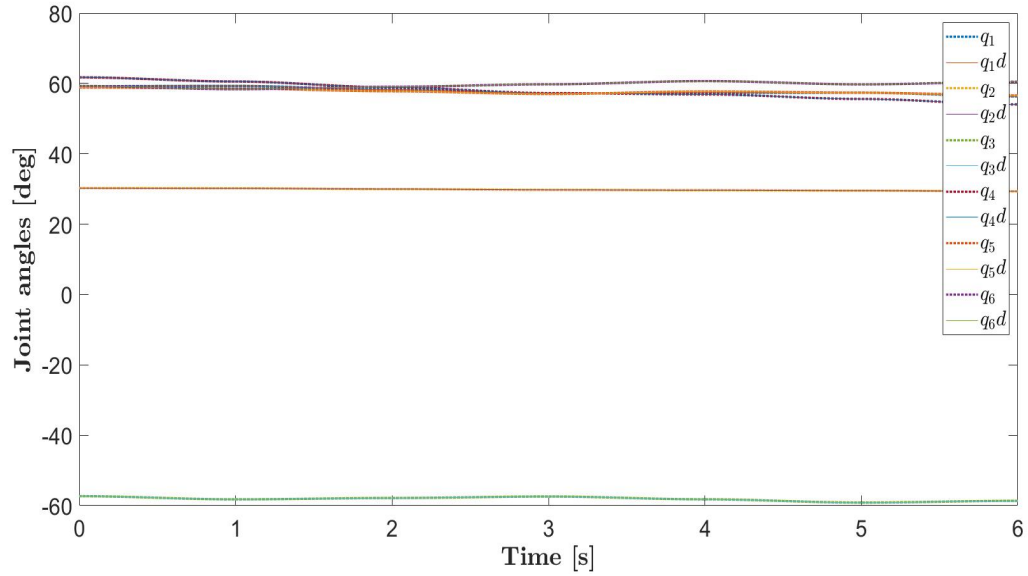


Figure 7.19: Underactuated control of the joints

The tracking error is indeed very low and in the order of $10^{-3}\,deg$, as reported in the following Figure 7.20

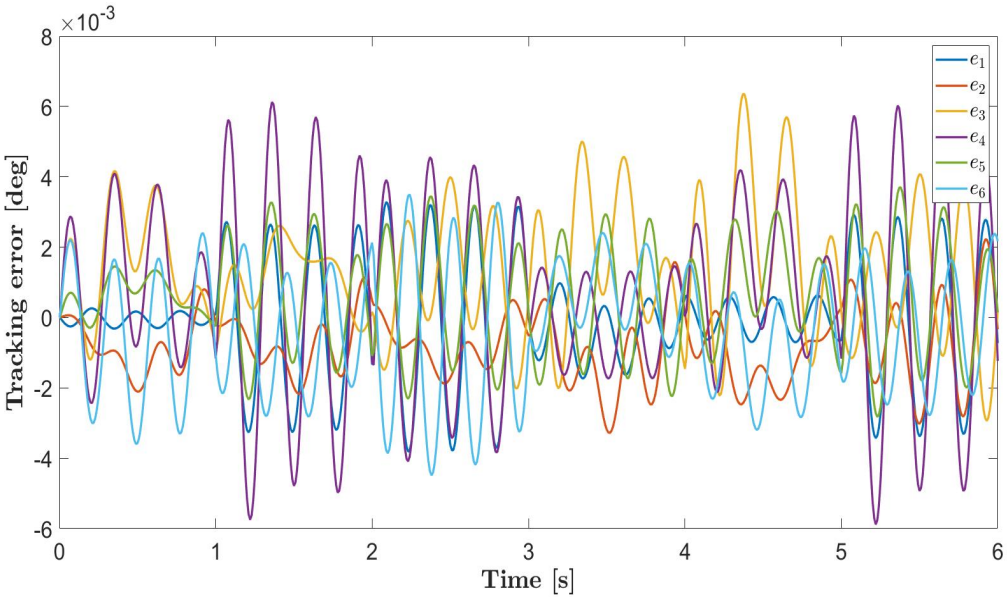


Figure 7.20: Tracking error

At the end of the day, both controllers show very good tracking capabilities for the actuated states, fact that make these logics valid candidates for SMS control as far as the controlled states are concerned.

8 | Conclusions

Space manipulator systems are expected to be a key technology in many future space missions, thanks to their versatility that make them the perfect solution for on-orbit servicing tasks, like refuelling and repairing already existent spacecrafts or removing debris. Anyway, this technology is very complex to be modelled and its application is still a highly challenging field.
In this work a global analysis of the 3D behavior of a SMS is presented, from the kinematics and dynamics study to the path planning algorithm and control implementations. This was carried out dividing the SMS functionalities into blocks and hence by treating each block separately.

A new solution for path planning of a space manipulator is presented. This solution is based upon a machine learning technique called Q-learning that offers great advantages especially as it allows to have a system with a certain autonomy in decision making. The trajectory generated by the algorithm is then ensuring some critical aspects of typical SMS mission like singularity-free feature, joint and attitude limit avoidance. Moreover, the algorithm is set up in order to promote the motion of the SMS that allow for a small displacement of the base of the spacecraft.
The results aim at analysing the algorithm assessing both its implementation correctness and the system's response. Regarding the latter, the method shows a good robustness to different initial positions of the SMS end effector in the environment; furthermore, the constraints previously considered are fulfilled.

Finally a control system based on model-based techniques is implemented. In particular, the computed control torque technique has been implemented for full actuation, due to its simplicity of implementation and tracking guarantee. Anyway this control method shows drawbacks that make it a very good solution only in the ideal case of perfect mass and coriolis matrix computation.
Eventually, an underactuated control is also proposed for fuel consumption purposes. This method is based on the same idea of the computed control torque, with the difference that lies in the fact that now the only actuated states are tracked.

Both controllers show a good tracking capability regarding the states under control. Obviously, the fact that the underactuated system is not controlling all the states overall translates into a non efficient tracking of the end effector trajectory.

8.1. Future works

There are some points in which the analysis presented in this thesis can be further developed and they are mainly:

1. Multiple manipulators consideration
2. Dynamical modelling assumptions
3. Sensors and actuators integration
4. Underactuated control improvement

The first possibility consists in a further development of the kinematics and dynamics modeling algorithms in order to study multiple manipulators systems, i.e. spacecraft on which more than robotic arms are mounted.

The second development direction concerns the integration in the dynamics blocks of those features that have been currently neglected. For example, the implementation of an Orbital Mechanics module that would take into consideration the orbit motion of the chaser.

The third developement is related to the introduction in the model of proper sensors and actuators that would make the analysis more realistic, by taking into consideration both the errors in the measurements and the saturation limits of the actuators dedicated to the control.

The fourth developement considers the study and development of an improved underactuated system that would lead to the correct tracking of both underactuated and actuated states, even though controlling the actuated states only. This field is still unexplored and highly challenging.

Bibliography

[1] Pietro Chevallard, "Mission Design and Modeling of Space Manipulators for In-Orbit Servicing Missions"

[2] Fatina Liliana Basmadji, "Space robot motion planning in the presence of non-conserved linear and angular momenta"

[3] F. Landis Markley, John L. Crassidis "Fundamentals of spacecraft attitude determination and control" (2014)

[4] B.Siciliano, "Robotics: modeling, planning and control"

[5] "Dynamics of Serial Multibody Systems Using the Decoupled Natural Orthogonal Complement Matrices"

[6] A. Ellery, "An engineering approach to the dynamic control of space robotic on-orbit servicers"

[7] Quan Vuong, "Machine Learning for Robotic Manipulation"

[8] Jianghua Duan, "Fast and Stable Learning of Dynamical Systems Based on Extreme Learning Machine"

[9] enfu Xu, "A modelling and simulation system of space robot for capturing non-cooperative target"

[10] Thai Chau Nguyen Huynh, "Adaptoie reactionless control of a space manipulator for post-capture of an uncooperative tumbling target"

[11] Panfeng Huang, "Tracking Trajectory Planning of Space Manipulator for Capturing Operation"

[12] Tomasz Rybus · Karol Seweryn · Jurek Z. Sasiadek, "Control System for Free-Floating Space Manipulator Based on Nonlinear Model Predictive Control (NMPC) "

[13] Kazuya Yoshida, "Space Robot Dynamics and Control: To Orbit, From Orbit, and Future"

[68] Lingling Shi Dynamic Modeling and Control of Free-Flying Space Robots

[15] Edoardo SERPELLONI, “Dynamics and Control of a Spacecraft-Manipulator System: Analisys, Simulation and Experiments”

[68] Mitsushige Oda , “On the dynamics and control of ETS-7 satellite and its robot arm ”

[68] Vincent Dubanchet, "Motion Planning and Control of a Space Robot to Capture a Tumbling Debris"

[18] Isacco Pretto, "Base reaction control of space manipulators"

[19] Zhicheng Xie a, Tao Sun a, Trevor Kwan, "Motion control of a space manipulator using fuzzy sliding mode control with reinforcement learning"

[68] Lingling Shi, "Robust coordinated control of a dual-arm space robot"

[68] Luca Didonè, "Robust Guidance of a Free-Floating Manipulator with Gaussian Mixtures Models"

[68] Riccardo Cipollone Adrea De Vittori, "Machine Learning References"

[68] Angel Flores-Abad, "A review of space robotics technologies for on-orbit servicing"

[68] Alex Ellery, "Tutorial Review on Space Manipulators for Space Debris Mitigation"

[25] S. Mohammad Khansari-Zadeh, "Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models"

[68] Sheng Xu, "Robot trajectory tracking control using learning from demonstration method"

[68] Rongrong Liu, "Deep Reinforcement Learning for the Control of Robotic Manipulation: A Focussed Mini-Review"

[28] Hai Nguyen, "Review of Deep Reinforcement Learning for Robot Manipulation"

[29] Hadi Hormozi, "The Classification of the Applicable Machine Learning Methods in Robot Manipulators"

[68] Josep Virgili-Llop, Spacecraft robotics toolkit: an open-source simulator for spacecraft robotic arm dynamic modeling and control

[31] Taiguo Li, "A Path Planning Algorithm for Space Manipulator Based on Q-Learning"

[68] Borna Monazzah Moghaddam, "On the guidance, navigation and control of in-orbit space robotic missions: Asurvey and prospective vision"

[68] Ali Abdi, A Novel Hybrid Path Planning Method Based on Q-Learning and Neural Network for Robot Arm

[34] Vafa, "A Virtual Manipulator Model for Space Robotic Systems"

[68] G.M.Buizza, "Robotic mobile manipulation in a completely unknown environment"

[36] A. Seddaoui, "Precise Motion Control of a Space Robot for In-Orbit Close Proximity Manoeuvres"

[68] Papadopoulos, "The kinematics, dynamics and control of free flying and free floating space roboyic systems"

[68] Otte, "A Survey of Machine Learning Approaches to Robotic Path-Planning"

[68] Das, "An Improved Q-learning Algorithm for Path-Planning of a Mobile Robot"

[68] "A robust attitude controller for a spacecraft equipped with a robotic manipulator"

[41] Lijun Zong, "Concurrent base-arm control of space manipulators with optimal rendezvous trajectory"

[42] Wankyun Chung ,"Motion Control"

[43] Tracking Control for the Grasping of a Tumbling Satellite With a Free-Floating Robot

[68] JohnJ.Craig, "Introduction to Robotics"

[45] Y. Huang, Y. S. Yong, R. Chiba, T. Arai, T. Ueyama, and J. Ota, "Kinematic control with singularity avoidance for teaching-playback robot manipulators"

[46] Matteo Acquarone, "Ottimizzazione di un agente di Q-learning per il controllo di veicoli ibridi elettrici"

[47] Gang Chen, Long Zhang, "Singularity Analysis of Redundant Space Robot with the Structure of Canadarm2"

[68] Le Anh Tuan, Soon-Geul Lee "Nonlinear Feedback Control of Underactuated Mechanical Systems"

[68] Course Notes for MIT, "Learning, Planning, and Control for Efficient and Agile Machines"

[50] Prof. De Luca, "Trajectory Tracking Control"

[68] Filippo Caldera, "Sliding Mode Attitude Control for Small Satellite with Manipulator"

[52] Eric Martin, "Dynamic interaction of a space manipulator with its base attitude controller"

[53] Donald J. Kessler, The kessler syndrome: implications to future space operations

[54] Gai, S.N.; Sun, R.; Chen, S.J.; Ji, S. 6-dof robotic path planning based on artificial potential field method

[55] A. Pedro Aguiar. Trajectory-Tracking and Path-Following of Underactuated Autonomous Vehicles with Parametric Modeling Uncertainty

[56] R. B. Ashith Shyam. Autonomous Robots for Space: Trajectory Learning and Adaptation Using Imitation

[57] Wikipedia. Sylvester's criterion

[58] R.Lampariello, G. Hirzinger. Freeflying robots - inertial parameters identification and control strategies

[59] Seyed Sina Mirrazavi Salehian. A Dynamical System Approach for Softly Catching a Flying Object: Theory and Experiment

[60] Mengyu Ji, Long Zhang, Shuquan Wang. A Path Planning Approach Based on Q-learning for Robot Arm

[61] Meng Guan, Fu Xing Yang. Research on path planning of mobile robot based on improved Q method

[68] The Path Planning of Mobile Robot by Neural Networks and Hierarchical Reinforcement Learning

[63] L. Kavraki, P. Svestka, and M. H. Overmars. Probabilistic roadmaps for path planning in high-dimensional configuration spaces.

[64] S. M. LaValle. Rapidly-exploring random trees: A new tool for path planning

[65] Vafa, Z. and Dubowsky S. On the Dynamics of Space Manipulators Using the Virtual Manipulator, with Applications to Path Planning

[66] Om P. Agrawal and Yangsheng Xu. On the Global Optimum Path Planning for Redundant Space Manipulators

[67] S.Dubowsky, E.E.Vance, M.A.Torres The Control of Space Manipulators Subject to Spacecraft Attitude Control Saturation Limits

[68] Nakamura, Y. and Mukherjee, R. Nonholonomic Path Planning of Space Robots via Bi-directional Approach

[69] S.M.Lavalle. Rapidly exploring random trees: a new tool for path planning.

Acknowledgements

Two years ago I started what has probably been the most thrilling and demanding challenge in my life. I would like to thank all the people that have helped me to conclude it.

First of all, I would like to thank my supervisor Professor Mauro Massari for his availability and timeliness in responding to my needs; this has been a challenging work and his wise advice and support have been fundamental for me. Thank you very much.

Then, I would like to thank my family. I would have never been able to arrive till the end without your never ending encouragement. Thank you for the support and for always believing in me. I love you.

Finally, I want to thank my friends for helping me in the difficult moments when I thought I wouldn't make it. You have always been ready to assist and listen to me; I will never forget it.

To all of you, each one special for different reasons, I'd like to express my deep gratitude.