Politecnico di Milano

Scuola di Ingegneria Industriale e dell'Informazione

Corso di laurea in Space Engineering

Robust Guidance of Robotic Manipulators based on Stable-Estimator for Dynamical Systems

Relatore: Prof. Mauro Massari

Tesi di Laurea Magistrale di:
Mauro Luigi Tramis Matr. 883336

Anno Accademico 2019-2020

Dedicata a mio zio Pino

Ringraziamenti

Questo è un traguardo importante, un obiettivo che ha richiesto fatica, determinazione, dispendio di energie fisiche e mentali, e che, a volte, ha comportato il manifestarsi di momenti di sofferenza, di paura e sconforto. Arrivare a questo punto dimostra che nella vita, per realizzare i propri obiettivi ed ottenere ciò che per noi conta davvero, bisogna resistere, dimostrare che nonostante tutto ci si può riuscire, anche a fronte dei fallimenti che si incontrano durante il cammino, pure se, anche solo per un secondo, si pensa di non potercela fare e di non esserne all'altezza. L'essere giunto a questo risultato non è un merito unicamente mio, sono molte le persone che ho da ringraziare per l'aiuto, il supporto e l'affetto che mi hanno dato in tutti questi anni e che, senza le quali, forse non sarei qui oggi a festeggiare.

Primi fra tutti devo ringraziare i miei genitori, sono loro che mi hanno cresciuto ed educato, mi hanno fatto diventare la persona che sono oggi e, per questo, non smetterò mai di essere loro grato. Devo ringraziarli per il continuo sostegno che mi hanno dimostrato, sia che le cose andassero bene e sia che andassero male, mai mi hanno fatto sentire inadeguato al percorso che ho scelto, anzi, ne sono sempre stati orgogliosi. Grazie per avermi fatto crescere, per avermi protetto e reso il testardo che sono oggi, perché ci sono state volte in cui mi sono sentito non all'altezza, come se stessi inseguendo un qualcosa di inarrivabile per le mie capacità, un fallimento scritto che avrebbe portato delusione non solo a me ma anche a voi; mai una sola volta mi avete fatto mancare il vostro appoggio, mi avete dato la forza per scacciare questi pensieri, la forza di non arrendermi e di poter dire oggi: ce l'ho fatta.

Un ringraziamento particolare va anche a mia nonna Anna, che ha contribuito attivamente alla mia crescita, che mi è stata sempre vicino e che riesce ad esserlo anche ora che siamo lontani; la nonna a cui ho sempre voluto bene, che me ne ha sempre voluto e per la quale sono stato e continuerò ad essere speciale.

Un grazie ai numerosi amici che mi hanno accompagnato fino ad oggi, chi da più e chi da meno tempo; agli amici che hanno reso questi anni unici ed indimenticabili, con cui ho passato momenti belli e momenti brutti; gli amici che riescono sempre a strapparmi un sorriso, qualsiasi sia l'occasione, che riescono a risollevar-

mi il morale e a distrarmi da quelli che sono stati gli aspetti opprimenti della mia carriera universitaria, gli amici con cui sono sempre stato semplicemente me stesso.

Grazie al gruppo di amici che sopravvive dai tempi del liceo e che diventa sempre più numeroso e unito; il gruppo con cui si fa ogni cosa, le uscite serali, le vacanze, le grigliate, e in particolare a Filippo, Emanuele, Mary, Federica, Romeo, Alessandro, Valentina, Gioacchino, Michele, Marco, Luca M., Simone, Christian e Alberto T.

Grazie al gruppo dei Fndani, Sara, Stefano e Roberta, per le cene, le sciate, le vacanze di Pasqua ai Ronchi, le serate insieme, i "bbevi" e i black russian; grazie per tutti i momenti passati insieme.

Grazie al magnifico gruppo Cilli, in particolare a Davide, Gigi, Federico, Matteo, Luca M. e al capitano Alberto Q., per le ormai innumerevoli vittorie, per le sconfitte affrontate a testa alta, per il meraviglioso gruppo che si è formato, per la lotteria truccata di fine anno, per i 5 euro e, soprattutto, per le cene da Carlo.

Un grazie molto particolare ad Alberto C., Federico T., Luca M. e Jacopo, perché sono quegli amici irrinunciabili, quelli che in tutti questi anni mi hanno sopportato più degli altri, sono stati a sentire i miei problemi e le mie paranoie, mi hanno aiutato nei momenti brutti. Quelli con cui si è condiviso davvero tutto, fino in fondo; quelli che, sono certo, ci saranno sempre e per i quali ci sarò sempre.

Infine, vorrei ringraziare lo zio Pino, al quale ho dedicato questa tesi, perché anche lui è stato un sostegno importante in questi anni, perché so che è sempre stato orgoglioso di me e non lo nascondeva con nessuno, perché, tragicamente, non ha potuto essere qui per festeggiare insieme a tutti noi.

GRAZIE

Contents

Nomenclature

Variables	**Description**
d	Dimension of the system
ξ	Vector of position coordinates
ξ^*	Vector of target coordinates
R^d	Set of real numbers
ϵ	Perturbations
Θ	Vector of motion parameters
π	Prior
μ	Mean value
Σ	Covariance matrix
q_{err}	Error between measured and desired joint configuration
$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}; \Theta)$	Probability Density Function
$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n} \| k)$	Conditional probability density function
V	Lyapunov function
$f(x), g(x)$	Function
$\mathcal{J}$	Cost function
N	Number of demonstrations in the training set
K	Number of Gaussian functions

Notations

Notation	Description
$\dot{()}$	First time derivative
$\ddot{()}$	Second time derivative
$\hat{()}$	Estimated value
$()^k$	Value considering the k^{th} Gaussian function
$()^T$	Value transposed
$()^{t,n}$	Value of the t^{th} datapoint of the n^{th} demonstration
$()_\xi$	Sub vector/matrix of indices 1:d
$()_{\dot{\xi}}$	Sub vector/matrix of indices d+1:2d
$()_{\xi\dot{\xi}}$	Sub matrix of indices (1:d,d+1:2d)
$()_{\dot{\xi}\xi}$	Sub matrix of indices (d+1:2d,1:d)

Acronyms

Acronym	Description
SEDS	Stable Estimator of Dynamical Systems
DS	Dynamical System
ODE	Ordinary Differential Equation
BIC	Bayesian Information Criterion
DIC	Deviance Information Criterion
AIC	Akaike Information Criterion
EM	Expectation Maximization
MSE	Mean Square Error
NLP	Non-Linear Programming
SQP	Successive Quadratic Programming
GMM	Gaussian Mixture Model
GMR	Gaussian Mixture Regression
RMS	Root Mean Square
FWHM	Full Width at Half Maximum
PDF	Probability Density Function
GPR	Gaussian Process Regression
LWPR	Locally Weighted Projection Regression
MAP	Maximum A Posteriori

List of Figures

List of Tables

Abstract

Robotics is one of the sectors that has made enormous strides in recent decades; the possibility of being able to reproduce human actions and movements, from the simplest to the most complex, can become of vital importance in various aspects of daily life. The possibility of constructing and being able to perfectly control a robotic object can also have infinite potential in numerous fields of application, for example in medicine to create increasingly realistic prostheses and able to best replicate the functionality of a limb; as in space or research fields, to carry out actions that would be too risky or even impossible for a human being. The purpose of this thesis is to provide a method capable of teaching a robot certain movements and making it able not only to reproduce them, but also to create new ones based on the needs and obstacles that it may encounter along its path. To do this, the proposed method uses SEDS algorithms that allow to model the dynamics of these movements in an efficient and, above all, faithful way; the reliability of this method will be shown by verifying that the robot considered is actually capable of reproducing movements between two designated positions.

Sommario

La robotica è uno dei settori che ha fatto enormi passi avanti negli ultimi decenni; la possibilità di poter riprodurre azioni e movimenti umani, da quelli più semplici a quelli più complessi, può diventare di vitale importanza in vari aspetti della vita quotidiana. Essere in grado di costruire e poter controllare ala perfezione un oggetto robotizzato può inoltre avere un infinito potenziale in numerosi campi di applicazione, ad esempio in medicina per creare protesi sempre più realistiche e in grado di replicare al meglio le funzionalità di un arto; come anche in campo spaziale o di ricerca, per operare azioni che per un essere umano sarebbero troppo rischiose se non addirittura impossibili. Lo scopo di questa tesi è di fornire un metodo in grado di insegnare ad un robot dei determinati movimenti e renderlo in grado non solo di riprodurli, ma anche di idearne di nuovi in base alle necessità e agli ostacoli che esso può incontrare lungo il proprio cammino. Per fare ciò il metodo che viene proposto utilizza gli algoritmi SEDS che permettono di modellizzare la dinamica di questi movimenti in modo efficiente e, soprattutto, fedele; verrà in seguito mostrata l'affidabilità di tale metodo verificando che il robot considerato sia effettivamente in grado di riprodurre movimenti tra due posizioni designate.

1 Introduction

The purpose of this work is to provide a method capable to make a robot learn discrete motions starting from a given set of demonstrations. In modelling, demonstrations are defined as point-to-point motions and describe different paths of different shapes but all of them reach the same final point (target) and are constrained in order to grant global asymptotic stability in that point. The final goal is to obtain a model capable to create new trajectories, always inside the same operational space of demonstrations and with the same characteristics of the initial set.

Procedure starts with the definition of a set of trajectories, taken randomly considering different paths of different shapes, that represents the motion I want to teach to the robot. I fix different starting points in the operational space randomly and define different trajectories with different behaviours with the only condition that all of them have to converge to the same target, in this work also taken randomly but that can be obviously fixed with a known point for a specific mission in case of necessity. After that I make some initial computations on all data points and define a non-linear autonomous dynamical system (from now on defined as DS) fixing some constraints in order to guarantee the global asymptotic stability at the target. At this point, I propose a learning method able to compute some time independent parameters from the initial set via an optimization problem under non-linear constraints; this method is called Stable Estimator of Dynamical Systems or SEDS. An advantage of setting these conditions on stability and time invariance is that I make the system capable of responding immediately to perturbations encountered during motion.

As already clarified, the process starts from the definition of an initial set of demonstrations, defined through a direct modelization of their trajectories in the operational space; indeed, to activate robot's learning process, I use a method called 'Programming by Demonstration', or PbD. This method is based on providing examples of the task of point-to-point motions that are previously defined from the robot's point of view; in this work initial motions are defined by programming their trajectories in the operational space but, with the correct instruments, they can be demonstrated by the user guiding the robot passively by back-driving or tele-operating it. Following I have to define what I need to imitate of these demonstrations and find a system to extract the generic characteristics of the dynamics.

For this purpose, I select the characteristics that are invariants across demonstrations; these characteristics should contain the main features of the desired task and need to be investigated considering some variations within a neighbourhood around the covered area.

The goal of this work is to obtain a proper model from demonstrations and verify its validity making it create new trajectories, starting from initial points different from the initial ones, capable to follow a path really close to the ones defined by demonstrations while converging to the same target point, obviously maintaining the condition of global asymptotic stability.

In following chapters I will discuss the entire procedure and explain in details all tools and algorithms used in the process. In Chapter 2 I will present the method I chose for this work, its concept and some of the theoretic aspects necessary for its application; in Chapter 3 I will expose the practical procedure of the method, defining constraints, fixing approximations and describing the optimization method I chose; Chapter 4 will present the design of the robot I considered and the model created in order to validate the results obtained by the use of algorithms while, in Chapter 5, all results and graphs obtained will be described and commented.

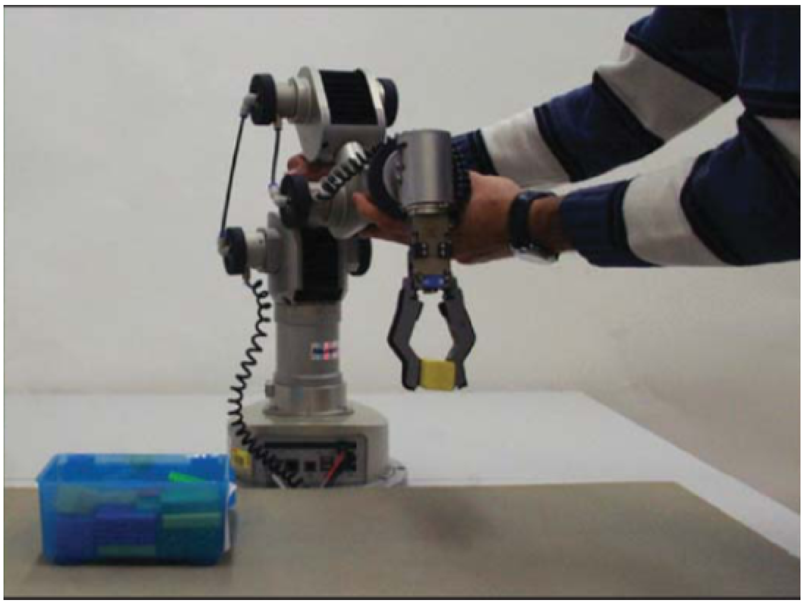


Figure 1: Katana-T robotic arm

2 SEDS: Concept and Theory

The SEDS method is proposed to make the robot learn discrete motions from a set of initial demonstrations and the reason to this circumstance is that it grants many advantages with respect to other methods.

Traditional means of encoding trajectories are based on spline decomposition after averaging across training trajectories, however these methods give a poor estimation of non-linear trajectories and are heavily dependent on time; this last aspect is the most troublesome because time dependency makes these techniques very sensitive to both temporal and spatial perturbations. SEDS method directly overcome this deficiency by considering only motion parameters that are time invariant, making its system robust against all types of perturbations.

Other approaches to statistical estimations use methods like Gaussian Process Regression (GPR), Locally Weighted Projection Regression (LWPR), or Gaussian Mixture Regression (GMR); however between all these methods, none optimize under constraint of making the system stable. In practice, they fail to ensure global stability, and they also rarely ensure local stability; this may converge to spurious attractors or miss the target. Unlike them, SEDS method works with some constraints set in order to guarantee the global asymptotic stability at the target.

2.1 Formalism

The encoding of point-to-point motions is formulated as a control law operated by an autonomous dynamical system. Let's consider a state variable $\xi \in R^d$ that can be used to define a discrete motion of a robotic system in an unique way and let define the set of the N given demonstrations as instances of a global motion model, driven by an ordinary differential equation, of first order and autonomous (ODE). I can define this global motion as

$$\dot{\xi} = f(\xi) + \epsilon \tag{1}$$

where f is a nonlinear continuous function, continuously differentiable with a single equilibrium point $\dot{\xi}^* = f(\xi^*) = 0$, θ is the set of parameters that I will consider while running the SEDS algorithms and ϵ represents the zero mean Gaussian noise.

This last term considers both inaccuracies coming from sensor measurements and errors from imperfect demonstrations.

Function f can be described using a set θ of parameters obtainable from the initial demonstrations using differential statistical approaches; thanks to this step, I can denote the noise-free estimation of f as $\hat{f}$, so that the estimation of the global motion will be

$$\dot{\xi} = \hat{f}(\xi) \tag{2}$$

According to these first two formulae, I can make two observations: first that the control law given by (2) generates trajectories without intersections and second that the motion of the system is defined uniquely by ξ, so its choice is a crucial point for the entire work.

Statistical approaches to modeling robot motions have become increasingly popular as a means to deal with noise inherent in any mechanical system. They have proved to be interesting alternatives to classical control and planning approaches when the underlying model cannot be well estimated. Existing approaches to the statistical estimation of f in (1) use either Gaussian Process Regression (GPR), Locally Weighted Projection Regression (LWPR), or Gaussian Mixture Regression (GMR), where the parameters of the Gaussian Mixture are optimized through Expectation Maximization (EM). GMR and GPR methods find an optimal model of $\hat{f}$ by maximizing the likelihood that the complete model represents the data well, while LWPR method minimizes the mean-square error between the estimates and the data.

Because all of the aforementioned methods do not optimize under the constraint of making the system stable at the attractor, they are not guaranteed to result in a stable estimate of the motion. In practice, they all fail to ensure the global stability and most of the time the local stability of $\hat{f}$, and thus may converge to a spurious attractor or completely miss the target; these errors are due to the fact that there is yet no theoretical solution for ensuring stability of arbitrary non-linear autonomous DS.

The use of DS is really advantageous, and the particular reason to this circumstance is that it makes a robot capable to adapt its trajectory instantly against perturbations; a controller driven by a DS is strong against perturbations because it incorporates all possible solutions to reach target inside one single function $\hat{f}$. Perturbations may either be due to a sudden displacement of the target with respect to robot or to delays in the execution of the tasks; I will refer to these two types as spatial and temporal perturbations respectively.

2.2 Model

During process motion is represented in a kinematic coordinates system and I assume that exists a low-level controller capable to convert its kinematic variables into motor commands. In figure 2 is presented a schematic view of the entire system, composed by two main loop paths. The inner loop consists in two blocks: a system block that models robot's dynamics and gives as outputs the robot's joint angles and their time derivatives, described by q, $\dot{q}$ and $\ddot{q}$; and a controller that generates the motor commands, described by u, required to follow the desired motion.

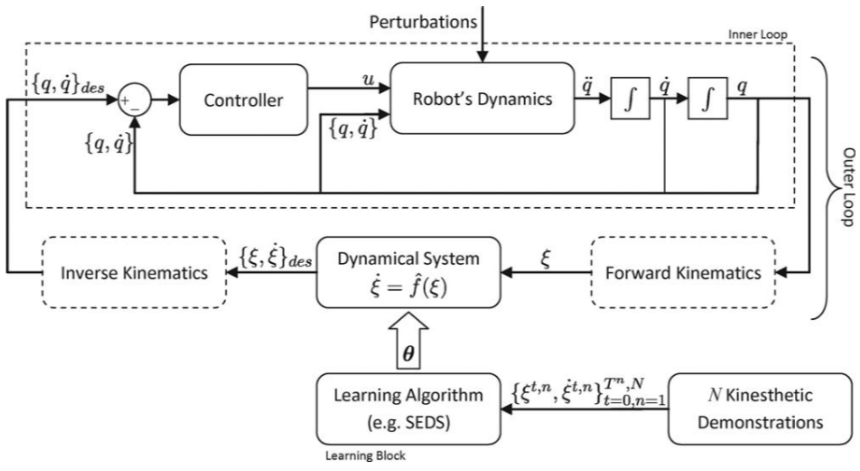


Figure 2: Scheme of the double-loop DS

The outer loop takes q from the robot dynamics and a forward kinematics block changes it into Cartesian coordinates of the end effector, described by ξ, then a second system block represents the dynamic of the desired positions of the end effector considering both ξ and the set of parameters θ that are obtained from the N initial demonstrations. The output of this second dynamics are ξ and $\dot{\xi}$ of the desired motion that are converted back to robot's joint angles before entering the first loop; here the desired results are confronted with the ones of the robot dynamics and, through the MSE method, this makes the controller able to define the u needed to make the motion closer and closer to the desired one.

Considering this architecture, both inner and outer loops need to be stable. Stability for inner loop requires the system to be input-to-state stable and the output should remain bounded, while outer loop stability is already ensured when learning the system.

The learning block determines a stable estimate of the DS that is used as the controller of the outer-loop path; but I assume this controller as not necessarily accurate because I want to focus my interest on designing a learning block that ensures stability. Learning process is data driven and uses a set of demonstrated trajectories in order to determine the parameters θ of my system; it proceeds as an optimization problem, that needs to satisfy asymptotic stability at the target. Both the optimization problem and the constraints necessary in order to grant asymptotic stability to DS will be detailed later, both in Chapter 3

2.3 Gaussian Functions

In mathematics, a Gaussian function is a function of the form

$$f(x) = a \cdot \exp\left(-\frac{(x-b)^2}{2c^2}\right)$$

considering arbitrary real constants a, b and a non zero c; the graph of a Gaussian has a characteristic symmetric "bell curve" shape. Parameter a represents the height of the curve's peak, parameter b defines the position of the center of the peak while parameter c, also known as standard deviation, or sometimes called Gaussian RMS width, controls the width of the "bell".

These functions are frequently used to represent the probability density function of a normally distributed random variable with expected value $\mu = b$ and variance $\sigma^2 = c^2$; presenting a form like:

$$g(x) = \frac{1}{\sigma\sqrt{2\pi}} \exp\left(-\frac{1}{2}\frac{(x-\mu)^2}{\sigma^2}\right)$$

They are also widely used in statistics to describe a normal distributions, in signal processing to define filters and in mathematics to solve heat and diffusion equations. Gaussian functions arise by composing the exponential function with a concave quadratic function:

$$f(x) = e^{\alpha x^2+\beta x+\gamma}$$

where: $\alpha = -0.5/c^2$
$\beta = b/c^2$
$\gamma = 0.5(\log(a) - b^2)/c^2$
and their logarithm is a concave quadratic function.

Parameter c is related to the full width at half maximum (FWHM) of the peak according to FWHM $= 2\sqrt{2\ln 2}\ c \approx 2.35482c$, so function can also be expressed in terms of the FWHM, described by w:

$$f(x) = a \cdot \exp\left(-4 \cdot \frac{(\log 2)(x-b)^2}{w^2}\right)$$

The most common method for estimating the Gaussian parameters is to take the logarithm of the data and fit a parabola to the resulting data set. While this provides a simple curve fitting procedure, the resulting algorithm may be biased by excessively weighting small data values, which can produce large errors. This problem can be partially compensated by reducing the weight of small data values, but this too can be biased. In order to remove the bias, one can instead use an iterative procedure, in which weights are updated at each step.

Once one has an algorithm for estimating the Gaussian function parameters, it is also important to know how precise those estimations are; any least squares estimation algorithm can provide numerical estimates for the variance of each parameter.

2.4 Gaussian Mixture Models

A Gaussian Mixture Model (GMM) is a parametric probability density function represented as a weighted sum of Gaussian component densities. They are commonly used as a model of the probability distribution of continuous measurements or features in a biometric system, such as vocal-tract related spectral features. GMM parameters are estimated from training data using the iterative

Expectation Maximization algorithm (EM) or the Maximum A Posteriori (MAP) estimation from a well-trained prior model.

Considering a generic dataset, the goal is to find sets of points that appear close together, defining μ as the mean (or centroid) of the cluster. A popular clustering algorithm is known as K-means, which will follow an iterative approach to update the parameters of each cluster; more specifically, it will compute the means of each cluster, and then calculate their distance to each data point. This process is repeated until some convergence criterion is met, for example when no further changes in the cluster assignments are seen. One important characteristic of K-means is that it is a hard clustering method, which means that it will associate each point to one and only one cluster. A limitation to this approach is that there is no uncertainty measure or probability that gives informations about how much a data point is associated with a specific cluster; GMMs try to overcome this limitation.

A Gaussian Mixture is a function that is comprised of several Gaussians, each identified by $k \in 1, \ldots K$, where K is the number of clusters of the dataset considered. Each Gaussian k in the mixture is comprised of the following parameters: a mean μ that defines its centre, a covariance Σ that defines its width, and a mixing probability π that defines how big or small the Gaussian function will be.

The mixing coefficients are themselves probabilities and must meet this condition:

$$\sum_{k=1}^{K} \pi_k = 1$$

Next step is to determine the optimal values of these parameters and to achieve this I must ensure that each Gaussian fits the data points belonging to each cluster. In general, the Gaussian density function is given by:

$$\mathcal{N}(\mathbf{x}|\mu, \Sigma) = \frac{1}{(2\pi)^{D/2}|\Sigma|^{1/2}} exp\left(-\frac{1}{2}(\mathbf{x}-\mu)^T\Sigma^{-1}(\mathbf{x}-\mu)\right)$$

Where $\mathbf{x}$ represents vector of data points, D the number of dimensions of each data point, μ and Σ are the mean and covariance. Next, by finding the log of this equation, differentiating it with respect to the mean and covariance and equating it to zero, I will be able to find the optimal values for these parameters, and the solutions will correspond to the Maximum Likelihood Estimates (MLE).

2.5 Expectation Maximization Algorithm

Maximum Estimation is an approach to density estimation for a data set by searching across probability distributions and their parameters; it is a general and effective approach that underlies many machine learning algorithms, although it requires that the training data set is complete, so that all relevant interacting random variables are present.

The Expectation Maximization algorithm is an approach for performing maximum likelihood estimation in the presence of latent variables; it does this by first estimating the values for the latent variables, then optimizing the model, and after that repeating these two steps until convergence. It is an effective and general approach and is most commonly used for density estimation with missing data, such as clustering algorithms like the Gaussian Mixture Model.

It describes an iterative approach that works on two steps: first step attempts to estimate the missing or latent variables and for this reason it's called estimation-step or E-step, while the second one attempts to optimize the parameters of the model to best explain the data, and it's called maximization-step or M-step.

This algorithm has a wide range of application, although is most well known in machine learning for use in unsupervised learning problems, such as density estimation and clustering; in fact, the most discussed application of the EM algorithm is for clustering with a mixture model, a model comprised of an unspecified combination of multiple probability distribution functions. A statistical procedure or learning algorithm is used to estimate the parameters of the probability distributions to best fit the density of a given training data set; the Gaussian Mixture Model, or GMM, is a mixture model that uses a combination of Gaussian probability distributions and requires the estimation of the mean and standard deviation parameters for each.

Let's now consider an example: looking at a random variable Y and a measurement vector $y = (y_1, \ldots, y_N)^T$, the probability of receiving some measurement y_i is given by the probability density function (PDF) $p(y_i|\Theta)$, where the PDF is governed by the parameter Θ. The probability of having received the whole series of measurements is then

$$p(y|\Theta) = \prod_{i=1}^{N} p(y_i|\Theta)$$

The likelihood function is defined as $L(\Theta) = p(y|\Theta)$,and the maximum likelihood (ML) estimation of Θ is found by maximizing L. Often, it is easier to maximize the log-likelihood

$$log\ L(\Theta) = log\ p(y|\Theta)$$

$$= log \prod_{i=1}^{N} p(y_i|\Theta) = \sum_{i=1}^{N} logp(y_i|\Theta)$$

because since the logarithm is a strictly increasing function, L and $log(L)$ have the same maximum.

The EM algorithm facilitate parameter estimation by introducing the so called hidden random variables, which are not observed and therefore define the unobserved data; instead of looking at complete data, the algorithm can facilitate even more computation by dealing with unobserved data. In fact the EM algorithm faces the presence of hidden variables by first finding an estimate for the likelihood function, and then maximizing the whole term; in order to find this estimate, each entry of z is a realization of a hidden random variable.

Its expectation given the observed data y is:

$$E[log\ L_C(\Theta)|y, \Theta^{(i)}]$$

The EM algorithm maximizes this expectation:

$$\Theta^{(i+1)} = \underset{\Theta}{argmax}\ E[log\ L_C(\Theta)|y, \Theta^{(i)}]$$

Compared to the ML approach which involves just maximizing a log likelihood, the EM algorithm makes one step in between: the calculation of the expectation; this is denoted as the expectation step. In the maximization step, a new update $\Theta^{(i+1)}$ for Θ is found, that it maximizes the whole term. This whole procedure is repeated until some stopping criteria becomes very small.

3 Learning Approach

3.1 Set of Demonstrations

First step is to define the set of N demonstrations from which start all the analysis and computations in order to find the desired motion parameters. I fixed them by selecting some random points in the operational space and considering them as the starting points of my demonstrations; for this work I fixed nine trajectories, that describe different paths and different shapes. Coordinates of starting points and target point are reported in table 1, notice also that starting points are divided according to the shape of their corresponding trajectory.

Target	**Linear**		**Circular**		**Sinusoidal**	
{17 -31 -13}	start 1	{-13 -8 20}	start 4	{-16 -8 -38}	start 7	{-45 20 -47}
	start 2	{-30 5 17}	start 5	{46 49 50}	start 8	{24 17 6}
	start 3	{-1 45 -33}	start 6	{42 -20 -33}	start 9	{-23 4 39}

Table 1: Coordinates of target and starting points

Also target point has been fixed randomly in the operational space but, obviously, if the task requires to reach a point of known coordinates, it can be easily fixed with the desired destination. As told before, I fixed the demonstrations representing three different shapes, three tarjectories represent a linear case, three represent a circular case and last three represent a sinusoidal case; all of them are defined in order to converge to the target. However, as can be seen in figure 3, trajectories that follow the sinusoidal behaviour don't converge exactly to the target; for this cases I increased the number of data-points that describes the trajectories and made a computation about the distance between each point of their trajectories and the target, making them stop after reaching that point that results to be the closest to target.

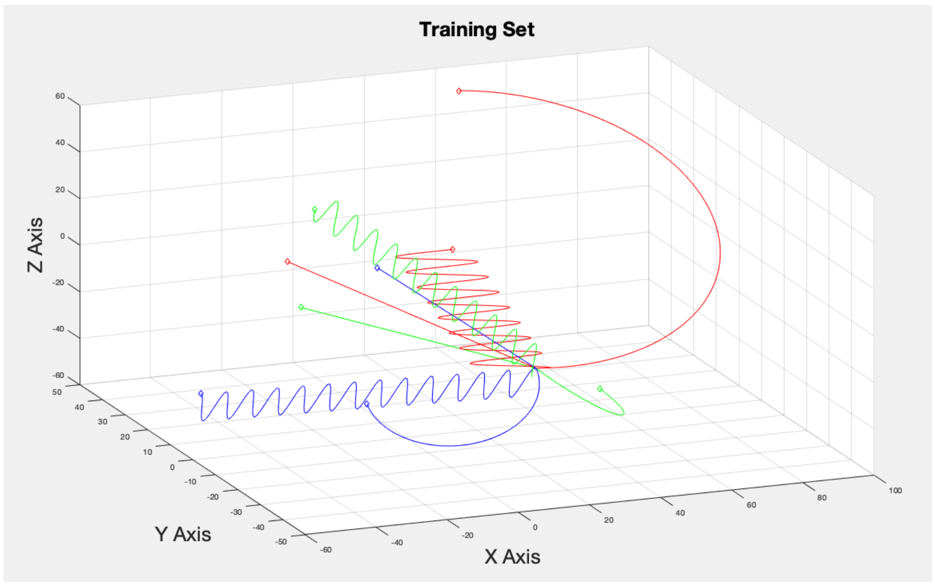


Figure 3: Training Set used to create the model

3.2 Algorithms

Having demonstrations, I can model $\hat{f}$ by using a finite mixture of Gaussian functions in a probabilistic framework. Using mixture models is a common approach for density approximation and it allows to define an appropriate model through an exchange between model complexity and variations of the available training data.

Mixture modeling is a method that builds a representation of the data density through a fixed number of mixture components; an optimal number of components could be found using different procedures, the most common are the Bayesian Information Criterion (BIC), the Deviance Information Criterion (DIC) or the Akaike Information Criterion (AIC).

While non-parametric methods offer an optimal regression they suffer for dimensionality, indeed the estimate regressor $\hat{f}$ grows linearly with the number of data, making itself poor for the re-computation of trajectories according to perturbations. To overcome this problem, I can use different techniques to reduce the dependency among the number of data points; however these techniques determine the optimal number of data points and become parametric.

By estimating f through a finite mixture of Gaussian functions, the unknown parameters that I need to compute for $\hat{f}$ become the priors π^k, the mean values μ^k

and the covariance matrices Σ^k, where $k = 1, ..., K$ indicates the parameter value considering the k^{th} Gaussian function; then I can collect all these parameters inside one single vector θ and define it as θ^k=$\{\pi^k, \mu^k, \Sigma^k\}$, or θ=$\{\theta^1...\theta^K\}$ considering all parameters of all Gaussian functions.

3.2.1 Procedure

Starting from the demonstrations presented in 3.1 first step is to compute their first time derivative so that I have all necessary materials to estimate parameters. I fix a known time step to be considered for all demonstrations and made computation considering a forward approximation and setting velocities for last data points equal to zero (supposing that the end effector will stop once it reaches the target). From this preprocess I obtain trajectories and their corresponding velocities for all data point of all demonstrations.

At this point I can make a first estimation for the desired motion parameters by using the EM method, an iterative method used to find local maximum likelihood or maximum a posteriori (MAP).

Now let's see how the motion parameters appear and how they are computed. Considering the k^{th} Gaussian function, the priors appear to be a scalar for each Gaussian, while the mean values and the covariance matrices are defined as:

$$\mu^k = \begin{pmatrix} \mu^k_{\xi} \\ \mu^k_{\dot{\xi}} \end{pmatrix}, \quad \Sigma^k = \begin{pmatrix} \Sigma^k_{\xi} & \Sigma^k_{\xi\dot{\xi}} \\ \Sigma^k_{\dot{\xi}\xi} & \Sigma^k_{\dot{\xi}} \end{pmatrix} \tag{3}$$

Given a generic set of N demonstrations, described by position and velocity vectors $\{\xi^{t,n}, \dot{\xi}^{t,n}\}_{t=0,n=1}^{T^n,N}$ where t and n refers to the t^{th} trajectory data point of the n^{th} demonstration, each recorded point in the trajectories $[\xi^{t,n}; \dot{\xi}^{t,n}]$ is associated with a probability density function (PDF) $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n})$ that can be defined as:

$$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}; \theta) = \sum_{k=1}^{K} \mathcal{P}(k)\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k) \quad \begin{cases} \forall n \in 1 \dots N \\ t \in 0 \dots T^n \end{cases} \tag{4}$$

where $\mathcal{P}(k) = \pi^k$ represents the prior for the k^{th} Gaussian function, and $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k)$ is the conditional probability density function that can be computed from the following formula

$$\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|k) = \mathcal{N}(\xi^{t,n}, \dot{\xi}^{t,n}; \mu^k, \Sigma^k)$$

$$= \frac{1}{\sqrt{(2\pi)^{2d}|\Sigma^k|}} e^{-\frac{1}{2}([\xi^{t,n}, \dot{\xi}^{t,n}]-\mu^k)^T(\Sigma^k)^{-1}([\xi^{t,n}, \dot{\xi}^{t,n}]-\mu^k)} \tag{5}$$

Taking the posterior mean estimate of $\mathcal{P}(\dot{\xi}|\xi)$ yields

$$\dot{\xi} = \sum_{k=1}^{K} \frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^{K}\mathcal{P}(i)\mathcal{P}(\xi|i)} (\mu_{\dot{\xi}}^k + \Sigma_{\dot{\xi}\xi}^k(\Sigma_{\xi}^k)^{-1}(\xi - \mu_{\xi}^k)) \tag{6}$$

Notation of equation (6) can be simplified through a change of variables and some new arrangements by defining

$$\begin{cases} A^k = \Sigma_{\dot{\xi}\xi}^k(\Sigma_{\xi}^k)^{-1} \\ b^k = \mu_{\dot{\xi}}^k - A^k\mu_{\xi}^k \\ h^k(\xi) = \frac{\mathcal{P}(k)\mathcal{P}(\xi|k)}{\sum_{i=1}^{K}\mathcal{P}(i)\mathcal{P}(\xi|i)} \end{cases} \tag{7}$$

Then, substituting (7) into (6) brings to

$$\dot{\xi} = \hat{f}(\xi) = \sum_{k=1}^{K} h^k(\xi)(A^k\xi + b^k) \tag{8}$$

At this point $\hat{f}$ is expressed as a nonlinear sum of linear DS. According to equation (8) I can say that the nonlinear weighting terms $h^k(\xi)$, where $0 < h^k \leq 1$, give a measure of the relative influence of each Gaussian function locally; due to this terms the resulting function $f(\xi)$ is nonlinear and flexible enough to model a wide variety of motions. If I estimate this mixture using classical methods, such as the EM technique mentioned before, I cannot guarantee the asymptotic stability of the system, so I have to determine sufficient conditions on the learning parameters θ to ensure asymptotic stability of $\hat{f}(\xi)$; conditions that will be presented and discussed in Section 3.4.

Now what I need to do is determine a procedure to compute the unknown parameters from equation (8), $\theta = \{\pi^1 \dots \pi^K; \mu^1 \dots \mu^K; \Sigma^1 \dots \Sigma^K\}$, such that the resulting model is globally asymptotically stable. As already said, the learning algorithm that I propose is called SEDS and is based on computing optimal values of θ by solving an optimization problem under the constraint of global asymptotic stability. Two different methods are provided to solve the optimization problem: the Mean Square Error (MSE) method and the log-likelihood method.

3.2.2 MSE method

The MSE method can be summarized as

$$\min_{\theta} J(\theta) = \frac{1}{2\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} \|\hat{\dot{\xi}}^{t,n} - \dot{\xi}^{t,n}\|^2 \tag{9}$$

where $\hat{\dot{\xi}}^{t,n} = \hat{f}(\xi^{t,n})$ and subjected to

$$\begin{cases} (a) & b^k = -A^k \xi^* \\ (b) & A^k + (A^k)^T < 0 \\ (c) & \Sigma^k > 0 \\ (d) & 0 < \pi^k \leq 1 \\ (e) & \sum_{k=1}^{K} \pi^k = 1 \end{cases} \qquad \forall k \in 1 \dots K \tag{10}$$

The first two constraints in (10) are stability conditions, while the last three are imposed by the nature of the Gaussian mixture model to ensure that Σ^k are positive definite matrices, priors π^k are positive scalars smaller than or equal to one, and sum of all priors is equal to one.
This method can be formulated as a non-linear programming (NLP) problem and can be solved using standard constrained optimization techniques; in this work I use a successive quadratic programming (SQP) approach that relies on a quasi-Newton method to solve the optimization problem.
SQP minimizes a quadratic approximation of the Lagrangian function over a linear approximation of the constraints. This implementation has several advantages: firstly, I have an analytic expression of the derivatives, that grants a significant improvement of performances; and secondly, the code is customized to solve my specific problem.

A feasible solution to these NLP problems always exists. An efficient feasible initial guess for the optimization parameters is given by an algorithm capable of transforming covariance matrices obtained from demonstrations so that they satisfy optimization constraints given by (10b) and (10c); after that it computes an initial mean guess, given by

$$\tilde{\mu}^k_{\dot{\xi}} = \tilde{\Sigma}^k_{\dot{\xi}\xi}(\tilde{\Sigma}^k_{\xi})^{-1}(\tilde{\mu}^k_{\xi} - \xi^*) \tag{11}$$

and gives as output the initial guesses of the optimization parameters θ. Then, starting from this estimation of π^k, μ^k and Σ^k, with $k = 1 \dots K$, the solver tries to optimize values of θ in order to minimize the cost function J.

Since the NLP problem is non convex, I cannot ensure that the result is the globally optimal solution; solvers are usually sensitive to initialization process of parameters and will often converge to a series of local minima. However, running algorithms with the initial guesses described before usually brings to a good local minimum.
To choose the optimal set of Gaussians I use the BIC, that grants a good trade off between optimizing the model's probability and the number of parameters that are needed to encode the data

$$BIC = \mathcal{T}J(\theta) + \frac{n_p}{2}log(\mathcal{T}) \tag{12}$$

where $J(\theta)$ is the cost function of the model that is computed using (9), and n_p is the total number of free parameters.
In the case of MSE approach the number of parameters to be estimated is $K(1 + 3d(d + 1))$ since when constructing $\hat{f}$, the term Σ^k_{ξ} is not used and can be omitted during optimization.

3.2.3 Likelihood method

The alternative method to MSE presented before is the log-Likelihood method, that can be described by

$$\min_{\theta} J(\theta) = -\frac{1}{\mathcal{T}} \sum_{n=1}^{N} \sum_{t=0}^{T^n} log\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|\theta) \tag{13}$$

where $\mathcal{P}(\xi^{t,n}, \dot{\xi}^{t,n}|\theta)$ is given by (4), and $\mathcal{T} = \sum_{n=1}^{N} T^n$ is the total number of training data points; constraints are the same for MSE case, given by (10).

Likelihood method has the same characteristics of MSE, it can be formulated as a NLP problem and solved using SQP approach; the main difference is in the number of parameters that have to be estimated. In Likelihood case I cannot omit Σ^k and the total number of parameters to estimate becomes $K(1+3d+2d^2)$, where K is the number of Gaussian Functions considered and d describes system dimensions; however, their number can be reduced since the constraints given by (10a) provide an explicit formulation to compute $\mu_{\dot{\xi}}^k$ from other parameters, so the total number of parameters to construct a GMM with K Gaussians is $K(1+2d(d+1))$. Anyway, for both approaches, learning grows linearly with the number of Gaussians and quadratically with dimensions.

3.3 Computation analysis

In previous section I exposed the theory behind this work, describing physical environments, procedures and main formulae involved; but now I need to present some of the expedients and precautions that I considered during numerical computations. All computations can be grouped in three parts to be operated in the same order of the following list:

- Time Derivatives
- Expectation Maximization
- Mean Square Error

First part requires simply a reorganization of data coming form the set of demonstrations that I already described in 3.1 and the computation of their first time derivatives; in order to do that, I selected as computational method a simple finite difference approximation method, of course after fixing a constant time step.

Second step is to operate the EM method described in 2.5; this portion didn't require any specific approximation for computing the initial guesses, excepting for the ones already described previously and correlated to stability problems of the system.

Third and last part operates the real computation of motion parameters following the MSE method detailed in 3.2.2 and, about that, I need to make a clarification. Once the cost function is defined, in order to solve the problem related to it's minimum values, I used a function already present in the software library named $fmincon$; this function implements four different algorithms but I selected the SQP algorithm to solve the constrained system for the reasons already discussed.

3.4 Stability Analysis

A formal definition of stability could be given by the following sentence: the function $\hat{f}$ is globally asymptotically stable at the target ξ^* if $f(\xi^*) = 0$ and the generated motion converges asymptotically to ξ^*, as

$$\lim_{t \to \infty} \xi^t = \xi^* \qquad \forall \xi^0 \in R^d \tag{14}$$

$\hat{f}$ is locally asymptotically stable if it converges to ξ^* only when ξ^0 is contained in a subspace $\mathcal{D} \subset R^d$. Nonlinear DS are inclined to instabilities; so ensuring that the estimate $\hat{f}$ results in asymptotically stable trajectories, that are trajectories convergent asymptotically to the attractor, is a fundamental requirement for $\hat{f}$ to provide a useful control policy. In this work I formulate the problem to estimate f and its parameters θ as a constrained optimization problem, for which I maximize the accuracy of the reconstruction while guaranteeing its global asymptotic stability at the target.

Stability analysis of DS is a wide subject, which can generally be divided into linear and nonlinear cases. Considering a linear DS, that can be written as

$$\dot{\xi} = A\xi + b \tag{15}$$

its asymptotic stability can be granted by simply requiring the eigenvalues of the matrix A to be negative. For nonlinear DS, instead, the stability analysis is a lot more complex and still an open question; pay also attention to the fact that the intuition that the nonlinear function $f(\xi)$ should be stable if all eigenvalues of matrices A^k, with $k = 1 \ldots K$, have strictly negative real parts is not properly true; the reason to this circumstance is that also if single matrices A^k determines a stable system, their combination could not grant stability for the entire system.

Next, I determine sufficient conditions to ensure global asymptotic stability of a series of nonlinear DS given by (6).

Theorem: Assuming that the state trajectory evolves according to (8), the function that is described by it is globally asymptotically stable at the target ξ^*

in R^d if

$$\begin{cases} (a) \quad b^k = -A^k \xi^* \\ (b) \quad A^k + (A^k)^T < 0 \end{cases} \qquad \forall k = 1 \dots K \tag{16}$$

where $(A^k)^T$ is the transpose of A^k, and < 0 refers to the negative definiteness of a matrix.

I start the demonstration of this theorem recalling the Lyapunov conditions for asymptotic stability of an arbitrary DS.

Lyapunov Stability Theorem: A DS that is determined by the function $\xi = f(\xi)$ is globally asymptotically stable at the point ξ^* if there exists a continuous and continuously differentiable Lyapunov function $V(\xi) : R^d \to R$ such that

$$\begin{cases} (a) \quad V(\xi) = 0 \qquad \forall \xi \in R^d, \quad \xi \neq \xi^* \\ (b) \quad \dot{V}(\xi) = 0 \qquad \forall \xi \in R^d, \quad \xi \neq \xi^* \\ (c) \quad V(\xi^*) = 0 \qquad \dot{V}(\xi^*) = 0 \end{cases} \tag{17}$$

Note that $\dot{V}$ is a function of both ξ and ξ^*; and remembering that ξ^* can be expressed in terms of ξ using (8), I can say that function $\dot{V}$ only depends on ξ. Consider a Lyapunov function $V(\xi)$ of the form

$$V(\xi) = \frac{1}{2}(\xi - \xi^*)^T(\xi - \xi^*) \tag{18}$$

I can immediately observe that $V(\xi)$ is a quadratic function and that it satisfies condition (17a). Condition given by (17b) follows from taking the first derivative with respect to time; then I have

$$\dot{V}(\xi) = \frac{dV}{dt} = \frac{dV}{d\xi}\frac{d\xi}{dt}$$

$$= \frac{1}{2}\frac{d}{d\xi}((\xi - \xi^*)^T(\xi - \xi^*))\dot{\xi}$$

$$= (\xi - \xi^*)^T \xi^* = (\xi - \xi^*)^T \hat{f}(\xi)$$

$$= (\xi - \xi^*)^T \underbrace{\sum_{k=1}^{K} h^k(\xi)(A^k\xi + b^k)}_{=\dot{\xi}\ (from(8))}$$

$$= (\xi - \xi^*)^T \sum_{k=1}^{K} h^k(\xi)(A^k(\xi - \xi^*) + \underbrace{A^k\xi^* + b^k}_{=0\ (from(11a))})$$

$$= (\xi - \xi^*)^T \sum_{k=1}^{K} h^k(\xi) A^k(\xi - \xi^*)$$

$$= \sum_{k=1}^{K} \underbrace{h^k(\xi)}_{h^k>0} \underbrace{(\xi - \xi^*)^T A^k(\xi - \xi^*)}_{<0} < 0 \qquad \forall \xi \in R^d, \quad \xi \neq \xi^* \tag{19}$$

Conditions given by (17c) are satisfied when substituting $\xi = \xi^*$ into (18) and (19)

$$V(\xi^*) = \frac{1}{2}(\xi - \xi^*)^T(\xi - \xi^*)\bigg|_{\xi=\xi^*} = 0 \tag{20}$$

$$\dot{V}(\xi^*) = \sum_{k=1}^{K} h^k(\xi)(\xi - \xi^*)A^k(\xi - \xi^*)\bigg|_{\xi=\xi^*} = 0 \tag{21}$$

Therefore, an arbitrary ODE function $\dot{\xi} = \hat{f}(\xi)$, given by (8), is globally asymptotically stable if conditions of (16) are satisfied. Conditions (16a) and (16b) are sufficient to ensure that an arbitrary nonlinear function that is given by (8) is globally asymptotically stable at the target ξ^*. This type of model is advantageous because it grants that, starting from any point in the space, the trajectory always converges to the target.

4 Simulation

4.1 Concept

Once all computations have been completed, I obtained the motion parameters that should make the robot capable to learn the desired trajectory, in order to take it from a random starting point to the fixed target; however I'm not completely sure that the numerical results acquired are valid to make my model work correctly.

In order to verify results, I need to reproduce the model already described in 2.2 and shown in figure 2; by selecting a new random starting point and setting a precise robot design, already shown in Section 4.2, I created and run a dynamic simulation using Simulink software. This time, the learning parameters are known, and I can use them to compute both position and velocity of the desired motion through the methods detailed previously in Section 3.2.2.

4.2 Robot Design

The robot geometry that I chose to use in this work is almost the same of the one used in another work of similar goal. I designed my robot following the geometry of a PUMA560 robotic arm, shown in figure 4, considering the same movement capability; however, my arm has only three free joints with respect to the six joint of PUMA.

According to my final goal, three free joints are sufficient, in fact for this work I don't need to reach target with a defined orientation of the end effector, but I simply need to reach it; first joint is the one that makes link 1 of the robot capable to rotate around axis z of the triad located in the origin while both second and third joints make their corresponding links rotate the new z axis of their triads (same direction of axis x of origin triad) so that the end effector can reach any point inside the workspace.

Numerical geometric values for my robot arm has been fixed according to the definition of my workspace, usually should be the opposite but because I don't have any physical constrain for it I decided to adapt geometry to space. In table 2 are reported the main geometric values of my design, called Denavit-Hartenberg parameters; each column describes one of these parameters that are, respectively, the twist angle α_{i-1}, angle between z_{i-1} and z_i measured about x_{i-1}, the link length a_{i-1}, distance between z_{i-1} and z_i measured along x_{i-1}, the offset length d_i, distance between x_{i-1} and x_i measured along z_i, and the joint angle θ_i, angle from x_{i-1} to x_i measured about z_i; this last angle is unknown because it's the angle that defines the configuration of the robotic arm that will be computed later. Note also that parameters defined with $()_i$ refers to the same triad of the joint I am considering, while the ones defined with $()_{i-1}$ refers to the previous one.

$\boldsymbol{i}$	$\boldsymbol{\alpha_{i-1}}$	$\boldsymbol{a_{i-1}}$	$\boldsymbol{d_i}$	$\boldsymbol{\theta_i}$
1	0	0	0	θ_1
2	$\pi/2$	0	5	θ_2
3	0	45	0	θ_3
end-effector	0	45	0	-

Table 2: DH parameters

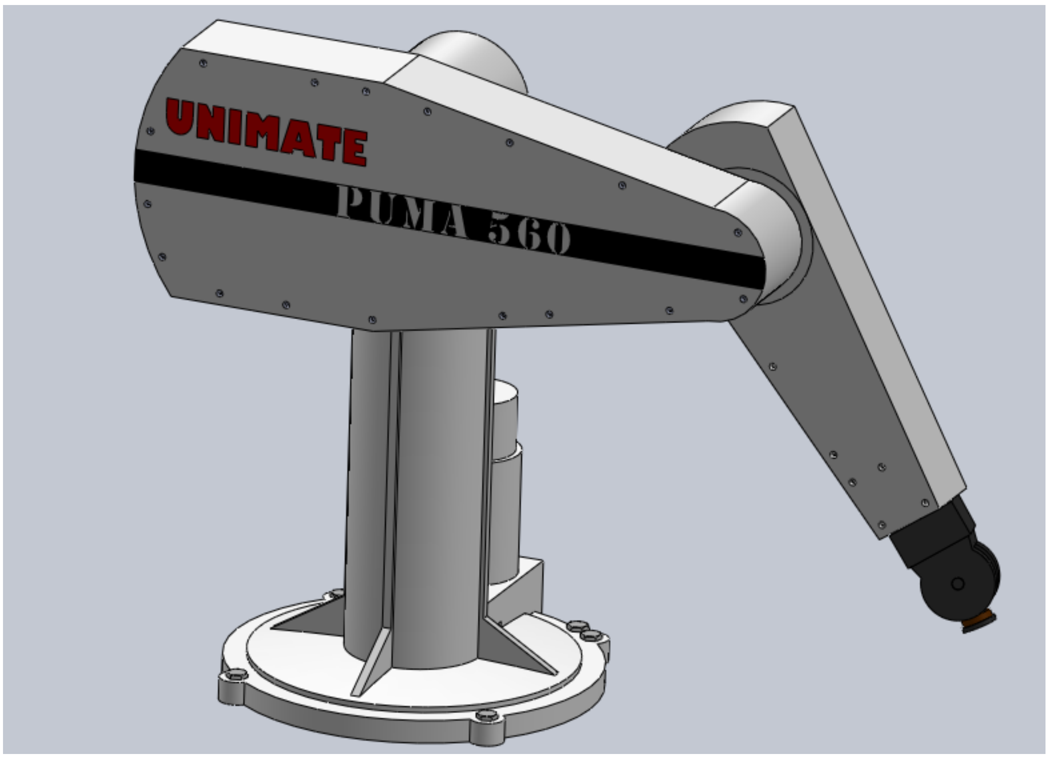


Figure 4: Puma560 robotic arm

4.3 Simulation Model

Figure 5 shows the model I created for validating my numerical results; it presents eight operational blocks, some directly obtained form software's library and others coded manually, all representing entirely the assumptions and formulae of this work.

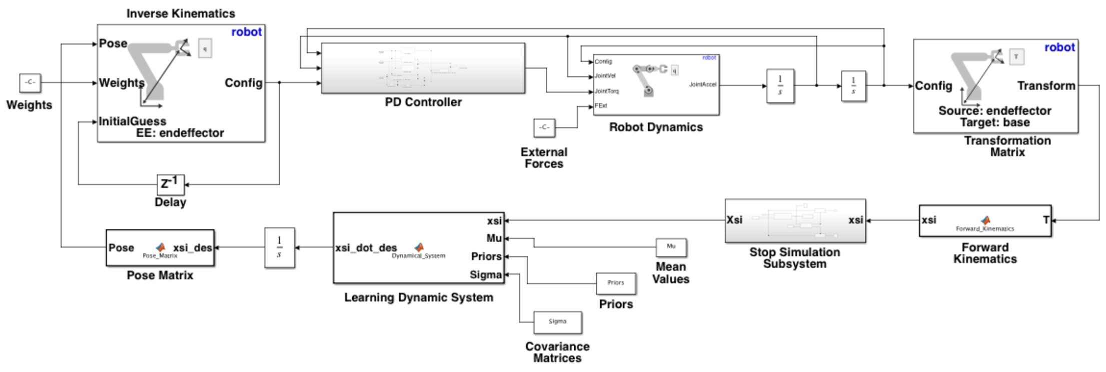


Figure 5: Simulink Model

4.3.1 Robot Dynamics

The inner loop can be easily modeled just with the Robot's Dynamics block, findable in the software library; the only alteration needed is to define the exact design of the robot arm that I want to consider for simulation, in this case the one I modeled. This block expresses the entire dynamics of the robot computing immediately the joints accelerations of all free joints, expressed in radiant per seconds square. Simply through a two step integration I can obtain also joints velocities and configurations.

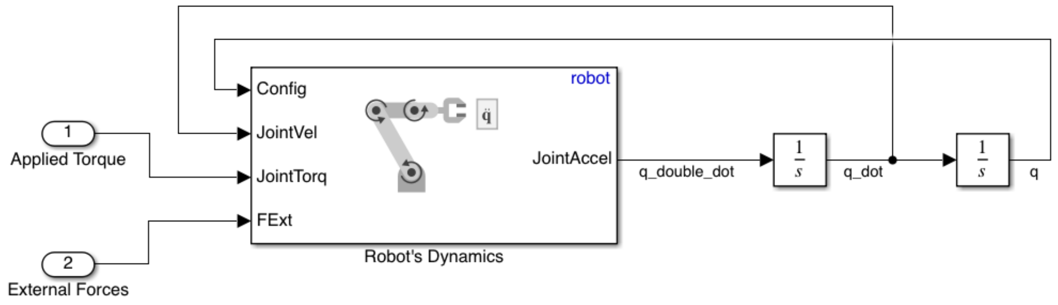


Figure 6: Robot's Dynamics Block

4.3.2 Forward Kinematics

These part represents the conversion from the joint space to the Cartesian space through the use of the Forward Kinematics; the most important step is the computation of the transformation matrix, necessary to convert coordinates. In fact, to proceed with the learning algorithms and the computation of the desired motion, I need velocities and positions expressed in Cartesian space. The transformation matrix also gives informations about the orientation of the end effector defined by the angles of roll, pitch and yaw, obtainable from the rotation part of this matrix.

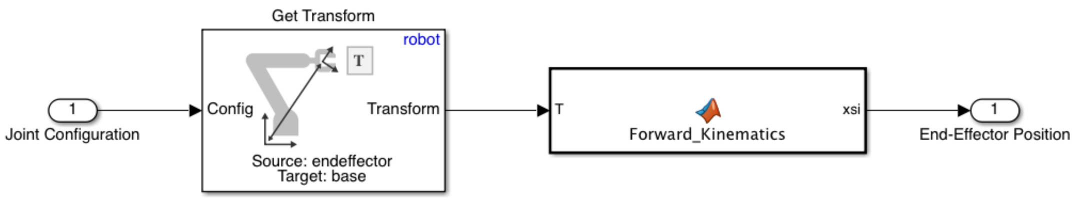


Figure 7: Forward Kinematics Block

4.3.3 Dynamic System

This is the only main operational block of the entire simulation, and the particular reason to this circumstance is that it coded all the learning algorithms and methods that has been shown in Chapter 3. Following all formulas and all constraints this block computes the desired motion velocity, considering the numerical values of parameters that I obtained before.

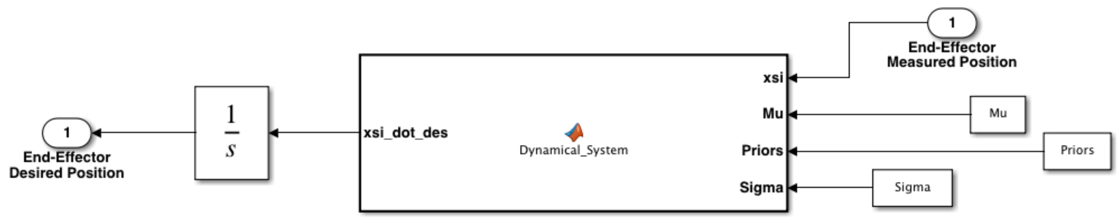


Figure 8: Dynamic System Block

4.3.4 Inverse Kinematics

This operational part is exactly the opposite of forward kinematics; while before I obtained the orientation angles and the coordinates of the end effector in the Cartesian space from the transformation matrix, now I need do recreate that matrix, but, of course, considering the new position coordinates found from previous block, the ones describing the desired motion and no more the measured one. The result of this operation is that I change again to the joint space obtaining the orientation of joints for the desired configuration.

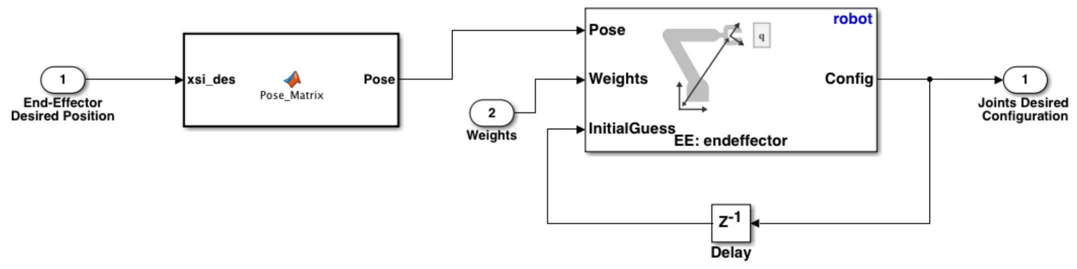


Figure 9: Inverse Kinematics Block

4.3.5 Controller

The controller subsystem is not as complex as it could seem in figure 10, the purpose of considering a controller in the model is to make a direct confrontation between the measured motion and the desired one in order to find a proper torque action to be applied to the robot and make it adjust its position as close as possible to the desired one.

The final applied torque is composed by considering a summation of multiple torques, given by three different sources: one obtained considering gravity, one considering the actual velocity of the end effector and last one obtained directly from comparison between motions. Torques given by gravity and velocity are easily computed using the corresponding block and summed together, a bit longer is the computation of the third component.

Having as inputs the desired and measured joints orientation and the measured joint velocity I can confront them directly in order to verify how much the controller need to modify the actual robot motion. First step is to compute the difference between measured and desired motions operating the following equations

$$\begin{cases} q_{err} = q_m - q_d \\ \dot{q}_{err} = \dot{q}_m - \dot{q}_d \end{cases} \tag{22}$$

where q_m and q_d refers respectively to measured and desired joints configurations and same for velocities; second step is to use an existing block to obtain

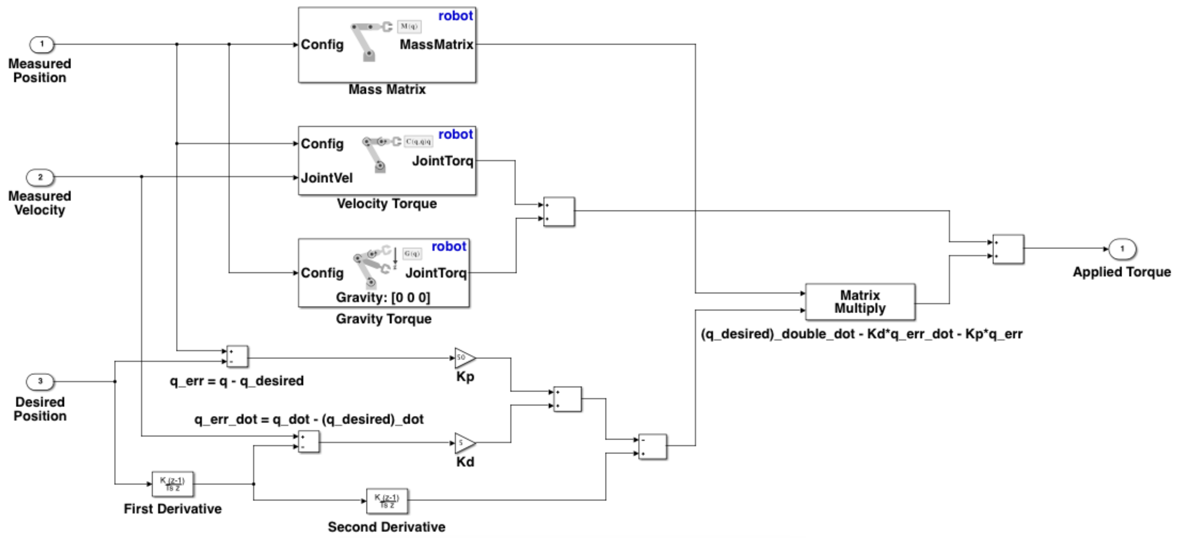


Figure 10: Controller Subsystem

system's mass matrix, that I need for final computation. In fact, to find the torque due to motion discrepancy I use the following generic controller equation

$$\ddot{q}_d - K_d\dot{q}_{err} - K_p q_{err} \tag{23}$$

where K_d and K_p are two scalar values defining the weights of the proportional and the derivative components of the controller on torque definition. Now with all three components the final torque to be applied to the robotic system is obtained by summing them together.

According to the definitions above, I selected a PD controller for this work, I decided to use this type and not a PID controller because, during computations, I observed that the system was able to follow the desired motions even without taking in consideration the integration component of the error.

4.3.6 Stop Simulation Subsystem

I left this operational subsystem as the last one and the particular reason to this circumstance is that it describes an approximation that I decided to consider in the simulation process; I use this subsystem to make a quick comparison between the actual position of the end effector and the fixed target that it has to reach. Computing the distance between them I know how far my end-effector is with respect to the desired target and because it is possible that, following the new computed trajectory, the end-effector will never reach exactly target's coordinates, I decided to set a tolerance value that is compared with this distance at each iteration. Once the end-effector reaches a position sufficiently close to target, with a distance smaller than tolerance from the computational point of view, simulation stops.

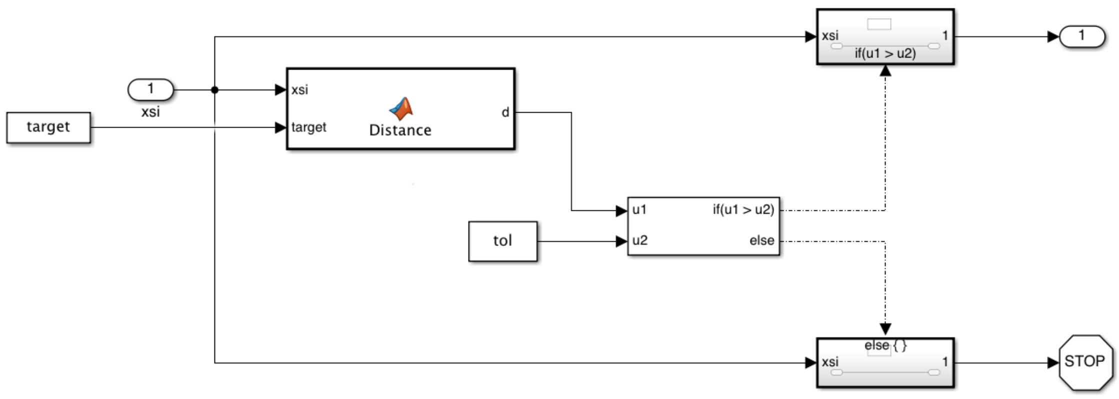


Figure 11: Stop Simulation Subsystem

5 Results

The final results that has been obtained are exactly what I was expecting; making the model run using the motion parameters obtained from first computations made the robot I designed capable of drawing new trajectories, completely different from the ones used as initial demonstrations, but reflecting some aspects of their shapes. Starting from a new random point, always inside the workspace

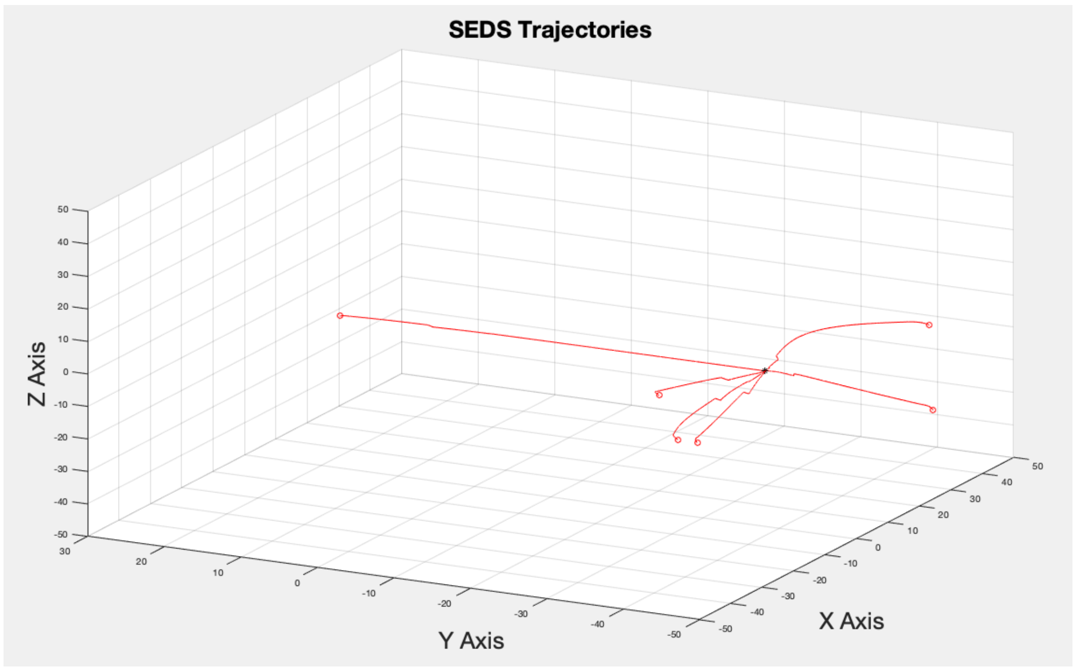
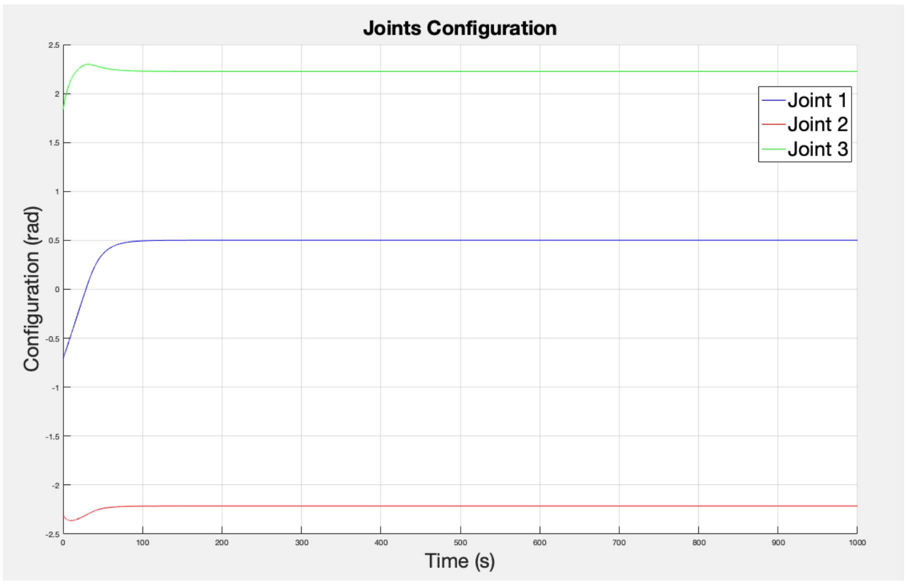


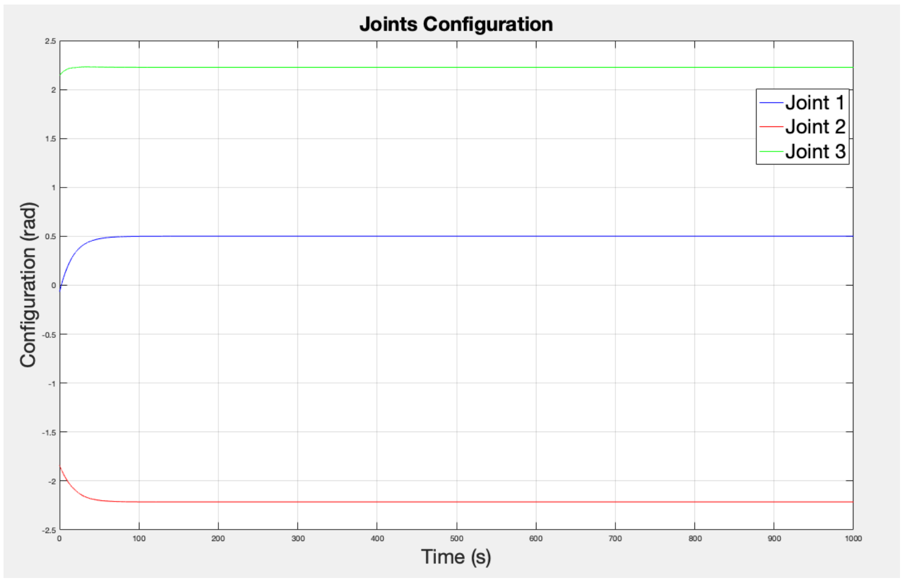
Figure 12: Trajectories computed using SEDS

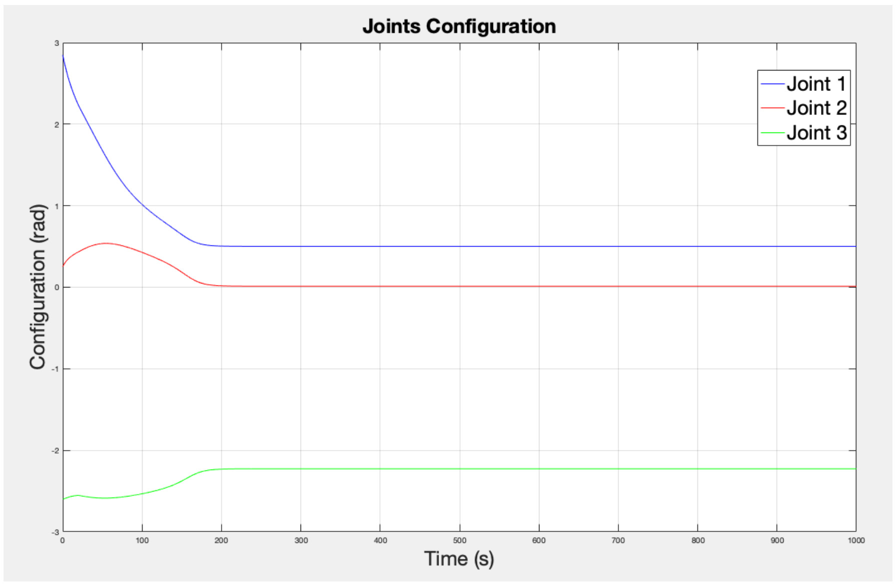
of my robot, the system is able to define a precise trajectory characterised by a path that is a mixture of the ones of the demonstration trajectories and capable to make the robot reach the same target point. In figure 12 a bunch of trajectories computed by the model is presented and, as can be seen, all of them perfectly reach the target following a path different from the others; this is the first and direct proof that the learning method I proposed is working.

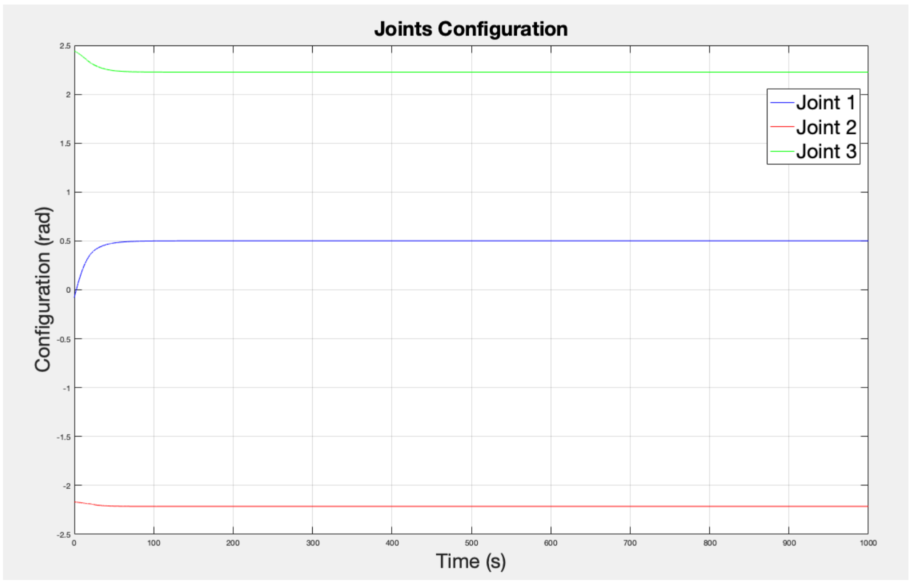
In the following figures I present some other aspects of the new computed trajectories, all significant to observe the correct functioning of the learning algorithms and of the model.

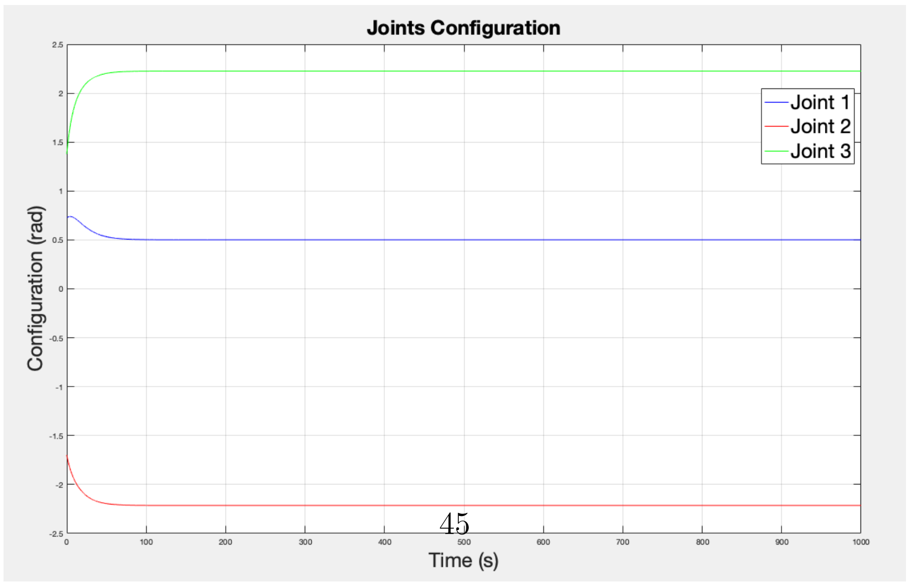
Next figures represent the behaviour of the joints configurations computed from the inverse kinematics according to time of the simulation; as can be seen all of them have a first part of settlement and then stabilize on the configuration that describes the robot's end effector positioned in the target. The graphs show also that configurations reach relatively quickly the value of stabilization and continue almost linearly, this is given by the fact that position is really close to the target but still hasn't reach the minimum distance that I set to stop the simulation.






Joints Configuration
Joint 1
Joint 2
Joint 3
Configuration (rad)
Time (s)


Joints Configuration
Joint 1
Joint 2
Joint 3
Configuration (rad)
Time (s)


Joints Configuration
Joint 1
Joint 2
Joint 3
Configuration (rad)
Time (s)

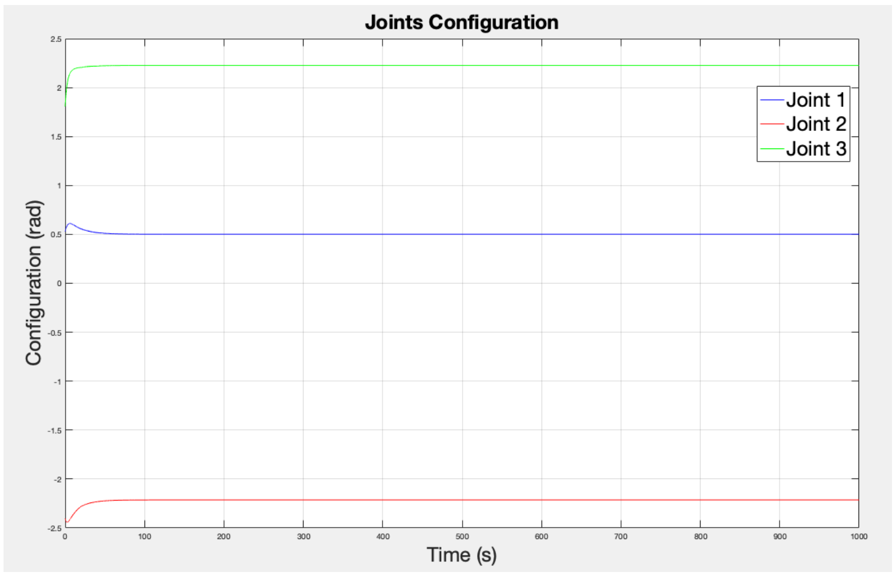
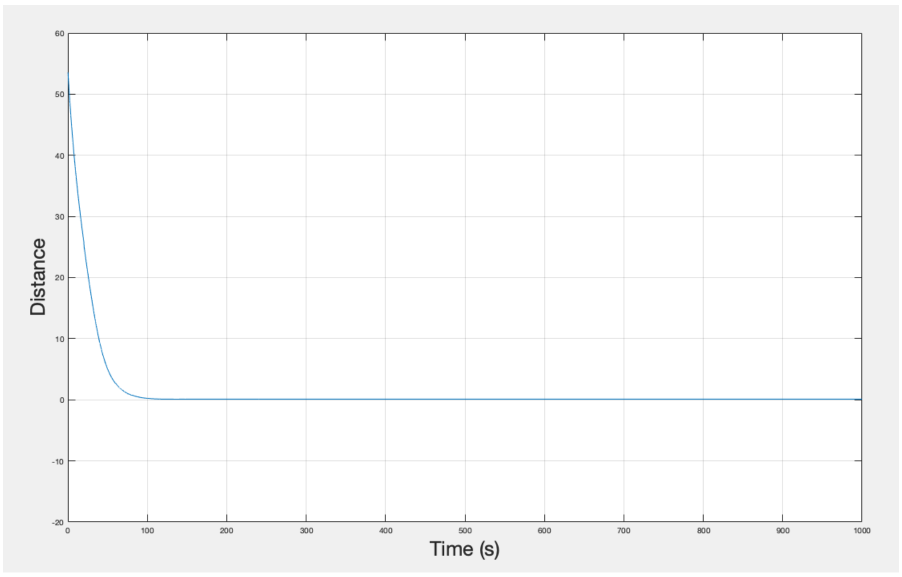
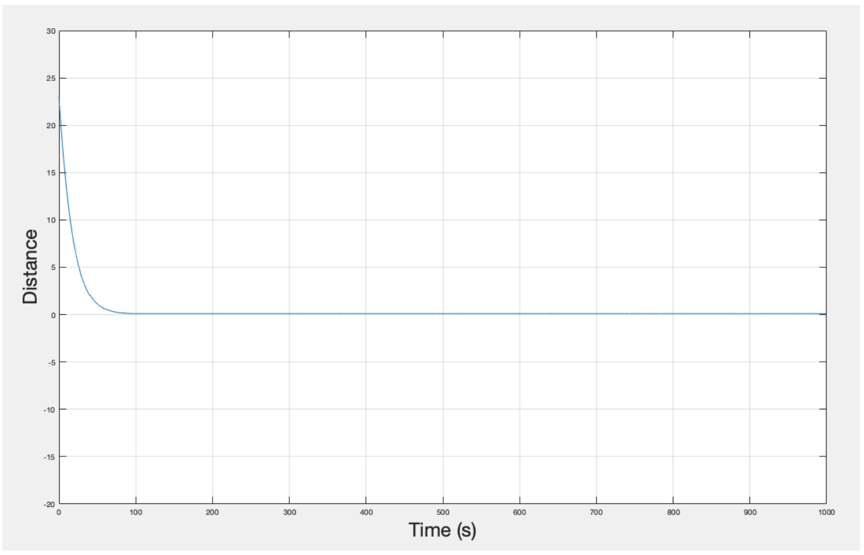


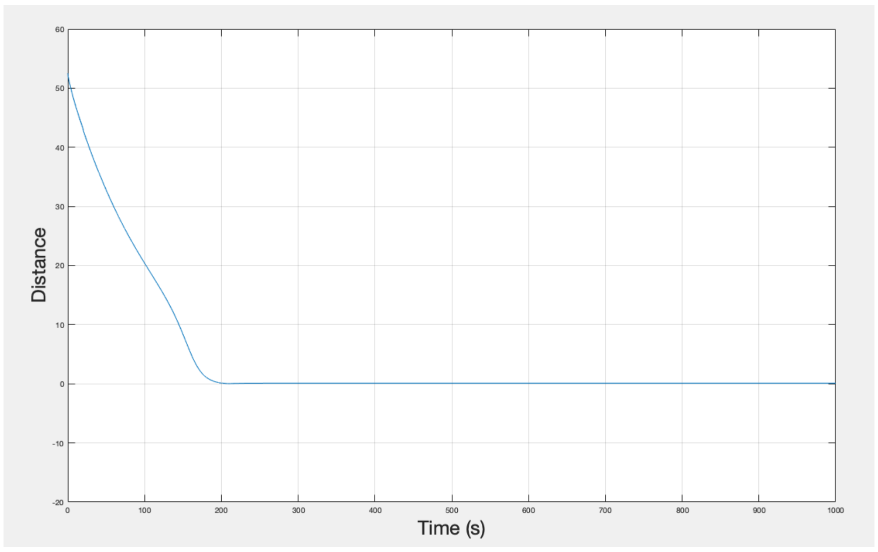
Figure 13: Trend of joints configuration according to simulation time

In figure 14 I wanted to show the behaviour of the distance between target point and the position of the end effector measured at each time iteration; as I expected the graph has a decreasing trend in time, this means that the system has been modelled correctly, capable to correct its motion in order to make the end effector follow the desired trajectory till it reaches the target.

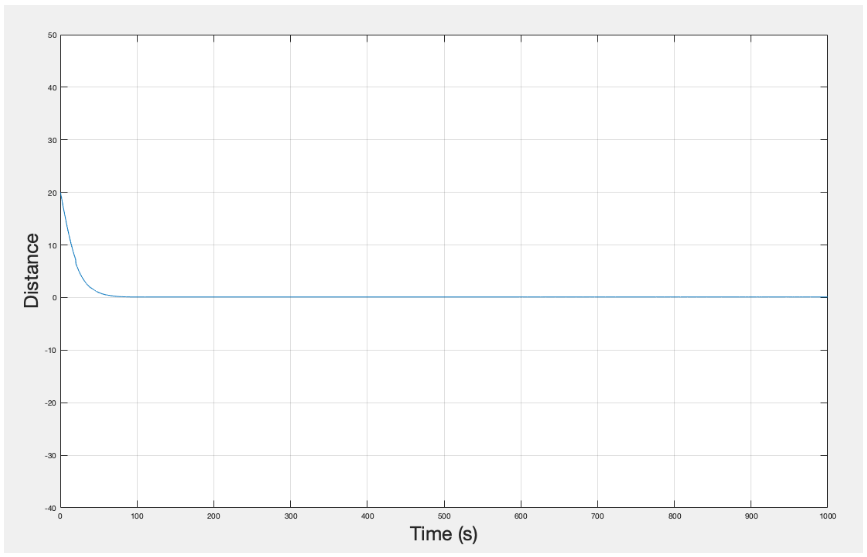




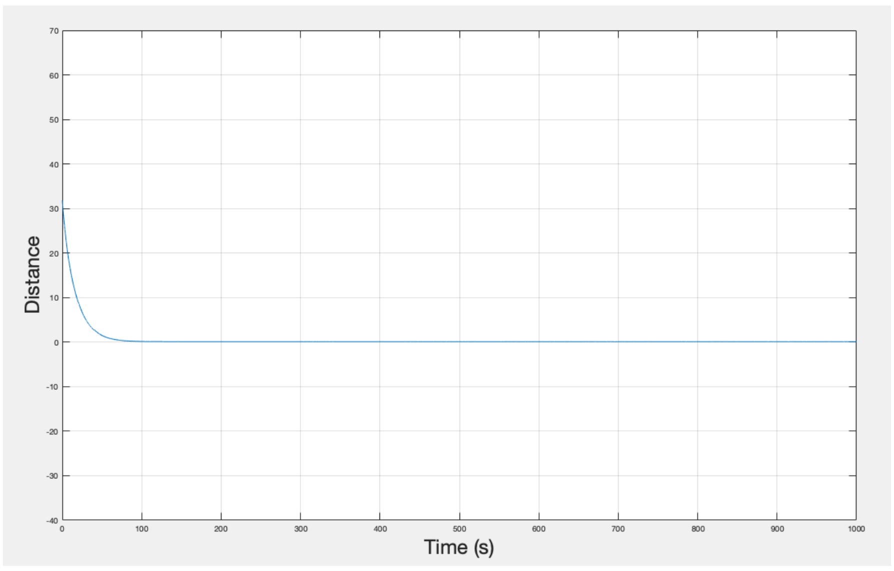
Distance
Time (s)



Distance
Time (s)



Distance
Time (s)



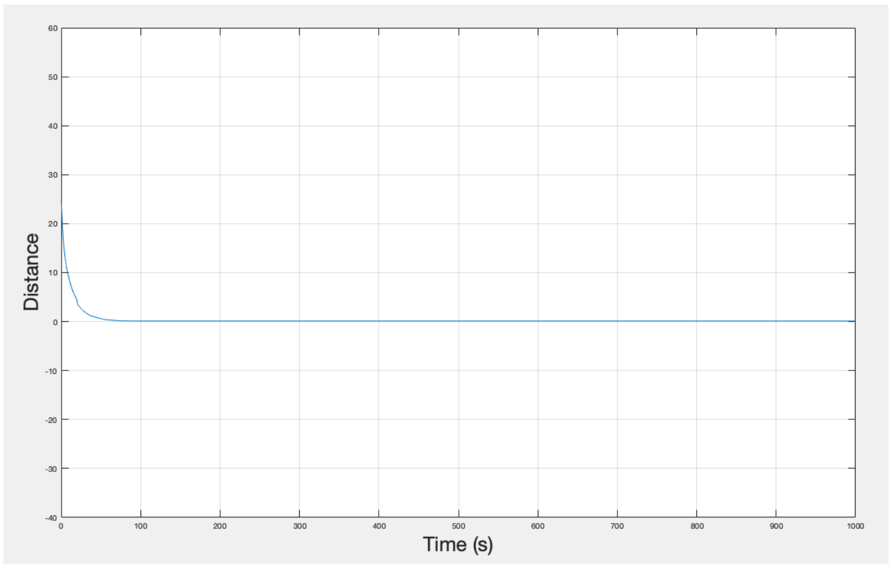
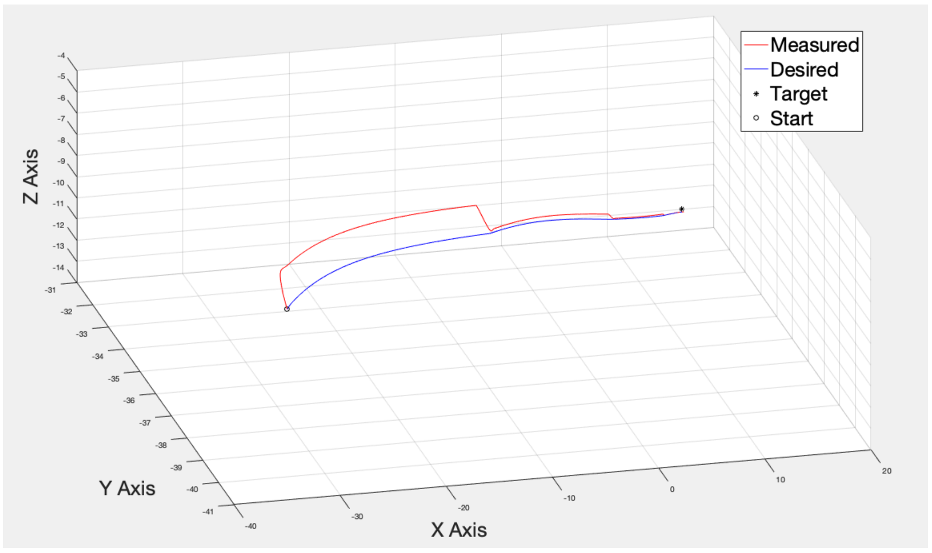
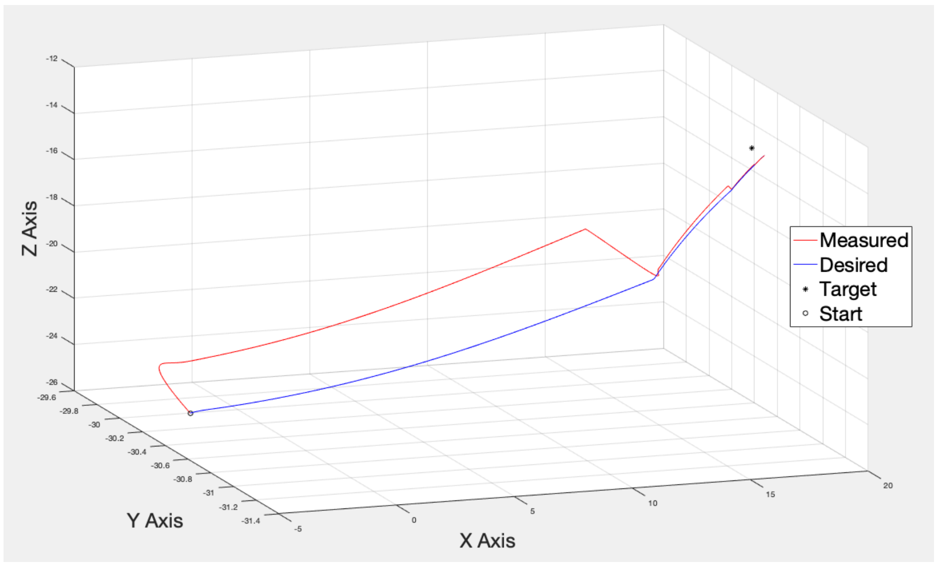
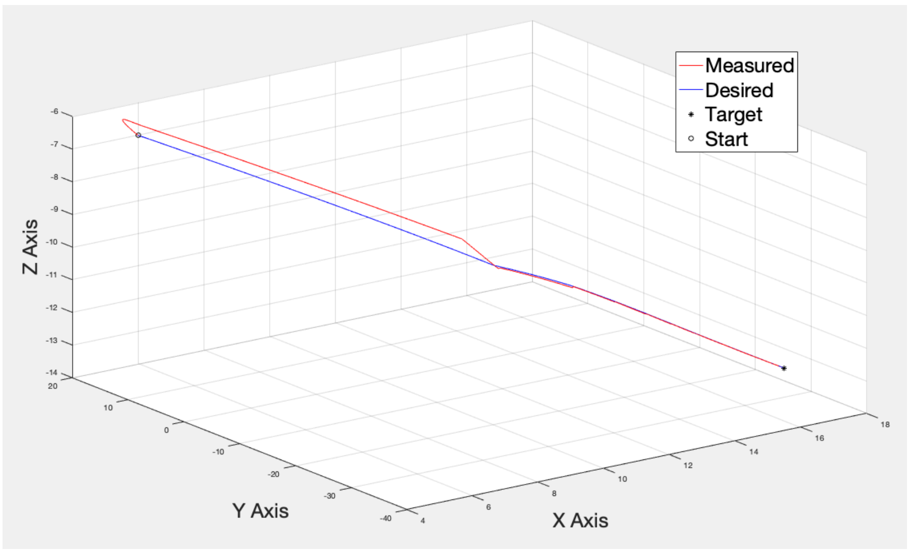


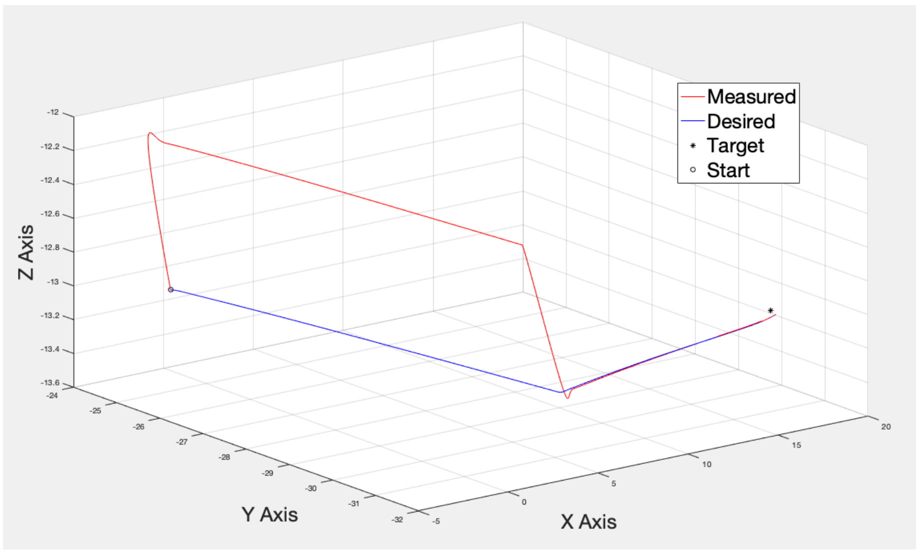
Figure 14: Distance between target and measured positions according to time

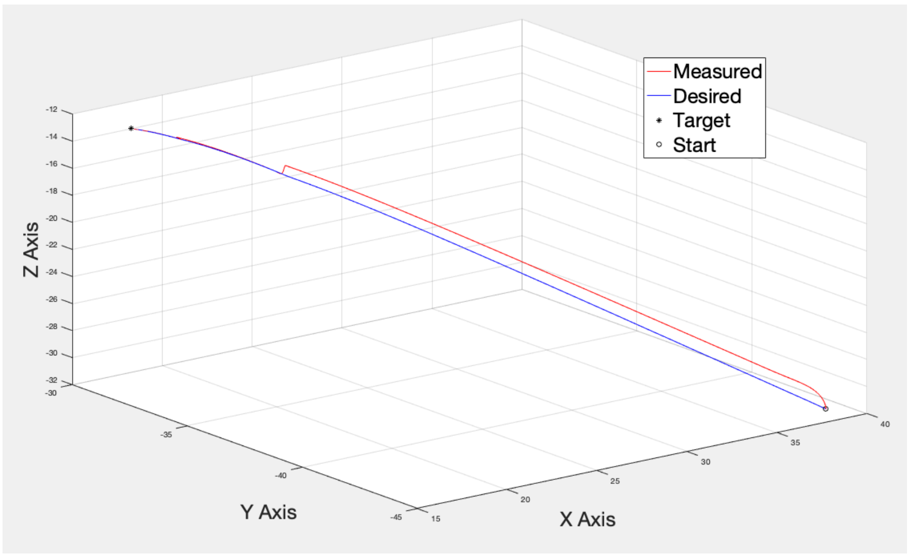
Another significant aspect that is worth seeing is the confrontation between the measured position and the desired one for each new trajectory; this comparison is useful to immediately see if the system succeeded in modeling correctly the motion or if the two trajectories are completely disconnected from each other; figure 15 shows this comparison. As shown almost all cases present an initial discrepancy between desired and measured trajectory, but the fact that they never move away drastically and that they successfully realign proves that the model adjusts its motion correctly






Measured
Desired
Target
Start
Z Axis
Y Axis
X Axis


Measured
Desired
Target
Start
Z Axis
Y Axis
X Axis


Measured
Desired
Target
Start
Z Axis
Y Axis
X Axis

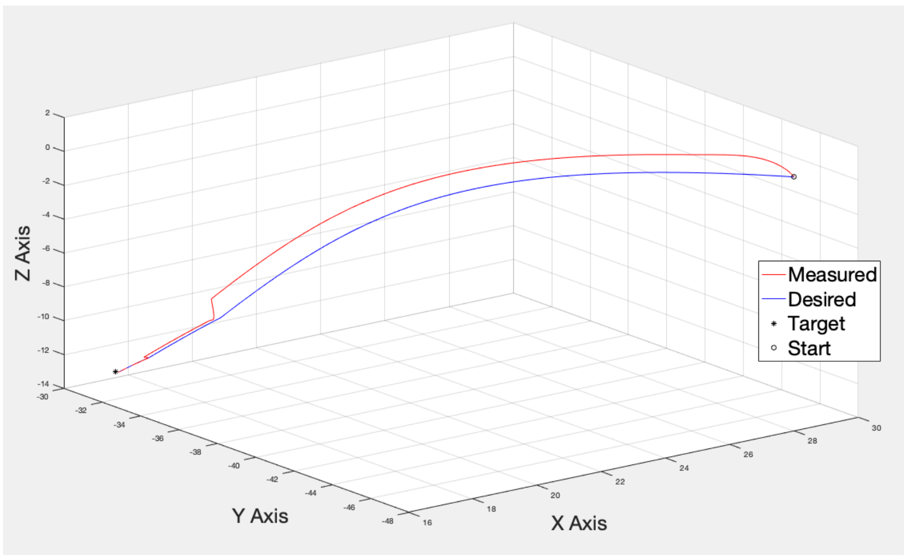


Figure 15: Comparison between measured and desired positions

6 Conclusions

In this work, I presented a method to learn arbitrary discrete motions by modeling them as a nonlinear autonomous DS, proposing a method called SEDS to learn the time invariant parameters of a GMM by solving an optimization problem under strict stability constraint and considering two possible objective functions for this optimization problem: the MSE and Likelihood.

Both models benefit from the inherent characteristics of autonomous DS, like online adaptation to both temporal and spatial perturbations; however, each objective function has its own advantages and disadvantages. As already said in 3.2.2 and in 3.2.3 I decided to use the MSE cost function because, though it's slightly more time consuming since it requires computing GMR at each iteration for all training data points, however, it requires fewer parameters than the likelihood one which makes the algorithm faster for a three or more dimension case or when considering a higher number of components. Considering a fixed number of Gaussian functions, the former usually results in having a more accurate model, while the latter is faster to train.

The stability conditions at the basis of SEDS are sufficient conditions to ensure global asymptotic stability of nonlinear motions when modeled with a mixture of Gaussian functions; although these global stability conditions might be too stringent to accurately model some complex motions.

Considering the work presented, choices made and according to the results obtained, I can affirm that the method presented is valid to comply the initial requests, that it is sufficiently efficient during numerical resolution and that it provides acceptable and adequate results for the fulfillment of the task

Appendix A

Motion parameters numerical results

Parameters	**K**					
	1	**2**	**3**	**4**	**5**	**6**
π	0,2250	0,3295	0,2186	0,1211	1,5674e-13	0,1019
μ	13.8726	-1.8045	11.2366	-20.3471	70.6065	52.2679
	-34.0091	-14.6182	0.4762	-13.1531	19.7984	-31.1325
	-29.2338	6.1256	-13.2506	-30.0170	43.4873	3.4361
	0.1602	0.5835	0.4470	1.6185	-13.2094	-2.7911
	0.4248	-0.1029	-0.2712	-1.7373	-2.0262	0.0436
	1.1226	-0.3158	0.0176	0.0021	-12.5010	-1.4673

Table 3: Numerical results of Priors and Mean Values

Parameter	**K**					
	1					
Σ	412.279	2.692e-06	-2.325e-06	-20.8497	0	0
	-3.1475e-06	74.789	3.947e-05	0	-10.744	0
	-9.1524e-06	-1.1695e-05	78.655	0	0	-5.432
	-20.850	8.1576e-05	-8.629e-06	2.9022	0	0
	-1.2717e-05	-10.744	1.9934e-04	0	4.388	0
	-1.0597e-04	-3.2557e-04	-5.4322	0	0	1.151
	2					
	152.590	5.409e-06	6.4407e-06	-4.673	0	0
	2.5697e-06	105.7727	-3.815e-06	0	-0.595	0
	5.5244e-06	-8.4086e-06	154.7917	0	0	-2.482
	-4.6767	8.532e-05	9.3106e-05	88.6928	0	0
	-4.2505e-05	-0.606	9.7254e-05	0	115.805	0
	1.312e-04	-2.5445e-04	-2.488	0	0	0.248
	3					
	49.752	-9.2673e-06	-1.9665e-05	-3.846	0	0
	1.2665e-05	380.652	4.129e-06	0	-3.132	0
	-1.3463e-05	7.3582e-06	126.9904	0	0	-9.527
	-3.8468	-2.4985e-04	-9.2155e-05	150.1814	0	0
	3.1622e-04	-3.1352	2.0002e-04	0	3.6596	0
	-1.4257e-05	6.428e-05	-9.5275	0	0	0.9525
	4					
	342.4765	9.8079e-07	9.9456e-06	-14.8397	0	0
	1.0753e-06	244.8652	2.0432e-05	0	-23.794	0
	1.5586e-06	1.7576e-06	96.3332	0	0	-3.2e-15
	-14.8403	-1.017e-04	1.9354e-04	339.366	0	0
	-4.443e-05	-23.7944	1.928e-04	0	501.5464	0
	-8.6544e-05	9.8592e-05	-0.0135	0	0	1.0e-05
	5					
	110.964	1.1192e-05	4.2247e-05	-27.3883	0	0
	-7.5304e-07	364.167	-2.3812e-05	0	-14.486	0
	-5.704e-05	1.638e-05	57.6426	0	0	-12.770
	-27.3884	1.5776e-05	7.538e-05	8.6373	0	0
	-2.0202e-04	-14.4871	2.3997e-04	0	1.022	0
	-3.6406e-04	2.2454e-04	-12.7699	0	0	3.513
	6					
	347.6965	6.722e-05	2.8747e-05	-27.4935	0	0
	1.0645e-05	42.6105	4.1762e-05	0	-14.851	0
	1.325e-05	7.238e-05	143.347	0	0	-12.775
	-27.4935	4.632e-04	1.7561e-04	2.7761	0	0
	-5.7405e-05	-14.851	-7.1024e-05	0	9.003	0
	-1.8782e-05	6.6171e-05	-12.775	0	0	1.415

Table 4: Numerical results of Covariance Matrices

Appendix B

Transformation matrix

The transformation matrix obtained from the Forward Kinematics has the following aspect

$$T_N = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where n, o and a are unit vectors identifying the orientations of the end effector, and respectively defining direction of x, y and z; while p is a vector that defines the location of the origin in an hypothetical n^{th} coordinate system (in this work it defines directly position of the end effector in Cartesian space).

In the case of IK, I already know the end effector coordinates, and knowing how the transformation matrix is composed I can easily compute the angles that describes joints orientation. Considering a polar robot, I have

$$T = \begin{bmatrix} c(\theta_1)c(\theta_2) & s(\theta_1) & c(\theta_1)s(\theta_2) & dc(\theta_1)s(\theta_2) \\ s(\theta_1)c(\theta_2) & -c(\theta_1) & s(\theta_1)s(\theta_2) & ds(\theta_1)s(\theta_2) \\ s(\theta_2) & 0 & -c(\theta_2) & -dc(\theta_2) \\ 0 & 0 & 0 & 1 \end{bmatrix} = \begin{bmatrix} n_x & o_x & a_x & p_x \\ n_y & o_y & a_y & p_y \\ n_z & o_z & a_z & p_z \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

with c and s that are respectively cosines and sines of joint angles θ_1 and θ_2 and d is the distance between the two joints. From this equation I just consider the following linear system

$$\begin{cases} d\ cos(\theta_1)\ sin(\theta_2) = p_x \\ d\ sin(\theta_1)\ sin(\theta_2) = p_y \\ -d\ cos(\theta_2) \quad = p_z \end{cases}$$

and I solve it finding θ_1 and θ_2, the angles that defines the end effector position in joints space.

References

[1] Khansari-Zadeh Mohammad S., Billard Aude, *Imitation Learning of Globally Stable Non-Linear Point-to-Point Robot Motions using Nonlinear Programming*, in *"2010 IEEE/RSJ International Conference on Intelligent Robots and Systems"*, October 2010.
DOI: 10.1109/IROS.2010.5651259

[2] Khansari-Zadeh Mohammad S., Billard Aude. *Learning Stable Nonlinear Dynamical Systems With Gaussian Mixture Models*, in *"IEEE Transactions on Robotics"*, Volume XXVII, **n.5**, October 2011.
DOI: 10.1109/TRO.2011.2159412

[3] Salehian Seyed Sina Mirrazavi, Khoramshahi Mahdi, Billard Aude. *A Dynamical System Approach for Softly Catching a Flying Object: Theory and Experiment*, in *"IEEE Transactions on Robotics"*, Volume XXXII, **n.2**, April 2016. **DOI: 10.1109/TRO.2016.2536749**

[4] Kim Seungsu, Shukla Ashwini, Billard Aude. *Catching Objects in Flight*, in *"IEEE Transactions on Robotics"*, Volume XXX, **n.5**, October 2014.
DOI: 10.1109/TRO.2014.2316022

[5] Al-Qahtani H.M, Amin A. Mohammed, Sunar M.. *Dynamics and Control of a Robotic Arm Having Four Links*, in *"Arabian Journal for Science and Engineering"*, **n.42**, October 2016.
DOI: 10.1007/s13369-016-2324-y

[6] Coates Adam, Abbeel Pieter, Ng Andrew Y.. *Learning for Control from Multiple Demonstrations*, in *"Proceedings of the* 25th *International Conference on Machine Learning"*, Helsinki, Finland, 2008.

[7] Dempster A. P., Laird N. M., Rubin D. B., *Maximum Likelihood from Incomplete Data via the EM Algorithm*, in *"Journal of the Royal Statistical Society"*, Series B (Methodological), Volume XXXIX, **n.1**, 1977

[8] Khansari-Zadeh Seyed Mohammad, Billard Aude, *SEDS:a Framework to Generate Stable, Adaptive, Reactive, and Human-Like Robot Reaching Motions*, January 2013

[9] Hein Helle, Lepik Ülo, *Learning Trajectories of Dynamical Systems*, in *Mathematical Modelling and Analysis*, Volume XVII, **n.4**, September 2012. **DOI: 10.3846/13926292.2012.706654**

[10] Spong Mark W., Hutchinson Seth, Vidyasagar M.. *Robot Dynamics and Control*, Second Edition, January 2004.

[11] Bazaraa Mokhtar S., Sherali Hanif D., Shetty C.M.. *NonLinear Programming: Theory and Algorithms*, Third Edition, New Jersey, John Wiley & Sons Inc., 2006

[12] Siciliano Bruno, Khatib Oussama. *Springer Handbook of Robotics*, Springer-Verlag Berlin Heidelberg, 2008

[13] Siciliano Bruno, Villani Luigi. *Robot Force Control*, New York, Springer Science+Business Media, LLC, 1999

[14] Siciliano Bruno, Sciavicco Lorenzo, Villani Luigi, Oriolo Giuseppe. *Robotics Modelling, Planning and Control*, Springer-Verlag London Limited, 2009. **DOI: 10.1007/978-1-84628-642-1**

[15] Siciliano Bruno, Sciavicco Lorenzo. *Modelling and Control of Robot Manipulators*, Second Edition, Springer-Verlag London Limited, 2000. **DOI: 10.1007/978-1-4471-0449-0**

[16] Siciliano Bruno, Valavanis Kimon P.. *Control Problems in Robotics and Automation*, Springer-Verlag London Limited, 1998

[17] Siciliano Bruno, Khatib Oussama, Groen Frans. *Experimental Robotics VIII*, Springer-Verlag Berlin Heidelberg, 2003

[18] Streit Roy L. *Poisson Point Processes:Imaging, Tracking and Sensing*, Springer New York Dordrecht Heidelberg London, 2010. **DOI: 10.1007/978-1-4419-6923-1**

[19] Guang Sung Hsi, *Gaussian Mixture Regression and Classification*, Thesis for *Rice University*, May 2004

[20] Valli Monica, *Nonlinear Estimation and Filtering for Space Applications*, Doctoral Dissertation for *Politecnico di Milano*, 2013

[21] SEDS Algorithms exposed in [2]. https://bitbucket.org/khansari/seds/src/master/

[22] https://towardsdatascience.com/gaussian-mixture-models-explained-6986aaf5a95

[23] https://robotacademy.net.au/masterclass/robot-joint-control/