ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

Task-adaptation for assistive robotics using switching dynamical systems

---

Semester project

Spring 2017

Student:
Thomas Triquet
Section Microtechnique

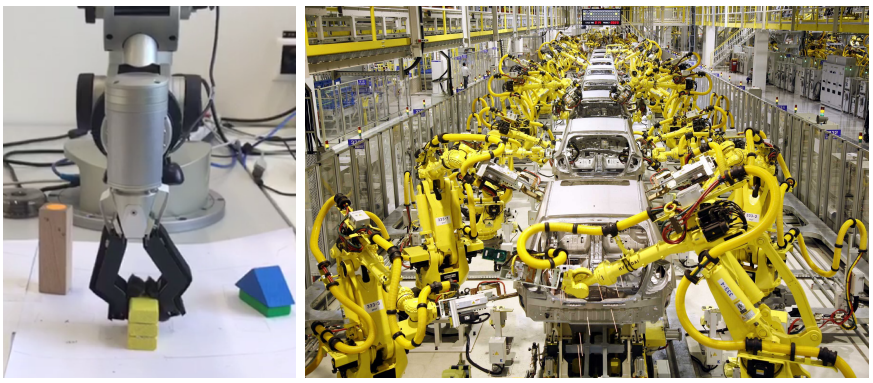
Supervisors:
Mahdi Khoramshahi
Prof. Billard Aude

Contents

1 Introduction

1.1 Compliant human-robot interaction for industrial settings

Today, the robotic domain is growing considerably due to the development of technologies used to build robots. Indeed, they become useful in many different fields ranging from small businesses to large companies where robots perform simple tasks such as pick-and-place. These type of motions are used to automatize the production lines in some industry such as the car industry. They are repetitive tasks that consist in doing simple manipulations on objects such as painting, brushing, welding, etc. However, it leads to a production line where each robot is dedicated to perform only one task. The drawback is that these lines are not flexible and also not human-friendly. Indeed, as they do fast movements in order to boost their efficiency, if an obstacle is on their path they will hit it and/or stop immediately. They will not interact with them. An example of pick-and-place motion is shown in figure 1 with a production line of Hyundai[1].



(a) Pick-and-place (b) Exemple of a production line of Hyundai

Figure 1: Examples of pick-and-place motions done by robots used in the industry

In some other applications, it is desirable to have robots work in coordination with the humans. To do so, they are controlled compliantly so users can interact with them. An example is shown figure 2[2] (the human is working with a robot arm). In some applications it would be interesting if a robot can understand a human intention and then adapt its motion accordingly. One can thinks at home applications or assistive robotics.

Figure 2: Example of a compliant robot

[1]Hyundai, URL : http://imgur.com/CF8tEct (08/06/2017)
[2]Picture by Universal Robots. URL : https://www.entrepreneur.com/slideshow/274355 (08/06/2017)

1.2 Dynamical systems as active motion generators

It exists mathematical operators, called dynamical systems (D.S.) that map the position of an object (such as robotic joints or the end-effectors) to its velocity:

$$\dot{\overrightarrow{x}} = f(\overrightarrow{x}) \tag{1}$$

They can be used as motion generators thanks to certain important properties. First, the motions that they generate are stable. A given task (at the level of kinematics/motion) can be modeled by such dynamical systems. They can be fitted to a set of demonstration [1] (i.e., learning from demonstrations). Then they generalize to unseen contexts. They also provide active motion planning. It means that if something disturbs the robot and moves it, a similar motion to those demonstrated will be generated by the dynamical system. This property (i.e., active motion generation) is illustrated in figure 3.

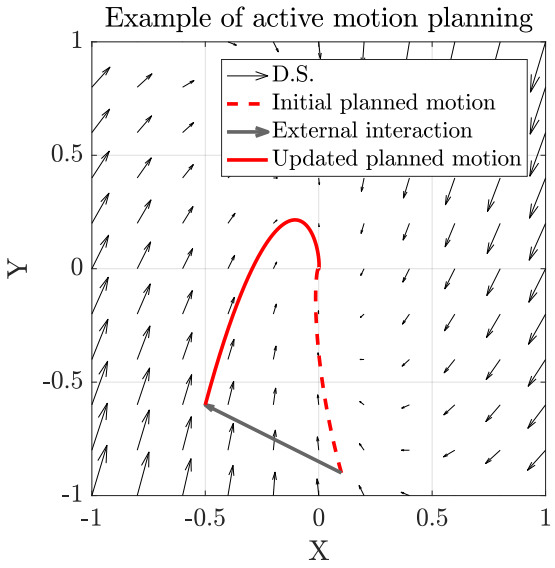


Figure 3: A simple dynamical system as an active motion generator. The robot was initially performing the dashed-line. Due to a perturbation, the position of the robot changes, and a new trajectory is suggested by the dynamical system to converge to the attractor.

In order to obtain a compliant control using dynamical systems (as active motion generators), the control architecture [2] of the system can be defined as follows (figure 4):

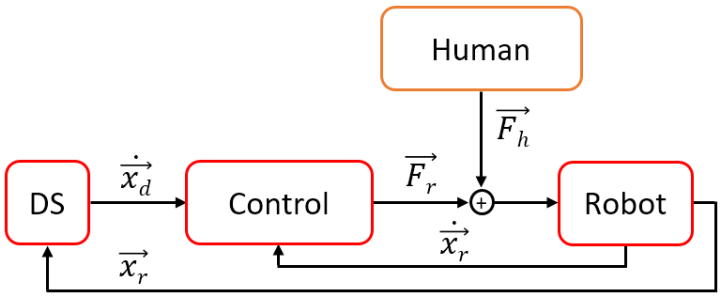


Figure 4: Architecture used to have a DS as motion generator. The DS takes the position of the robot in entry and gives a desired velocity to the control block. Then the controller tries to make the velocity wanted by the DS being the same as the one of the robot. It applies a resulting force to the robot at which the external forces (such as the human perturbation) are added.

1.3 Problem statement

Since the human robot interaction is compliant and the human can safely perturb the motion of the robot, it would of particular benefit for the human-user to use this compliant behavior to show his/her intention to the robot. Therefore, it is necessary for the compliant robot to understand the intention of the user. This can be formulated as a prediction problem, and can be tackled using adaptive methods.
To allow for changes in the behavior of the robot, we can think of dynamical systems that have extra degrees of freedom as to generate different types of motions. For example, next section presents simple parameters that allows for simple geometric transformations such as rotation, translation or scaling. These degrees of freedom, or "parameters", can be identified by the robot as to adapt its motion-generator to the intention of the human. This can be seen in the following architecture depicted in figure 5 where a parameterized dynamical system is used for motion-generation and the parameters can be adapted (using optimization techniques on a given cost-function).

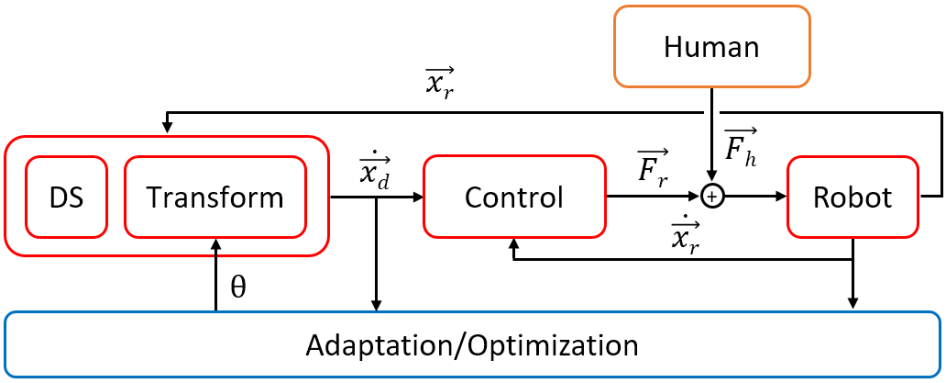


Figure 5: Architecture used to have an adaptation process. It uses blocks of the previous architecture. The transform block is new, next to the DS. It is used to transform geometrically the DS given a set of parameters θ. The adaptation block computes new parameters θ to make the velocity of the DS match the one of the human.

In comparison to the first architecture (Figure 4), there are 2 blocks that have been added here:

- Transform. Its aim is to modify the dynamical system thanks to some parameters θ. This is done in parameterizing the D.S. like it was told before.
- Adaptation/Optimization. This one regroups processes that compute the new parameters for the D.S. given the actual speed wanted by the controller of the robot and its real speed. This is equivalent to a minimization problem as we want to analyze a cost function that will be described later.

To conclude, we optimize the parameters that modify the DS dynamics, and consequently, modify the generated motions; motions that are in accordance with the intention of the human-user. In this project, we propose adaptation methods for a given parameterized DS as to adapt to the human intention.

2 Task adaptation

In this part, we first present parameterizations of dynamical systems for geometrical transformations. Then we define the cost function we study in this project.

2.1 Parameterized dynamical systems

Here we present some methods to globally modify a 2D dynamical system (it can be extend to the 3D case). Indeed, we try to find transformation matrices T that modify a D.S as we want, using the equation 2.

$$\dot{\overrightarrow{x}} = \begin{pmatrix} \dot{x} \\ \dot{y} \end{pmatrix} = T^{-1} \overrightarrow{f} \left(T \begin{pmatrix} x \\ y \end{pmatrix} \right) \tag{2}$$

2.1.1 Scaling

A scaling transformation could be useful in robotic applications that need to adapt the shape of the motion of a robot. We can take the example of a polishing robot. Let's imagine that it does a cyclic motion. It is possible that an operator wants the motion to be stretch along one axis because the surface to polish is larger for the actual piece that it works on.
The scaling (also called dilation) transformation can be formulated with a matrix presented equation 3. It is composed of two parameters a and b: the scaling coefficients along the x axis and the y axis. The result of the transformation on a dynamical system is shown in figure 6.

$$T_d = \begin{pmatrix} a & 0 \\ 0 & b \end{pmatrix} \quad and \quad T_d^{-1} = \begin{pmatrix} \frac{1}{a} & 0 \\ 0 & \frac{1}{b} \end{pmatrix} \tag{3}$$

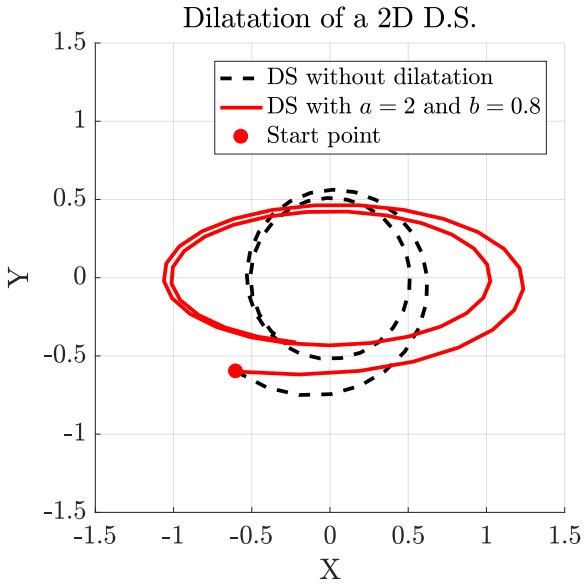


Figure 6: Scaling of a D.S.

2.1.2 Translation

A translation transformation should be interesting in many domains. Indeed, the Pick&Place motion can be done in translating an attractor from the pick position to the place position. Concerning the

polishing example discussed before, what if the surface to polish has moved to another place ? A human could then show the robot where to polish.
The mathematical operator corresponding to this geometrical transformation is the shift operator T: $T\{f(x)\} = f(x+t)$. This is not a linear transformation. By definition, a transformation matrix can only exists if the transformation is linear. Mathematicians have developped a way to do translation with matrices (it is useful in many applications and mostly in robotics). They have created the homogeneous matrix presented Equation 4 with its inverse (the aim is to add a mute coordinate). t_x and t_y are the translation along the x axis and y axis.

$$H = \begin{pmatrix} 1 & 0 & t_x \\ 0 & 1 & t_y \\ 0 & 0 & 1 \end{pmatrix} \quad and \quad H^{-1} = \begin{pmatrix} 1 & 0 & -t_x \\ 0 & 1 & -t_y \\ 0 & 0 & 1 \end{pmatrix} \tag{4}$$

In this case, the Equation 2 is modified into the Equation 5 because of the mute variable. The result of the transformation is shown in figure 7.

$$\begin{pmatrix} \overrightarrow{\dot{x}} \\ 1 \end{pmatrix} = \begin{pmatrix} \dot{x} \\ \dot{y} \\ 1 \end{pmatrix} = H^{-1} \overrightarrow{f} \left(H \begin{pmatrix} x \\ y \\ 1 \end{pmatrix} \right) \tag{5}$$

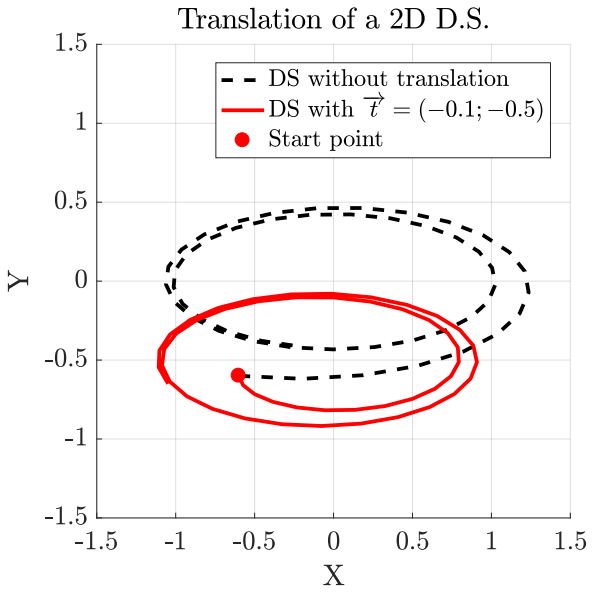


Figure 7: Translation of a D.S.

2.1.3 Rotation

The last transformation that is presented here is the rotation. This one could also have many applications. Let's take one more time the example of the polishing motion. Imagine that the piece to polish has turn. The human would like the robot to also turn its motion to polish the right way.
The rotation matrix is the most known one. It is presented with its inverse Equation 6. α is the angle of rotation. A rotated D.S. is presented Figure 8.

$$R = \begin{pmatrix} cos(\alpha) & sin(\alpha) \\ -sin(\alpha) & cos(\alpha) \end{pmatrix} \quad and \quad R^{-1} = \begin{pmatrix} cos(\alpha) & -sin(\alpha) \\ sin(\alpha) & cos(\alpha) \end{pmatrix} \tag{6}$$

Property of the transformation matrices

If we want to combine multiple transformations, we only need to multiply their matrices. Let's imagine that we want to dilate then to rotate a D.S. The transformation matrix T and its inverse are:

$$T = RD \quad and \quad T^{-1} = D^{-1}R^{-1} \tag{7}$$

It is possible to also do the same with the homogeneous matrix presented in the equation 4. To do so we need to modify the scaling or rotation matrix in adding a mute coordinate. Their equations become :

$$D = \begin{pmatrix} a & 0 & 0 \\ 0 & b & 0 \\ 0 & 0 & 1 \end{pmatrix} \quad and \quad R = \begin{pmatrix} cos(\alpha) & sin(\alpha) & 0 \\ -sin(\alpha) & cos(\alpha) & 0 \\ 0 & 0 & 1 \end{pmatrix} \tag{8}$$

Using the equations 7 and 8 we can combine the translation and rotation transformations. An example is shown figure 9.

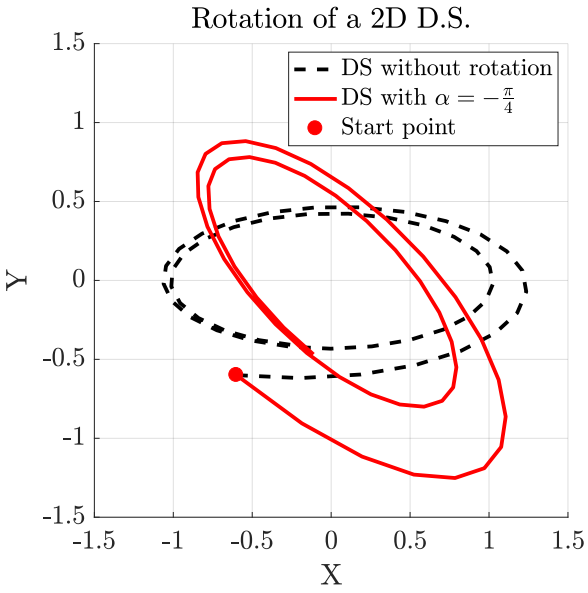


Figure 8: Rotation of a D.S.

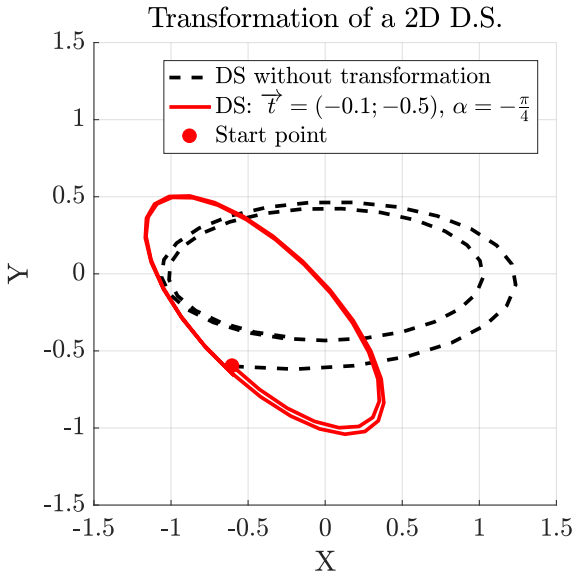


Figure 9: Rotation & Translation of a D.S.

2.2 Cost function: velocity error

In the following, we present a cost-function based on similarity between the robot and human behavior. Next, we propose a series of optimization techniques that can be efficient in minimizing this cost as to adapt the robot behavior to those from human.
Our task adaptation process will be focused on the translation and the rotation of the robot motion. We do not use the scaling process in this part. Our dynamical system can be express as follow : (θ is the set of parameters for the geometrical transformations)

$$\dot{\vec{x}_d} = \vec{f}(\vec{x}, \theta) \quad where \quad \theta = \{t_x, t_y, \alpha\} \tag{9}$$

The goal is to find a set of parameters θ_m so our velocity profile matches the human one ($\dot{\vec{x}_r}$ represents the velocity that the human is demonstrating to the robot). This can be done by minimizing the following cost function J :

$$\min_{\theta}(J(\vec{x}, \dot{\vec{x}_r}, \theta)) = J(\vec{x}, \dot{\vec{x}_r}, \theta_m) = \min_{\theta}(\frac{1}{2}(\vec{e}(\vec{x}, \dot{\vec{x}_r}, \theta))^2) = \min_{\theta}(\frac{1}{2}(\dot{\vec{x}_r} - \dot{\vec{x}_d}(\vec{x}, \theta))^2) \tag{10}$$

The cost function presented here corresponds to the squared error between the velocity shown by the human to the robot (the actual speed of the robot) and the one wanted by the dynamical system given its parameters at each time. Minimizing this function is equivalent to make the D.S. generates the same velocity that the human shows at each step, that's why it has been chosen here. There are different possible methods to minimize this function and we will present and compare some of them.

Remark : this cost is the same as what the controller tries to minimize: they both try to make $\dot{\overrightarrow{x_d}}$ and $\dot{\overrightarrow{x_r}}$ the same (one is acting on $\dot{\overrightarrow{x_d}}$ the other on $\dot{\overrightarrow{x_r}}$). As they work on different time scale, we consider this out of the scope this report.

2.3 Methods

In the following section, we present a set of methods to minimize the cost function.

2.3.1 Gradient descent

One classical way to do it is to use the gradient, thanks to this equation :

$$\Delta\theta_i = -\epsilon\frac{\partial J}{\partial\theta_i} \tag{11}$$

ϵ is the adaptation rate. It can also be declined into this formula :

$$\Delta\theta_i = -\epsilon\overrightarrow{e}^T.\frac{\partial\overrightarrow{e}}{\partial\theta_i} = -\epsilon\overrightarrow{e}^T.\frac{\partial\overrightarrow{f}(\overrightarrow{x},\theta)}{\partial\theta_i} \tag{12}$$

The equation 12 is the one we will use as the variation of the parameters is directly proportional to the gradient of the dynamical system which can be approximated easily.

2.3.2 Gradient descent with line search

A variation of the first method can be implemented. The aim here is not to move proportionally to the gradient at each step but to find the minimum on the line given by the direction of the gradient :

$$J(\theta(k+1)) = \min_{\theta} J(\theta) \quad with \quad \theta = w\nabla J + \theta(k); \quad w \in \Re \tag{13}$$

2.3.3 Stochastic search

Another possible approach of the problem can be to use a stochastic process. We can create n points randomly distributed along a normal law where the mean is the last variation of the parameters :

$$\theta_i \sim N(\overrightarrow{\mu},\Sigma) \quad with \quad \overrightarrow{\mu} = \theta(k) - \theta(k-1) \tag{14}$$

Then we can normalize them in order to project them into a sphere (we do not want to try too extreme parameters). Finally we just have to find the minimum of the cost function computed at each of these points :

$$J(\theta(k+1)) = \min_{\theta_i} J(\theta_i) \quad with \quad \theta_i = R_{sphere}\frac{\theta_i}{\max\theta_i} \tag{15}$$

A schema of a 2D example of this method is shown figure 10.

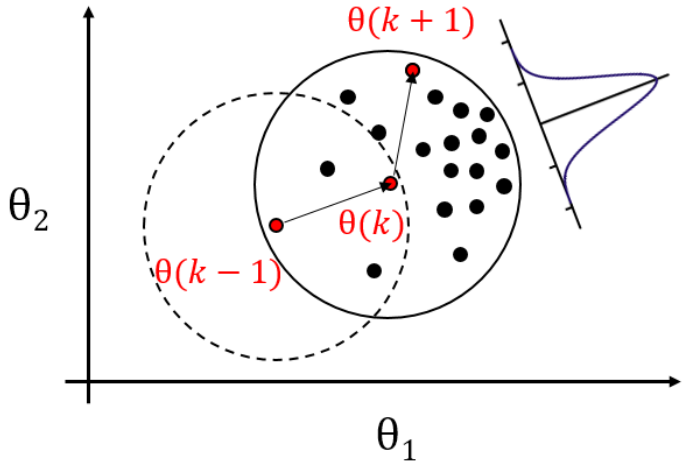


Figure 10: 2D example of the stochastic method

2.3.4 Stochastic gradient

This last method is a variation of the stochastic one. We replaced the mean of the distribution that was the last variation, by the gradient of the cost function as it is an other indication to find the minimum of the cost function :

$$\theta_i \sim N(\vec{\mu}, \Sigma) \quad with \quad \vec{\mu} = -\nabla J \tag{16}$$

2.4 Discretization

All the methods we presented were expressed in the continuous domain. In order to implement them we need to discretize them. We chose to approximate the gradient with the finite differences approximation. Indeed, we place ourself in the most general case (we do not find close form solution of the dynamical system). The gradient will be implemented thanks to the centered differences approximation :

$$\nabla J(\theta) = \frac{J(\theta + h) - J(\theta - h)}{2h} \tag{17}$$

The line search is also a continuous process that we need to discretize. To do so w (in the equation 13) take discretized values following this process :

$$w(k+1) = w(k) + \delta \tag{18}$$

δ is the step between two discretized values of w.
It means that we evaluate J at discrete points on the line shown by the gradient while the next J (evaluated at w(k) + δ) is inferior to the previous one.

2.5 Hyperparameters of each method

Now that we know how we will implement our methods, it appears interesting to do a recap of the parameters that we have to tune for each of them :

- Gradient descent : h, ϵ.
- Line search : h, δ.
- Stochastic search : R_{sphere}, Σ, n.
- Stochastic gradient : h, R_{sphere}, Σ, n.

3 Results

In this part, we present the implementations and their results. Implementation is done by numerical computation using Matlab R2016b.

3.1 Simulation setup

To test our algorithms, we need to use some specific dynamical systems. Dealing with numerical computation, we also need to consider a practical choice for the frequency of motion-generation and the frequency of the adaptation. We address these issues in the followings.

3.1.1 Dynamical systems implementations

For simple robotic application, two general types of motion can be imagined:
1/ Point to point motion (single attractors). Point-to-point motion can be generated by a simple linear dynamical system. We use this equation to create the D.S. shown figure 11 :

$$\dot{\overrightarrow{x}} = A\overrightarrow{x} \quad with \quad A = \begin{pmatrix} 1 & -1 \\ 1 & 1/2 \end{pmatrix} \tag{19}$$

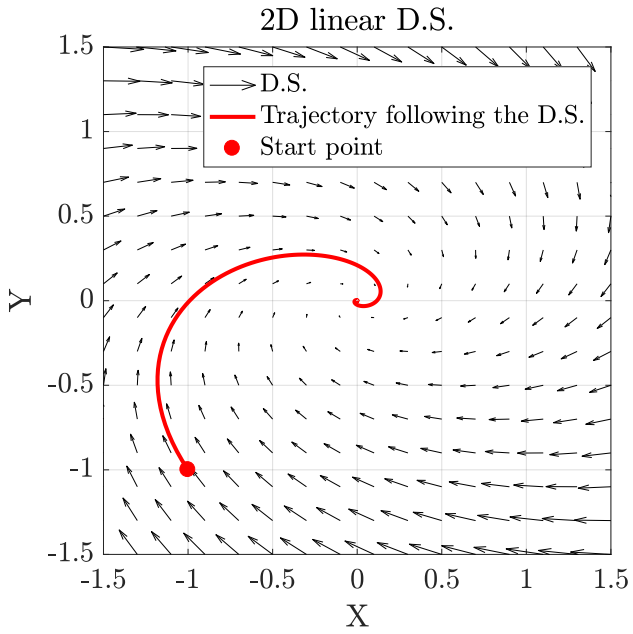
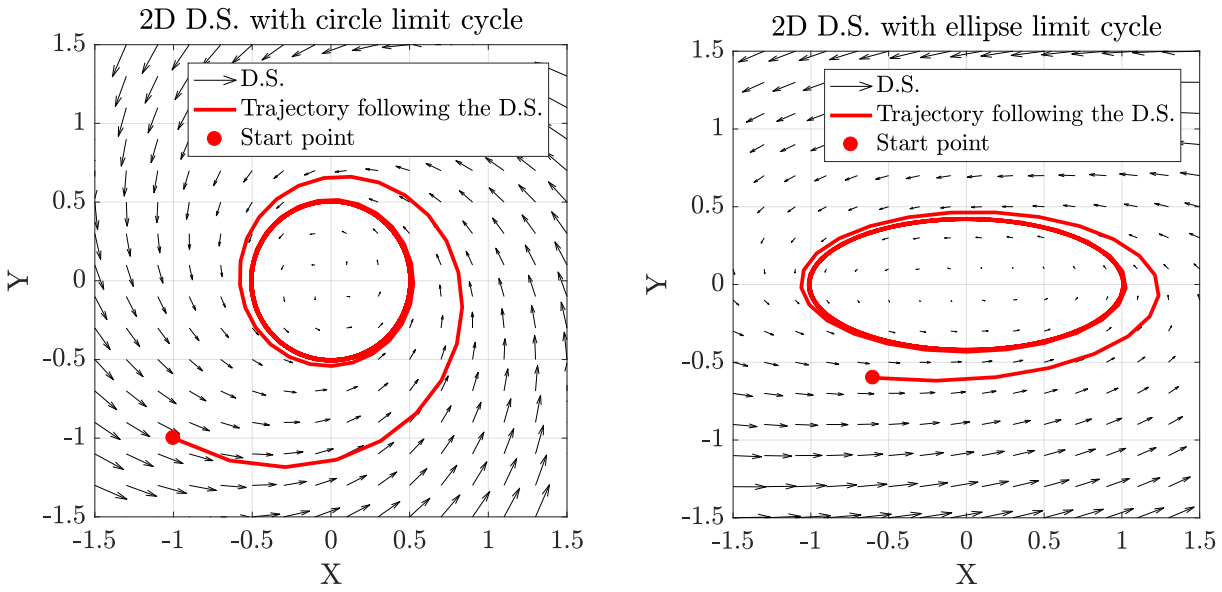


Figure 11: 2D linear D.S.

These types of dynamical systems are useful to create pick-and-place motions for example. One drawback of the linear D.S. is the fact that they can not have limit cycle which means that we cannot create periodic motions with them. However, for more complex point-to-point motion, we need to use nonlinear dynamics. For example, one can use a combination of locally linear systems as presented in [1].
2/ Cyclic motions (limit cycle). In order to have limit cycles the equation of the D.S. must be non linear. A simple DS with a limit cycle can be formulated as follows:

$$f_\theta(\overrightarrow{x}) = \begin{pmatrix} \dot{r} \\ \dot{\theta} \end{pmatrix} = \begin{pmatrix} -\alpha(r - r_0) \\ \omega \end{pmatrix} \quad where \quad r = \|\overrightarrow{x}\| \quad and \quad \theta = \arctan(y/x) \tag{20}$$

α is the radial speed (can be seen as the coefficient of the force applied to make the robot come back on the limit cycle). ω is the rotation speed.
In our case it is not really useful because we want to translate and rotate our dynamical system, and a circle has no orientation (you can rotate it with any angle it is still the same D.S.). In order to add an orientation we can modify it in dilating it thanks to the equation 3. The resulting D.S. as an ellipse as limit cycle. It is presented with the basic one in the figure 12. Its parameters are $a = 2$, $b = 1/1.2$, $\alpha = 3$, $r_0 = 0.4$ and $\omega = 5$.



(a) D.S. with a circle as limit cycle

(b) D.S. with an ellipse as limit cycle

Figure 12: Non linear dynamical systems with limit cycle

3.1.2 Frequency of adaptation

In order to implement these algorithms we need to define the frequency for the update of the parameters. Indeed, we want our simulation to be as close as possible to the real world. Using the figure 13, we choose to have a frequency $f = 100Hz$ which means that we will give 10 milliseconds at each step to our algorithms for computing the new parameters (using the command tic and toc in Matlab).

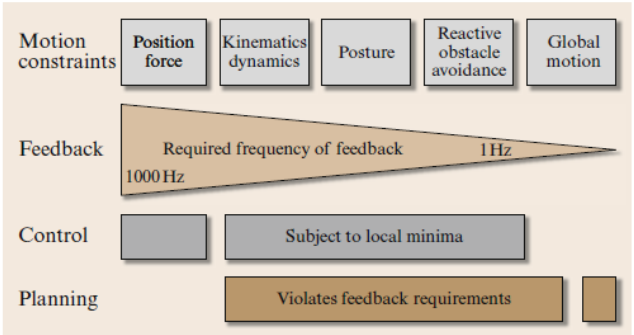


Fig. 26.3 Examples of motion constraints encountered in manipulation tasks and their feedback requirements

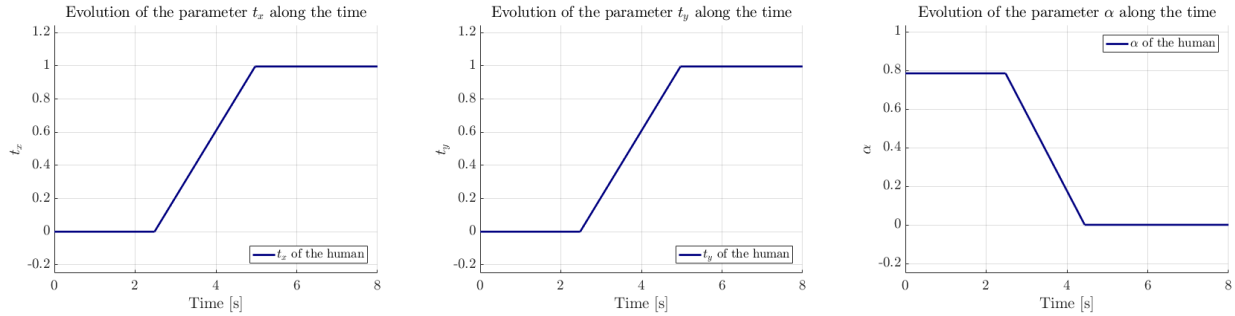
Figure 13: Based on these propositions [3], for our application in this project, we picked 100 Hz as the sampling rate for our adaptation method.

3.2 Scenarii

We study the performances of our methods in simple cases where the parameters slowly change to new values. Then we will try to be more realistic in adding some noise to the parameters.
Remark : After some manual tuning of the hyperparameters of the methods, we chose to fix them as follow : $h = 10^{-7}$, $\epsilon = 5.10^{-4}$, $\delta = 0.02$, $R_{sphere} = 0.07$, $\Sigma = 0.07I$ and $n = 150$. They could be adjusted depending on the scenario.

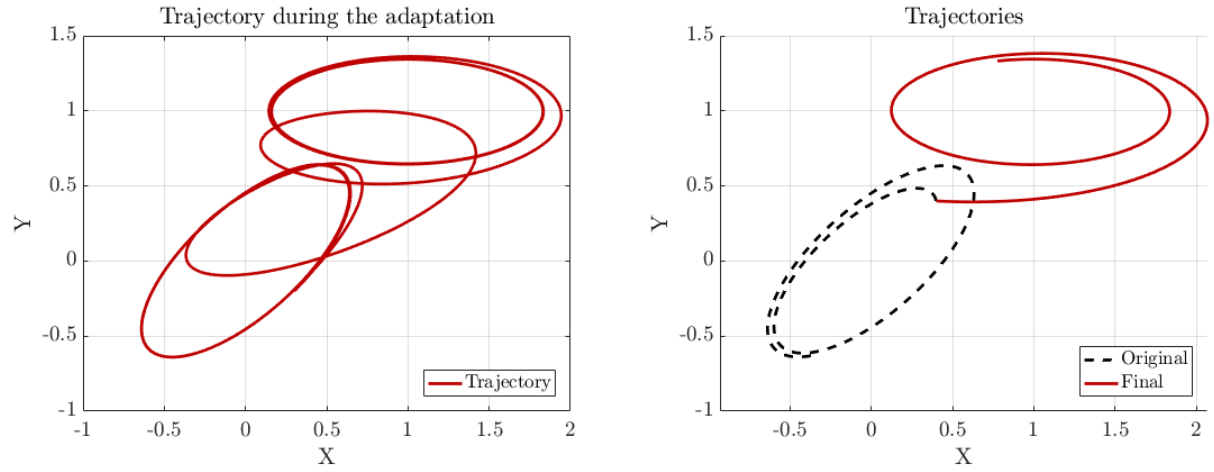
3.2.1 Human acts as a perfect dynamical system

The first assumption that we can make is the fact that the human, which is showing motions to the robot, acts as a perfect dynamical system. It means that, in the simulation, we will use our motion generator with different parameters in order to compute the real velocity of the robot $\dot{\overrightarrow{x_r}}$. An example of the parameters used over the time to create a motion like that is shown in figure 14. The corresponding trajectory followed is presented figure 15.



(a) t_x shown by the human (b) t_y shown by the human (c) α shown by the human

Figure 14: Evolution of the parameters shown by the human

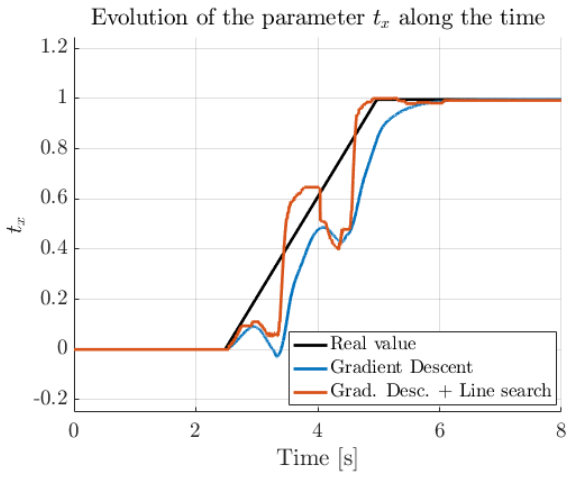


(a) Evolution of the trajectory shown (b) Original & final trajectories

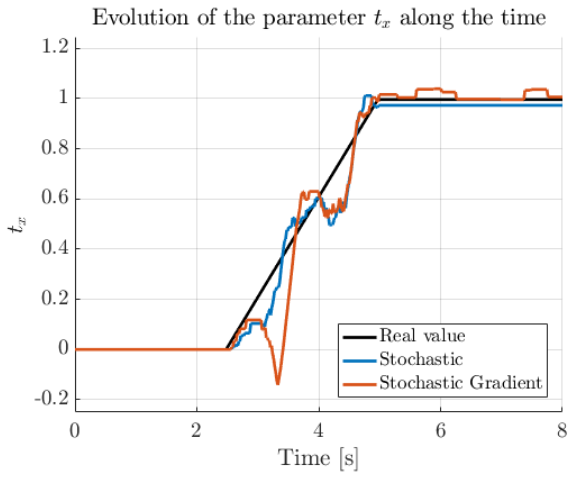
Figure 15: Trajectory shown by the human

The results of the adaptation of the parameters t_x, t_y and α are shown, respectively, in the figures 16, 17, 18. We can say that in this "perfect" case, all the methods work well. Indeed, they all converge rapidly to the parameters of the motion shown by the human. One thing interesting to mention is

the fact that, we did not choose a high value for the adaptation rate ϵ (used in the gradient descent method). This makes it converge slower even though it is still correct. We did this choice in order to see if in some scenarii, having methods that do not converge that fast works better. The evolutions of the cost function over the time as well as its integral are shown in the figures 19 for each methods. In those graphs we can clearly see the fact that the gradient descent converges slower than the others (J has larger peaks and its integral is higher at the end).

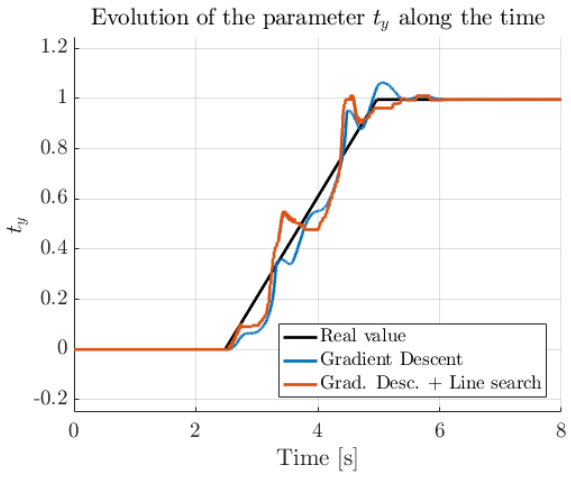


(a) Adaptation of t_x for the gradient methods

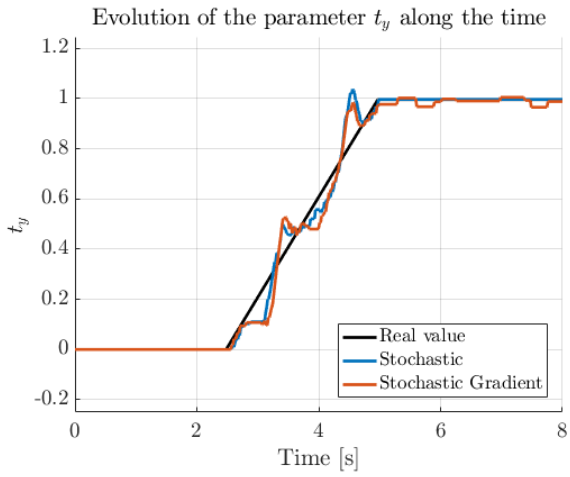


(b) Adaptation of t_x for the stochastic methods

Figure 16: Adaptation of the parameter t_x

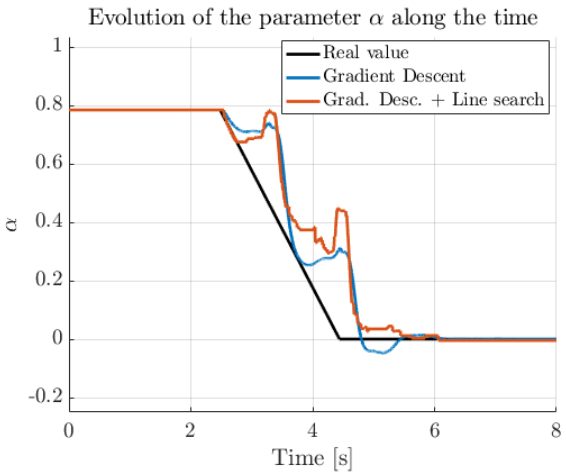


(a) Adaptation of t_y for the gradient methods

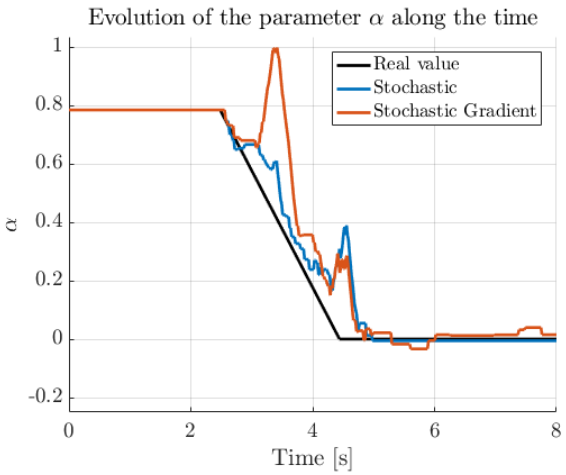


(b) Adaptation of t_y for the stochastic methods

Figure 17: Adaptation of the parameter t_y

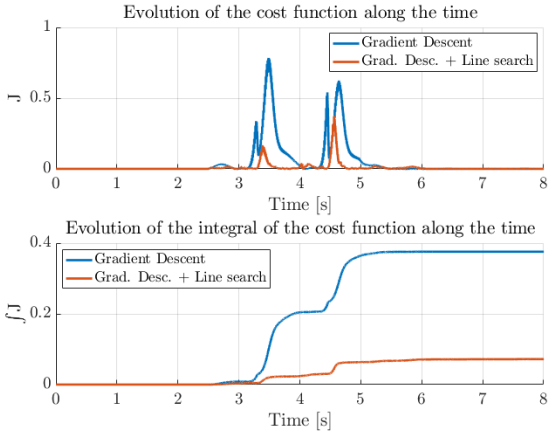


(a) Adaptation of α for the gradient methods

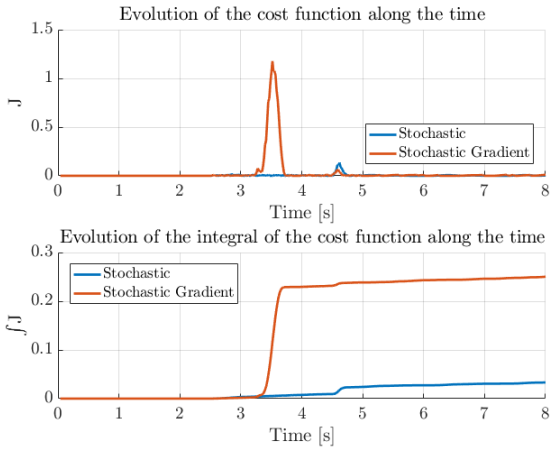


(b) Adaptation of α for the stochastic methods

Figure 18: Adaptation of the parameter α



(a) Cost function the gradient methods

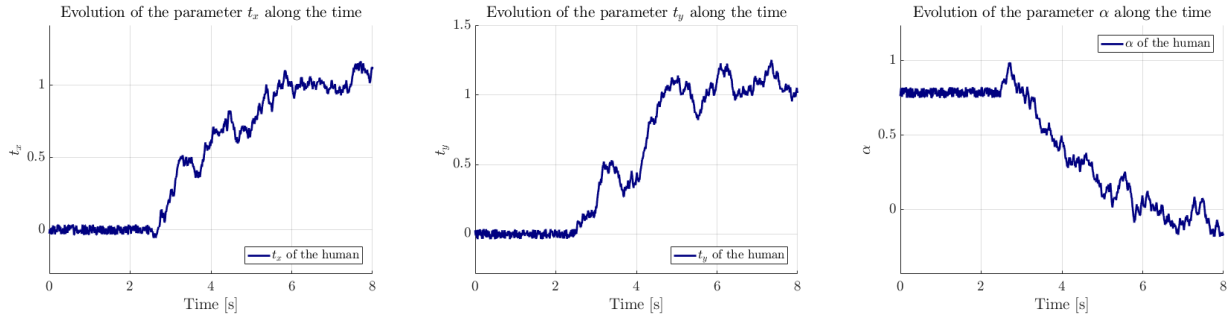


(b) Cost function of the stochastic methods

Figure 19: Evolution of the cost function and its integral of every methods

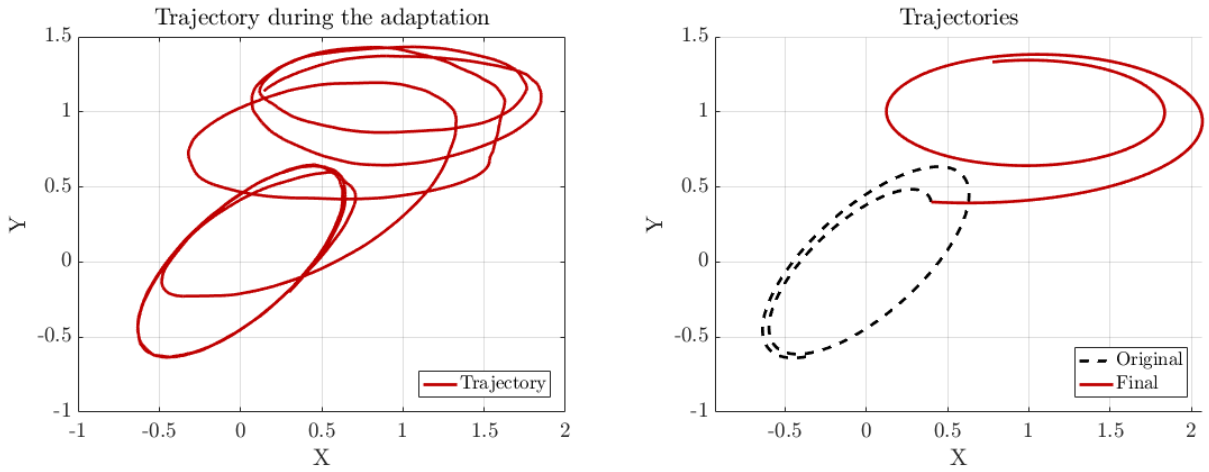
3.2.2 Adding noise to the human motion

In order to relax a bit the first assumption that we made, we can add noise on the parameters that control the human motion in the simulation. To do so we add a random $\delta \in [-0.04; 0.04]$ to each parameters at each step. An example of these parameters over the time is shown in the figure 20. The resulting trajectory is presented figure 21. It looks more like a real possible trajectory shown by a human.



(a) t_x shown by the human (b) t_y shown by the human (c) α shown by the human

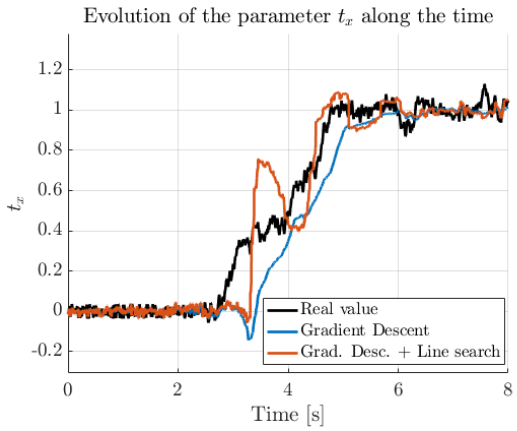
Figure 20: Evolution of the parameters shown by the human



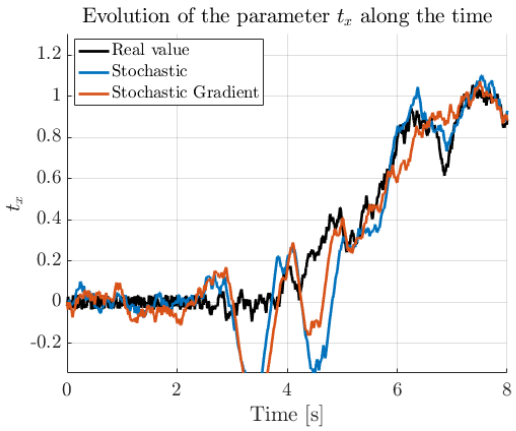
(a) Evolution of the trajectory shown (b) Original & final trajectories

Figure 21: Trajectory shown by the human

The results of the adaptation of the parameters t_x, t_y and α are shown, respectively, in the figures 22, 23, 24. The methods still work well in this case. Indeed they all converge fast to the noisy parameters of the motion shown by the human in the simulation. However, we can say that the efficiency of the stochastic methods is reduced. Indeed, some peaks appear during the adaptation of t_x and α. The stochastic methods are sensible to the noise. It can also be seen in their cost function (figure 26): their integral is bigger than the gradient methods which was not the case previously. Another point to mention is the fact that, thanks to its slower convergence, the gradient descent method gives results that are not too affected by the noise. Maybe the solution is to reduce the convergence speed of the other methods ? To see if this could improve the performances of the algorithms, we reduced some hyperparameters (to slow down the convergence speed) : $R_{sphere} = 0.035$ (stochastic methods) and $\delta = 0.007$ (Gradient descent + Line search). The resulting adaptation of t_x and t_y are shown figure 25. It gives worse results than previously. That's why the gradient descent method seems to be the more appropriate method in this case.

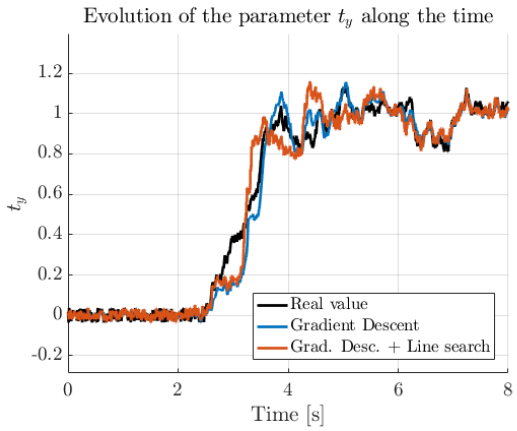


(a) Adaptation of t_x for the gradient methods

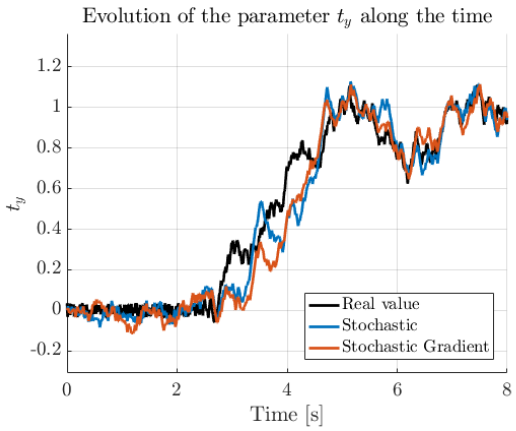


(b) Adaptation of t_x for the stochastic methods

Figure 22: Adaptation of the parameter t_x

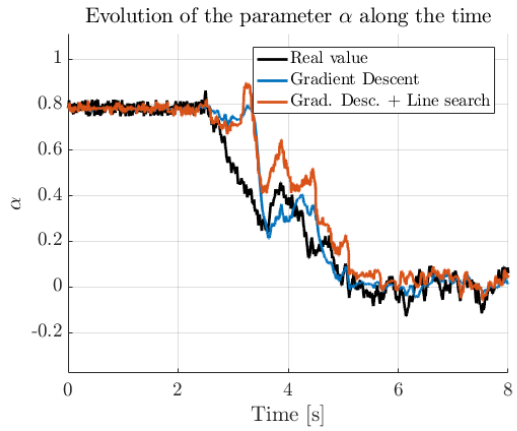


(a) Adaptation of t_y for the gradient methods

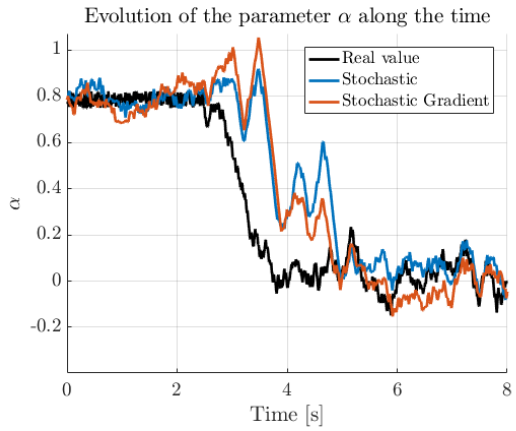


(b) Adaptation of t_y for the stochastic methods

Figure 23: Adaptation of the parameter t_y

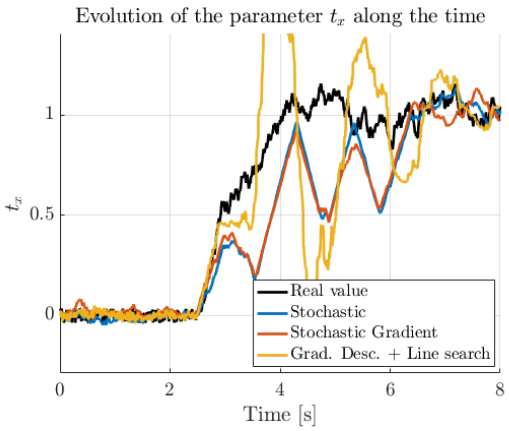


(a) Adaptation of α for the gradient methods

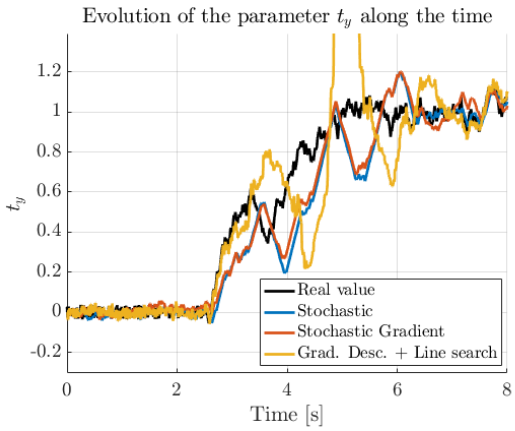


(b) Adaptation of α for the stochastic methods

Figure 24: Adaptation of the parameter α

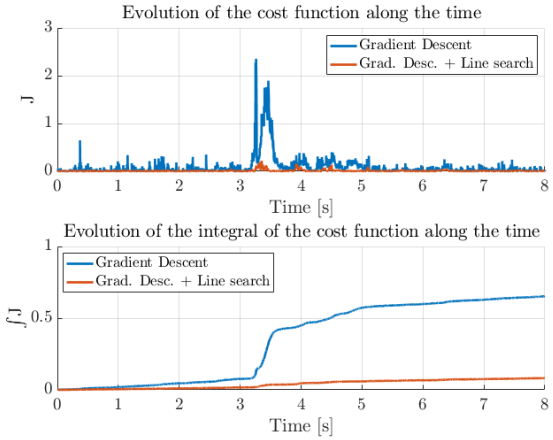


(a) Adaptation of t_x with low convergence

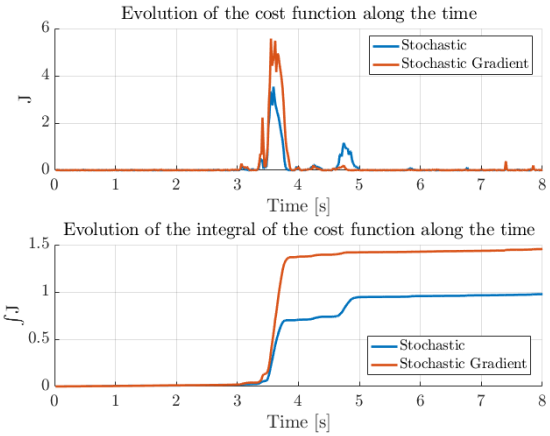


(b) Adaptation of t_y with low convergence

Figure 25: Adaptation of the parameters with low convergence



(a) Cost function the gradient methods

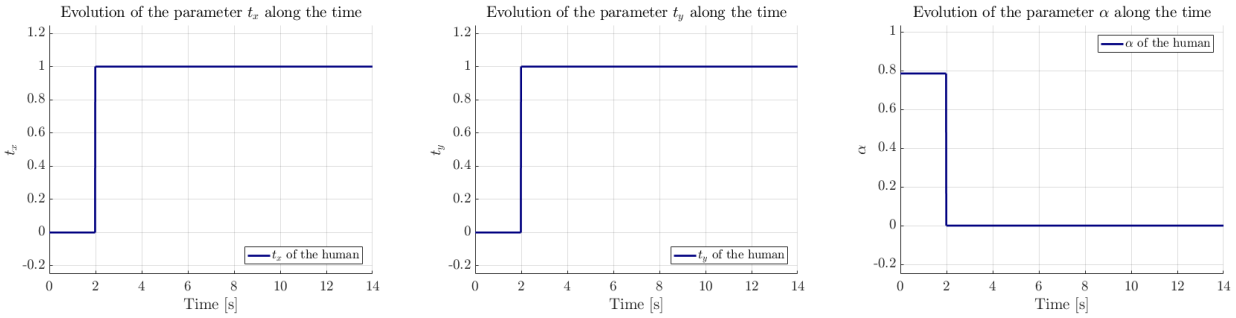


(b) Cost function of the stochastic methods

Figure 26: Evolution of the cost function and its integral of every methods

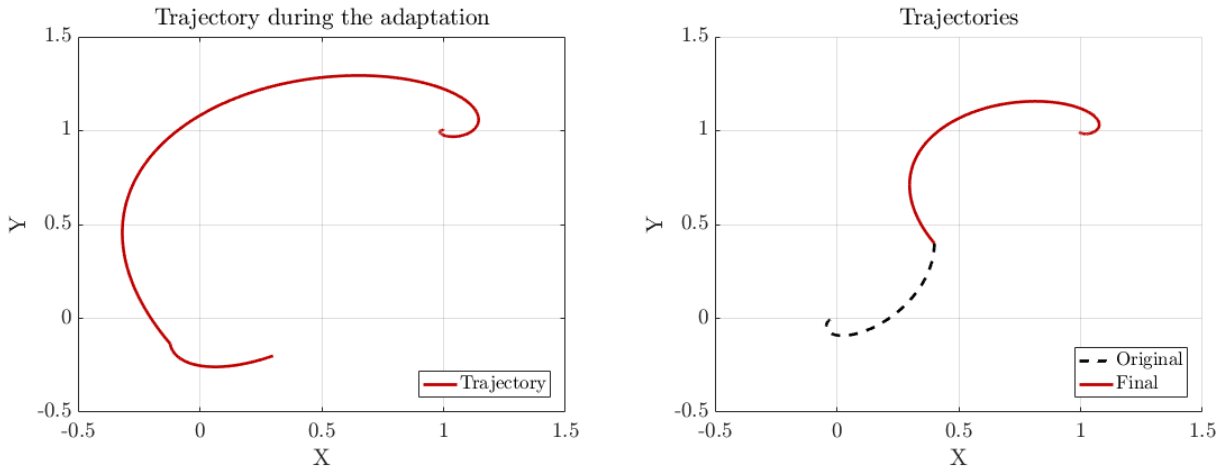
3.2.3 Point to point motion

We can also see how our methods behave with point to point motions. To do so we use the linear D.S. presented equation 19. We make the same assumption as in the first scenario (the human acts as a perfect D.S) to analyze first the "perfect" case. We also consider that the parameters change instantly like it is shown figure 27. The resulting trajectory is presented figure 28. It is composed by two curves: one corresponding to the parameters from $t = 0s$ to $t = 2s$, the other from $t = 2s$ to the end.



(a) t_x shown by the human (b) t_y shown by the human (c) α shown by the human

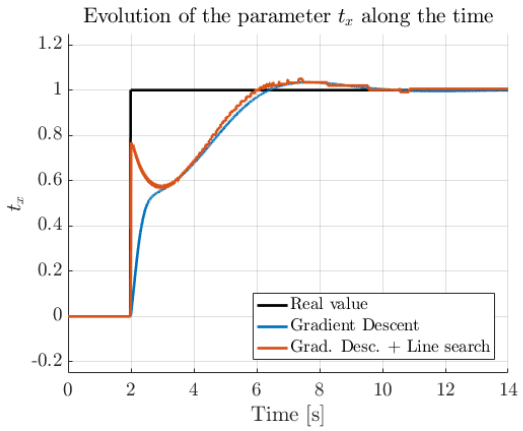
Figure 27: Evolution of the parameters shown by the human



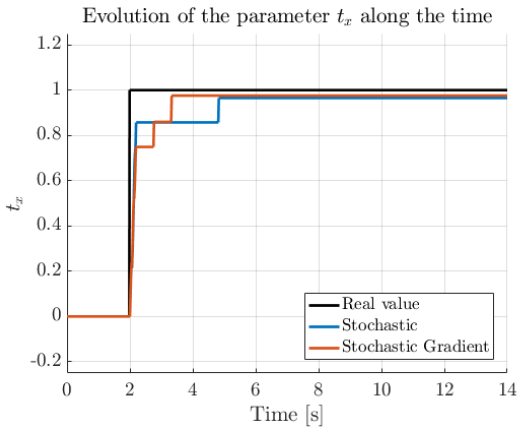
(a) Evolution of the trajectory shown (b) Original & final trajectories

Figure 28: Trajectory shown by the human

If we observe the figure 32, we see that, in the case of the "gradient descent + line search" method (which has the fastest convergence speed), it converges to the wanted value. We can try to increase the convergence speed of each methods in order to see if they work better. To do so we higher some hyperparameters : $\epsilon = 5.10^{-3}$ (gradient descent) and $R_{sphere} = 0.2$ (stochastic methods). The results of the adaptation of the parameters t_x, t_y and α are shown, respectively, in the figures 29, 30, 31. They are really impressive. If we look at the cost functions presented in the figure 33, we see that the methods with the better results are the ones with the faster convergence to 0 when a peak appears. This confirms the fact that speed convergence and adaptation efficiency are linked in this scenario. So, we need to have methods that converge faster for point to point motions because they are momentary. Indeed, the robot does not oscillate like with a limit cycle: it gives it less time to adapt to the human motion. Another aspect that we can observe is the fact that the gradient methods did not adapt the angle α properly. This is due to the fact that this linear D.S. has not a region that characterizes its orientation. Indeed, one with a repulsive squared region for example could have been rotated correctly as its orientation is more precise. However, the stochastic methods converge when adapting α. Indeed, we set one of their hyperparameters to a high value that make this processes converge really fast. The drawback is that they are not really precise as we can see some steps on their curve.

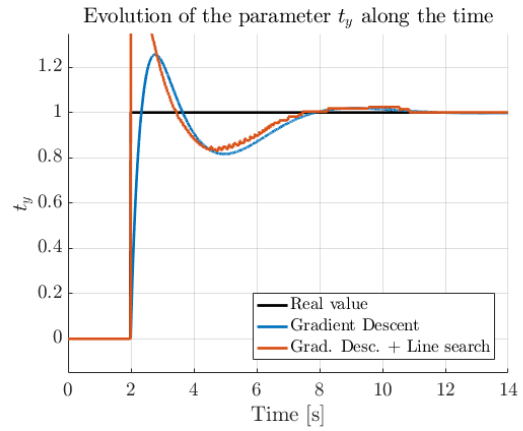


(a) Adaptation of t_x for the gradient methods

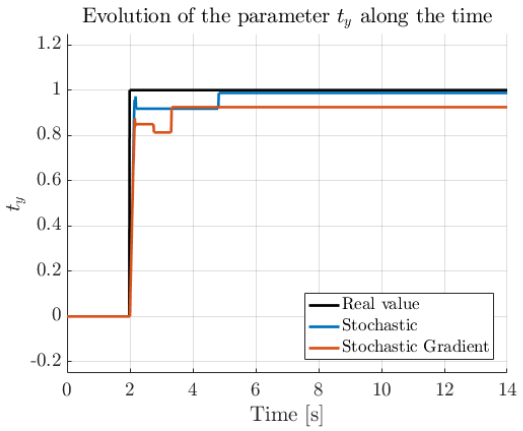


(b) Adaptation of t_x for the stochastic methods

Figure 29: Adaptation of the parameter t_x

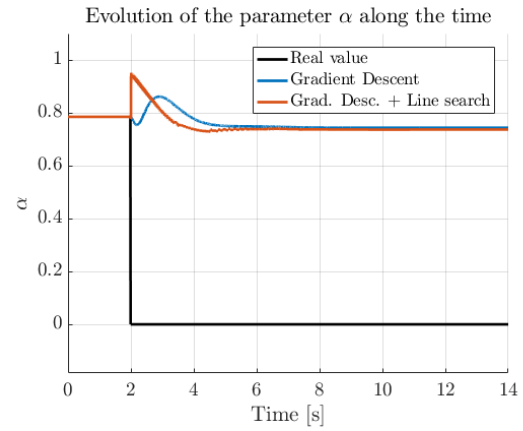


(a) Adaptation of t_y for the gradient methods

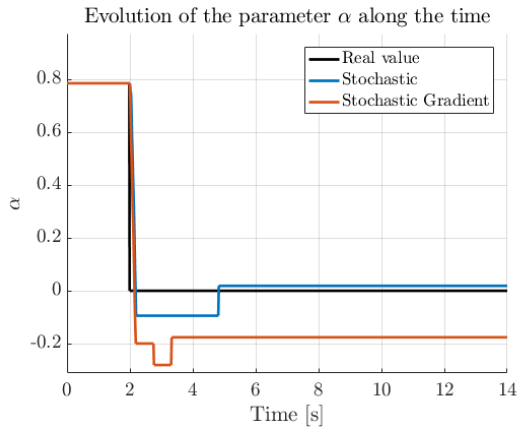


(b) Adaptation of t_y for the stochastic methods

Figure 30: Adaptation of the parameter t_y

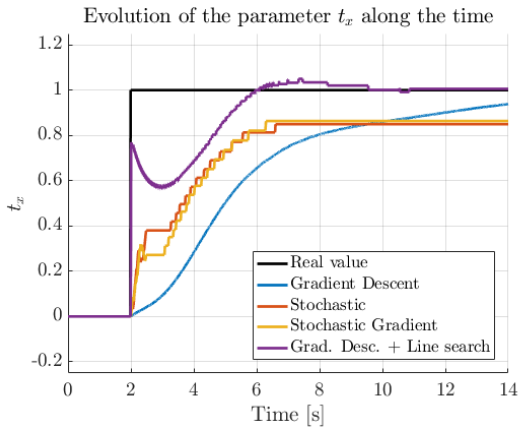


(a) Adaptation of α for the gradient methods

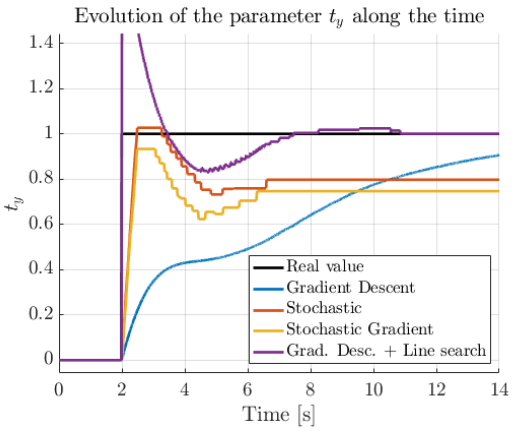


(b) Adaptation of α for the stochastic methods

Figure 31: Adaptation of the parameter α

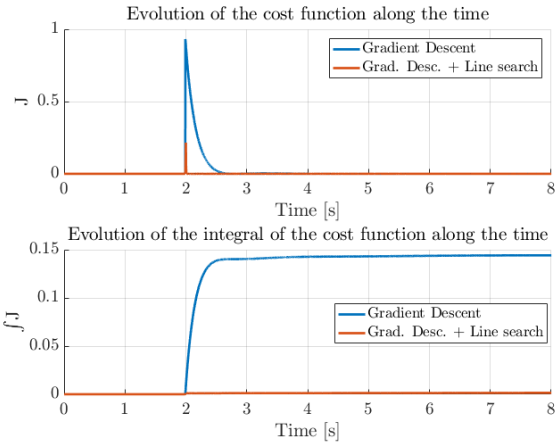


(a) Adaptation of t_x with low convergence

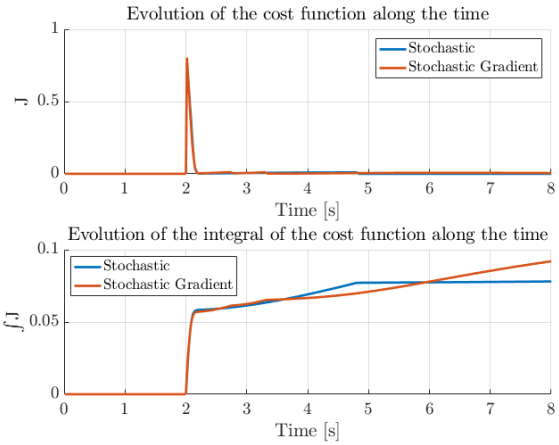


(b) Adaptation of t_y with low convergence

Figure 32: Adaptation of the parameters with low convergence



(a) Cost function the gradient methods

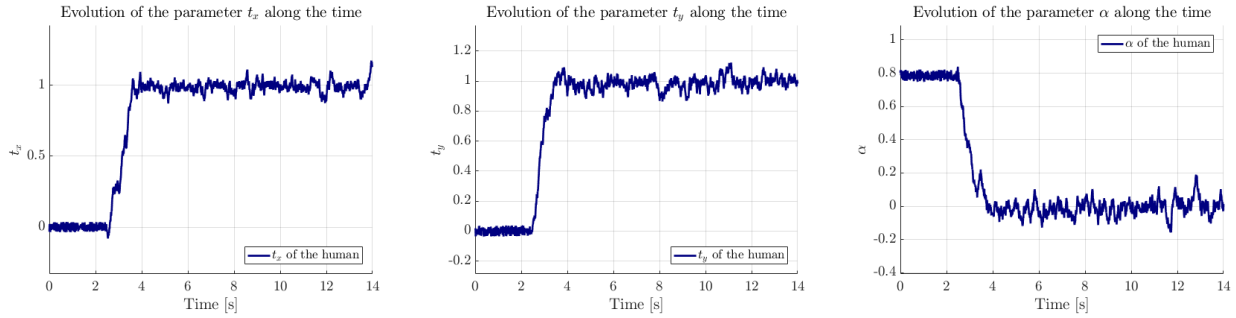


(b) Cost function of the stochastic methods

Figure 33: Evolution of the cost function and its integral of every methods

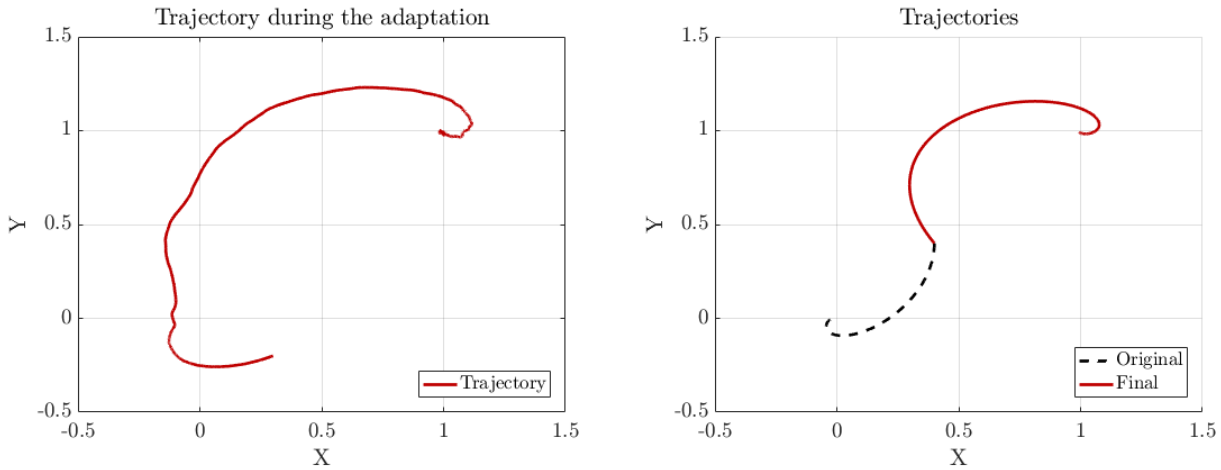
3.2.4 Adding noise to the point to point motion

We can do the same reasoning as we did for the scenario 2: let's add some noise on the parameters in order to make the motion shown by the human less perfect. The new evolution of the parameters is shown in figure 34. The new trajectories are shown in figure 35. As we saw in the last scenario that, due to its ephemeral motion, the convergence speed of the methods must be high, we keep the last value of the hyperparameters that we set before.



(a) t_x shown by the human (b) t_y shown by the human (c) α shown by the human

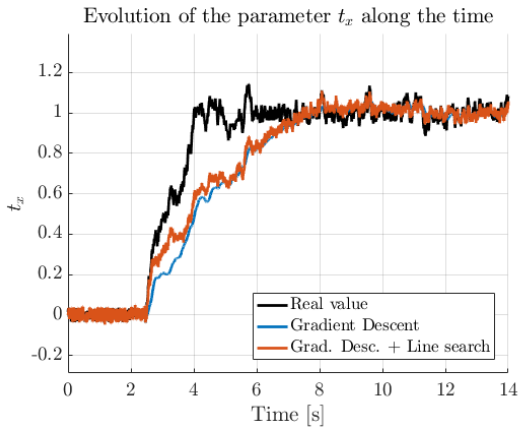
Figure 34: Evolution of the parameters shown by the human



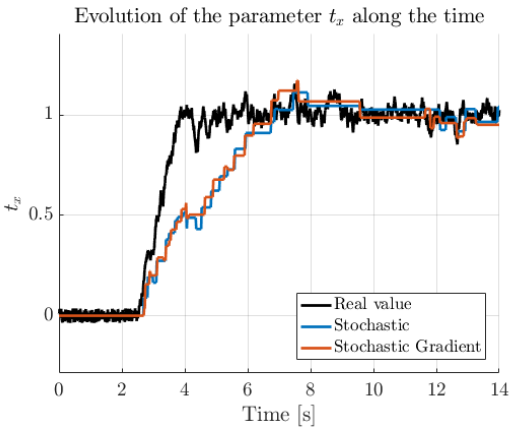
(a) Evolution of the trajectory shown (b) Original & final trajectories

Figure 35: Trajectory shown by the human

The results of the adaptation of the parameters t_x, t_y and α are shown, respectively, in the figures 36, 37, 38. The first quick observation that we can make is the fact that the methods have approximately the same behavior and they converge for t_x and t_y but not for the angle α (the reason has already been discussed in the last scenario). We can also see that this time the convergence speed of the "gradient descent + line search" method is too high which means that it is sensible to the noise. Indeed, if we observe the cost function of the methods presented figure 39, we see that the one corresponding to this gradient process is almost 0 at every time. It adapts itself also to the noise that we add. It's overfitting ! In fact, there is a compromise between convergence speed and sensibility to the noise.

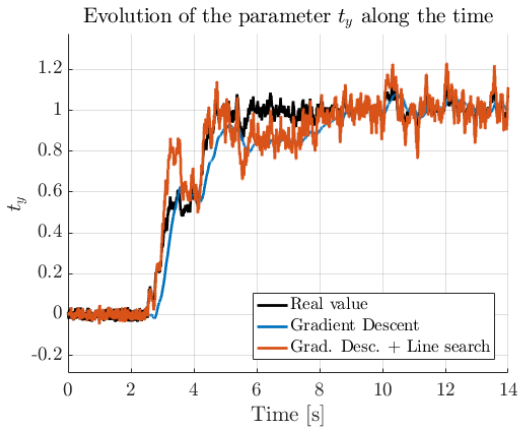


(a) Adaptation of t_x for the gradient methods

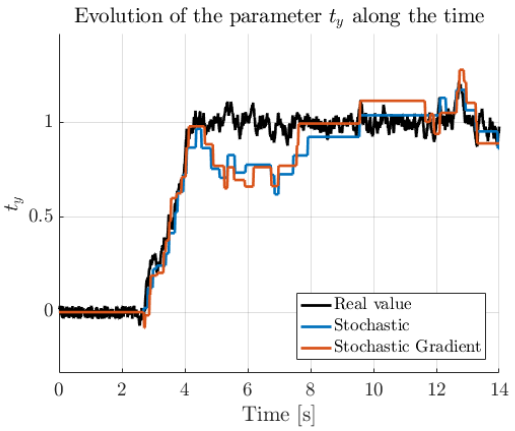


(b) Adaptation of t_x for the stochastic methods

Figure 36: Adaptation of the parameter t_x

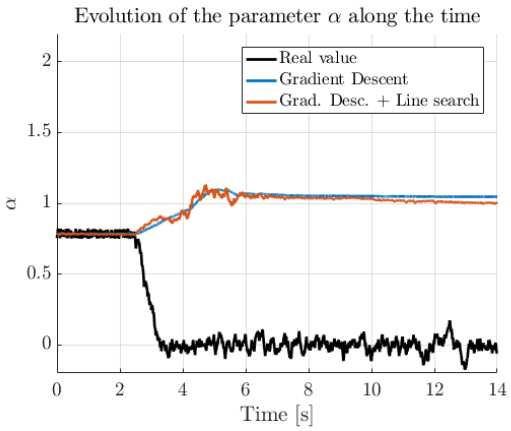


(a) Adaptation of t_y for the gradient methods

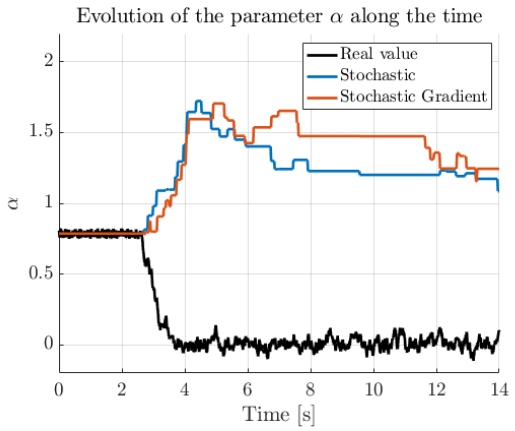


(b) Adaptation of t_y for the stochastic methods

Figure 37: Adaptation of the parameter t_y

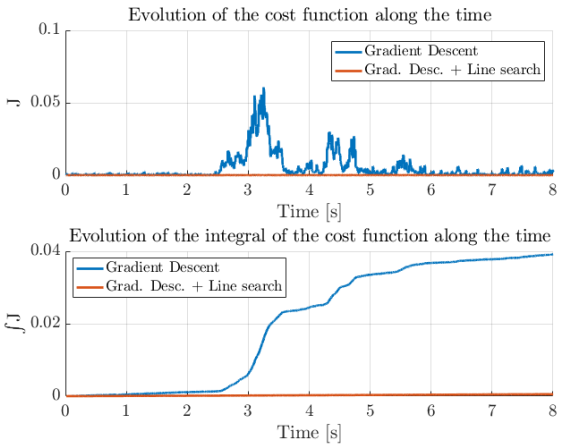


(a) Adaptation of α for the gradient methods

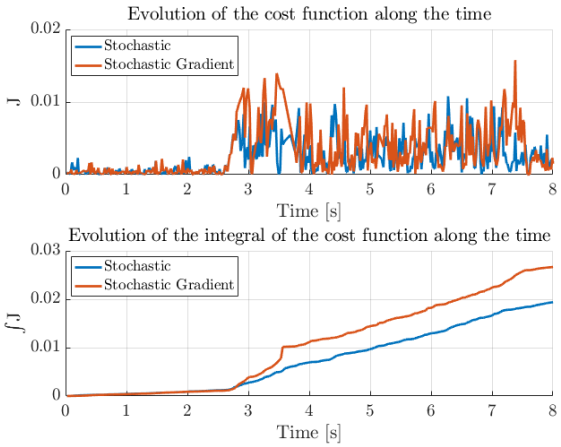


(b) Adaptation of α for the stochastic methods

Figure 38: Adaptation of the parameter α



(a) Cost function the gradient methods



(b) Cost function of the stochastic methods

Figure 39: Evolution of the cost function and its integral of every methods

3.2.5 Point to point motion with a limit cycle dynamical system

One last scenario that is interesting to analyze is the following: what if the robot is performing a cyclic motion and the human grabs it and moves it to another point ? It is equivalent to use the following dynamical system to simulate the robot state which corresponds to the most trivial point to point motion :

$$\dot{\overrightarrow{x_r}} = -a(\overrightarrow{x_r} - \overrightarrow{x_0}) \tag{21}$$

a is a constant determining the speed of the motion. $\overrightarrow{x_0}$ is the target point.
In order to simulate that, we give some initial parameters to the robot. Then from $t = 2s$ to the end we apply him this D.S. The resulting trajectory (with $\overrightarrow{x_0} = (-1.5; -1.5)$) is shown figure 40.

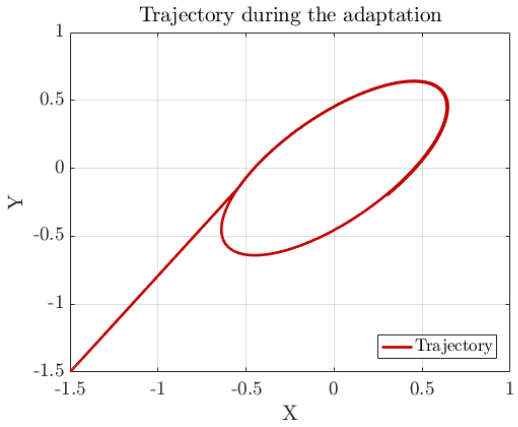
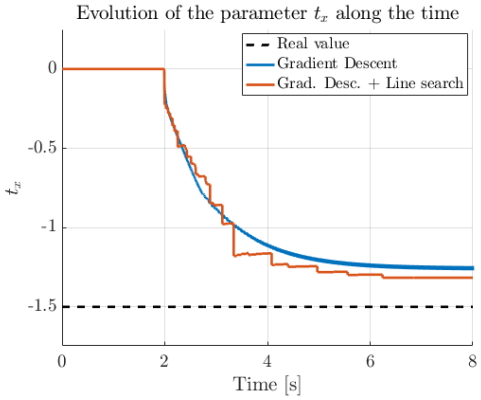


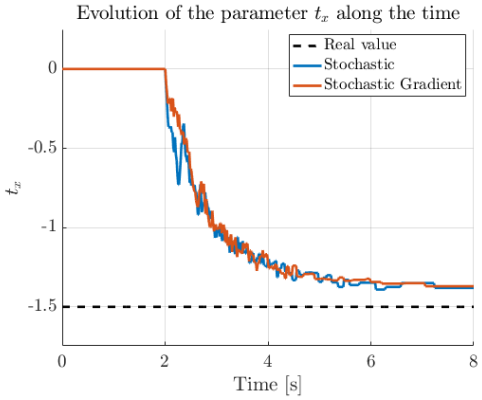
Figure 40: Evolution of the trajectory shown

The results of the adaptation of the parameters t_x and t_y are shown, respectively, in the figures 41 and 42. We do not show the results for α because it was irrelevant. Indeed, in this scenario we focus on translation. The results seems to converge to the right translation parameters which is interesting. The motion shown by the human was not parameterizable with the D.S. used by the robot and it still adapt its parameters the right way. However there is still a little error on the t_x adaptation. Furthermore, the cost function seems to be stuck in a local minimum as the cost functions presented figure 43 do not go to 0.

Given these results we can imagine robotic applications where we could make the robot understand that it has to translate its cyclic motion by moving him on a straight line to the new objective.

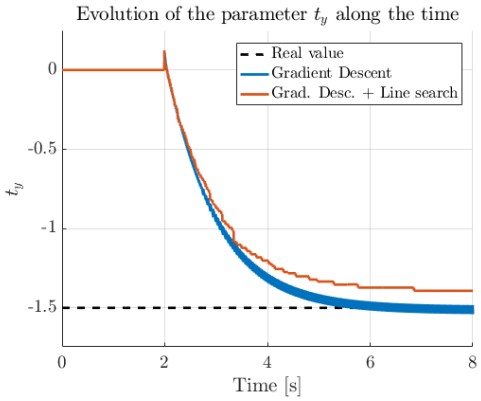


(a) Adaptation of t_x for the gradient methods

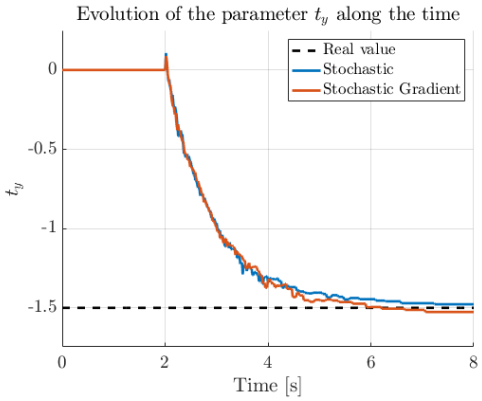


(b) Adaptation of t_x for the stochastic methods

Figure 41: Adaptation of the parameter t_x

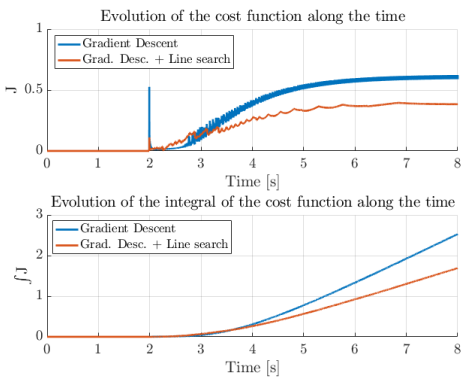


(a) Adaptation of t_y for the gradient methods

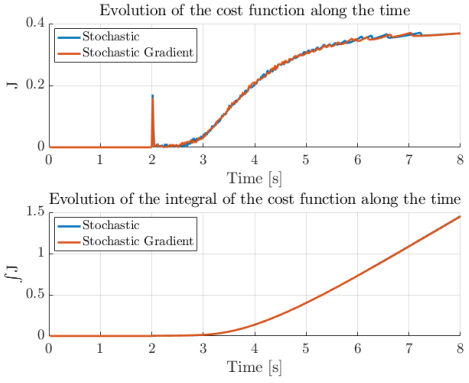


(b) Adaptation of t_y for the stochastic methods

Figure 42: Adaptation of the parameter t_y



(a) Cost function the gradient methods



(b) Cost function of the stochastic methods

Figure 43: Evolution of the cost function and its integral of every methods

4 Discussion

4.1 Performances

In this part we will explain which method is the best to use in a certain scenario.

- Cyclic motions. It appeared that to adapt the parameters while performing cyclic motions, the methods need to have a low convergence speed otherwise they are prone to fluctuate around the optimal parameters. The gradient descent process was the best method to use with a small adaptation rate ϵ.
- Point-to-point motion : With these type of motions it appeared that, the processes had to have a high convergence speed as the motions are transitory (meaning that the motion finishes when the it reaches its final point). The "gradient descent plus line search" method performs well. The stochastic methods are satisfactory in term of convergence but it suffers from low precision.
- Sensibility to noise. The performances of the stochastic methods are more sensible to the noise than the gradient ones. However it is possible to adapt parameters to the noise when the convergence speed is too high with the "gradient descent + line search" method.

4.2 Penalty factor

One possible problem of the implementation discussed is the fact that the parameters are not bounded. Indeed, they could diverge. A solution could be to implement a penalty factor into the cost function that penalizes for large values. This can be done by introducing a coefficient proportional to each translation parameters (as the angle is bounded), to the cost function like this :

$$J_{new} = J_{old}(1 + \sqrt[n]{t_x^2 + t_y^2}) \tag{22}$$

One can think at more complex implementation for example in using derivative terms of the parameters. Moreover, such boundaries on the parameters can be introduced into the optimization as the constraints. This can be formulated as follows.

$$\min_{\theta}(J) \quad with \quad \theta_{min} < \theta < \theta_{max} \tag{23}$$

where θ_{min} and θ_{max} are the minimum (or maximum) values that the parameter θ can take.

5 Conclusion

We saw an implementation of a task adaptation using switching dynamical systems. It was implemented thanks to global parameterizations of dynamical systems. The aim was to minimize a certain cost function based on the similarity between human and the robot velocities with gradient descent methods or stochastic processes. The results obtained with the simulation were encouraging. Our methods are successful in adapting to the optimal values in different scenaii and under different condition such as noisy environment. The next step to do would be to implement this algorithm on real robot in order to see how it performs in the real world. Nevertheless, some interesting behaviors were mentioned, for example: the translation of a cyclic D.S. can be done with a simple point to point motion. The methods to use to have the better results depend also on the type of motion the robot is performing. Transitory motions need to have algorithms that converges fast because of the ephemeral aspect of these motions, whereas the cyclic motions need a low convergence speed otherwise the parameters are not stable.

References

[1] S Mohammad Khansari-Zadeh and Aude Billard. "Learning stable nonlinear dynamical systems with gaussian mixture models". In: *IEEE Transactions on Robotics* 27.5 (2011), pp. 943–957.

[2] Klas Kronander and Aude Billard. "Passive interaction control with dynamical systems". In: *IEEE Robotics and Automation Letters* 1.1 (2016), pp. 106–113.

[3] Bruno Siciliano and Oussama Khatib. *Springer handbook of robotics*. Springer, 2016.