ÉCOLE POLYTECHNIQUE
FÉDÉRALE DE LAUSANNE

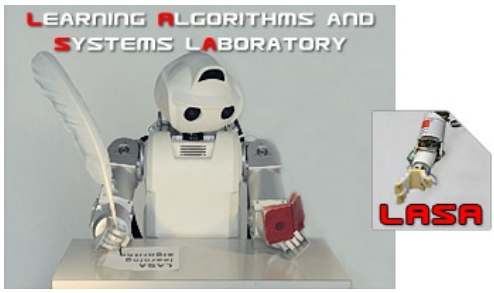
Trajectory representation for incremental robot skill learning

Semester project

September 2015 - January 2016

Stéphane Ballmer

École Polytechnique Fédérale de Lausanne
Learning Algorithms and Systems Laboratory



Supervised by Felix Duvallet, Klas Kronander and Professor Aude Billard

Friday 8th January, 2016

Abstract

This report presents the trajectory processing component, which is the transformation of a demonstration to simple commands on how the robot needs to adapt its trajectory.

The vocal order is attributed to a specific movement. The operator shows the robot by applying a kinesthetic correction in order to teach it how to move. To record a demonstration, here is the process: the robot starts somewhere while executing a behavior, user then provides a correction interactively, the robot reaches the attractor and must isolate the correction applied with the demonstration data, then it transforms it into a continuous curve and finally, to add the possibility of avoiding zones when moving, a Gaussian Process Regression has been used, and improved to take as an input a continuous curve.

To isolate the correction, we developed a specific algorithm based on the instant velocity of the robot and the original direction.

Then to convert the extracted correction into a continuous mathematical function, we will discuss on some interpolation processes and one is chosen: Cubic Spline Process, with benefits from a low computational cost and the best fitting possible.

Finally, we use Gaussian Process Regression to compute the trajectory of the robot in order to align it to the corrections. We will see that this GPR needs to be improved to work with continuous curves (splines), and to do so, an analytic algorithm is developed in order to get the closest point on a continuous curve to another point in the space. The continuous GPR is much faster to execute and also the most accurate for this application.

Some tests were done on the real robot and gave good results, the robot moves well to the order. We were confirmed that the continuous GPR works and gives much better results than the standard one, indeed the robot didn't move all the time as requested with the standard GPR.

There could be some improvements such as optimization of the algorithms (the research of the closest point on many cubic splines can be improved) or developping a concept of showing many times the same correction to the robot to have a better learning, algorithms like putting together many corrections in a 3D space would be developed. However the project is working and the only future work would concern optimization.

Contents

1 Introduction

1.1 Motivation

Traditionally, complex interfaces are used to command robots (joystick, remote, steering wheel, etc). The main goal is to create an easy interface for an untrained operator using natural linguistic language. The robot needs to learn movement that was shown by the operator, and then process it in order to learn from a demonstration. For this reason, the work is based on an incremental learning from a demonstration delivered by a kinesthetic correction.

The operator will shows the robot how to move to an order by doing a kinesthetic demonstration.

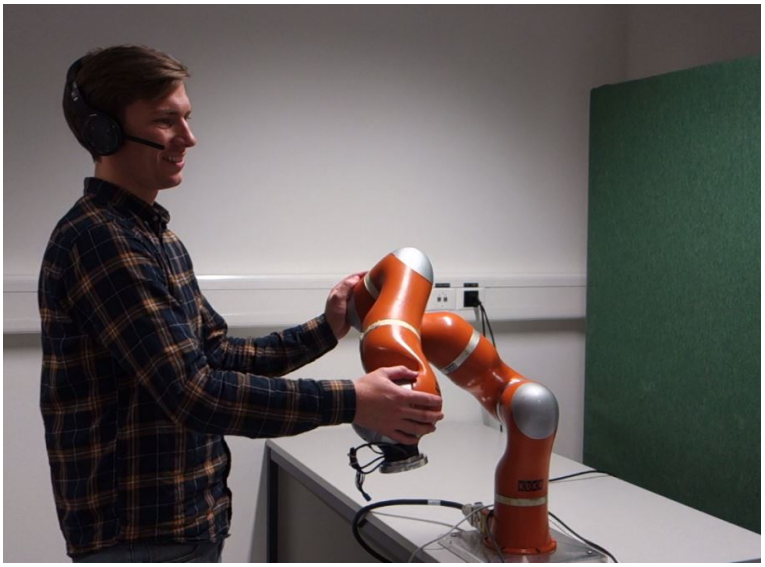


Figure 1: An operator doing a kinesthetic demonstration.

On this project, we worked on the demonstration processing, how the robot understands the movement that was shown to it.

1.2 Background

This project is an incremental work, the robot already knows how to move from one point to another. To show to the robot how to move, the operator needs to do a kinesthetic demonstration. To do so, a target point (or attractor point) is numerically set somewhere reachable in the space and the robot will move to this point. The mission of the operator will be to shift the robot from its trajectory by taking it with his own hands and applying a kinesthetic correction on it. This is a correction on the trajectory, it will be a movement that the robot will learn from and reproduce.

To be able to learn from a correction, a mathematical model has been created, the dynamical system. For any position in the space, the system returns a straight line to the target point.

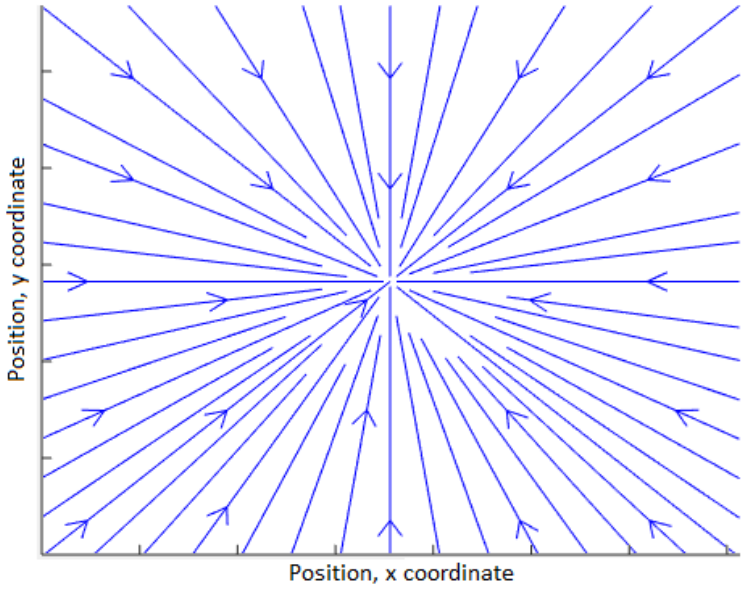


Figure 2: 2D representation of the dynamical system.
Blue arrows represent the direction to follow for the robot from anywhere in the space, the attractor point is situated in the middle.

With no corrections, the model will return a straight trajectory from anywhere to the target (see Figure 2). Also, it's easy to add corrections made by the operator. If a correction is present in the system, all the direction vectors, that initially went through it, should align with the correction (and never cross it). As a result it will curve those direction vectors by applying local little rotations. The rotations will be distributed around the correction by a Gaussian Process Regression, see Figure 3.

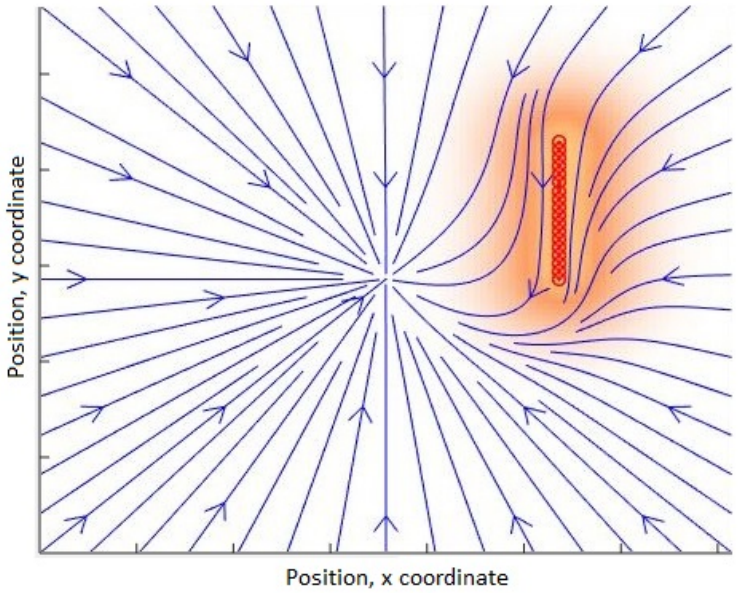


Figure 3: 2D representation of the dynamical system with one correction represented in red and a red gradient showing its influence.

1.3 Goals

After this project, the robot will be able to compute a demonstration and learn about it. The process is done in four steps: first, get the whole demonstration as an input, see Figure 4, then numerically isolate the correction, see Figure 5, after convert the obtained correction into a continuous function (using cubic splines), see Figure 6 and finally transmit the correction to the learning algorithm. It will be possible to compute the influence of the correction for any point in the space by applying a Gaussian Process Regression, see Figure 7.

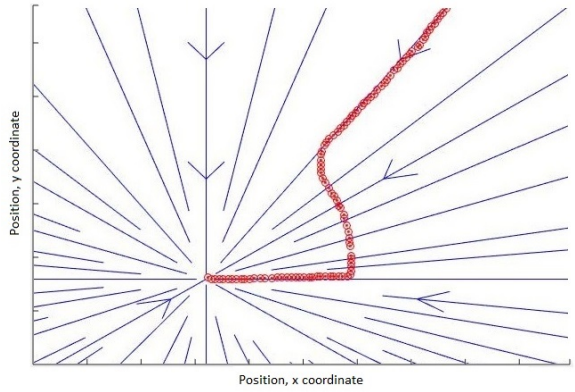


Figure 4:
Step 1: Recording a demonstration.
Red dots are the demonstration.

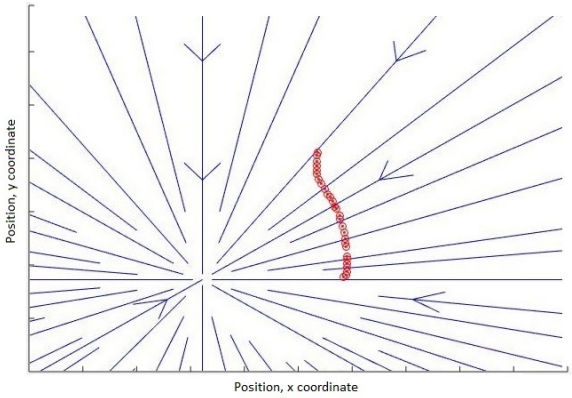


Figure 5:
Step 2: Isolating the correction.
Red dots only represent the correction.

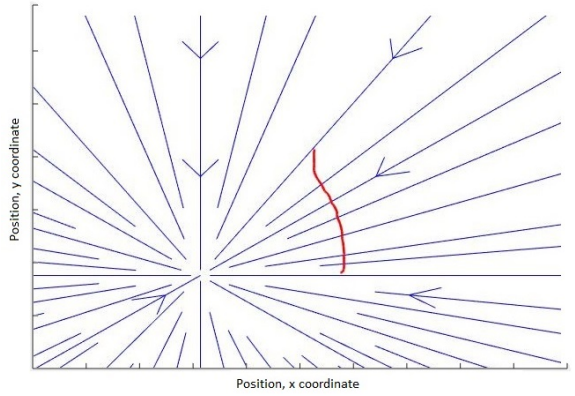


Figure 6:
Step 3: Transforming the correction into a continuous function: spline.
The red line is the spline.

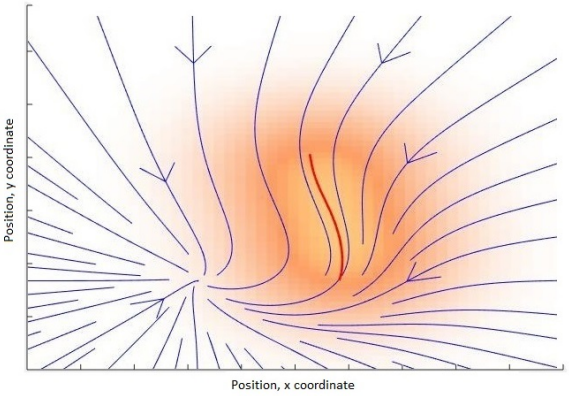


Figure 7:
Step 4: Computing influence.
Red gradient show the influence of the correction.

While using a Gaussian Process Regression in a discrete way to get around corrections, in this special case, some trajectories are still going through the correction instead of aligning to it. To solve this problem, we adapted the GPR for a continuous representation of the correction by using a distance algorithm from a point to a curve.

2 Correction Detection

To record a demonstration, here is the process: the robot starts somewhere while executing a behavior, user provides a correction interactively, the robot reaches the at tractor and must isolate the correction applied in the demonstration data.

2.1 Algorithm

When recording a demonstration, the robot is sending data. The data is composed by many different types such as the position points of the robot, the velocity or the force applied for each of those points. This data represents the complete demonstration, it will be an input for the correction isolation algorithm. This algorithm has to extract the correction by finding the beginning and the end.

2.1.1 Position and Velocity input

One way to do is to analyze the angle between the velocity vector and the original trajectory (which is a direction that is returned from the dynamical system from any point in the space to the attractor). If those vectors are not aligned, the point is part of a correction.

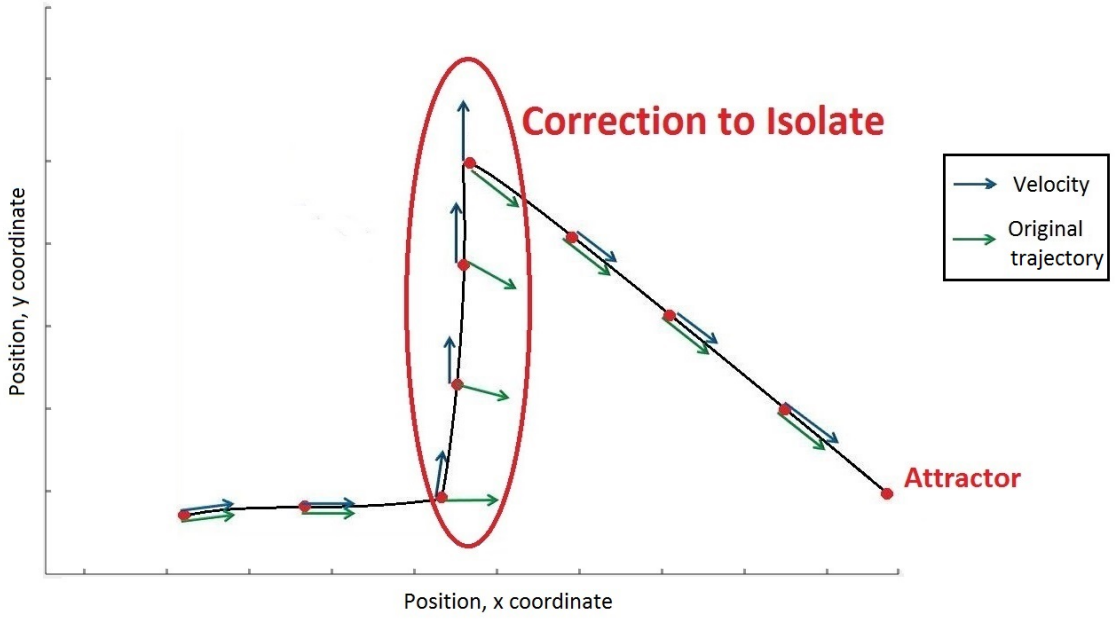


Figure 8: Demonstration containing one correction.

As we can see in Figure 8, when vectors are pointing to a different direction, it means that the point is part of the correction. To easily find the angle between two vectors, we implement a dot product.

$$\overrightarrow{Velocity} \cdot \overrightarrow{Original} = \| \overrightarrow{Velocity} \| \, \| \overrightarrow{Original} \| \cos(\overrightarrow{Velocity}, \overrightarrow{Original})$$

Vectors are both normalized, so if they are aligned, the cosine of the angle will be equal to 1.

Also, there is always noise in measurement and numerical computations, therefore it wouldn't be appropriate to verify that the vectors are exactly aligned. To do so, a precision factor is added.

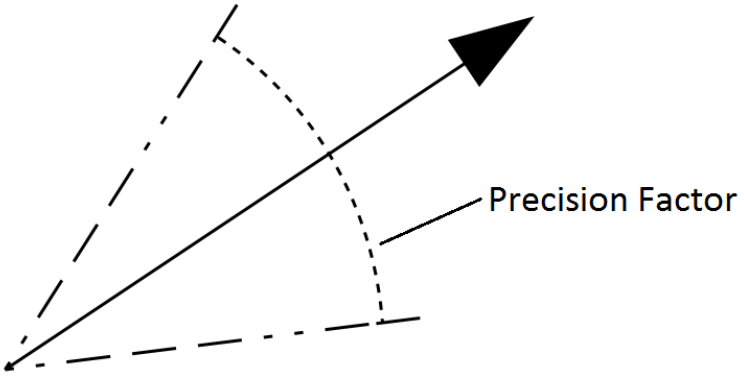


Figure 9: Graphical effect of the precision factor on vector comparison.

If the other vector is in the angle interval, then it means that they are considered to be aligned, see Figure 9. The precision factor is implemented as a simple threshold, it makes the algorithm parameterizable see Algorithm 1.

$\vec{v_1}$ and $\vec{v_2}$ are aligned if: $\cos(\overrightarrow{Velocity}, \overrightarrow{Original}) > 1 - precisionfactor$

Algorithm 1 Correction isolation

input : demonstration (provide x,y,z,x',y',z' which are the coordinates of each point)

for *i in demonstration* **do**
  $dotproduct = x_i x'_i + y_i y'_i + z_i z'_i$
  if $correction\_detected = false$ **and** $dotproduct < (1 - precision\_factor)$ **then**
    $correction\_detected = true$
    $beginning\_correction = i$
  else if $correction\_detected = true$ **and** $dotproduct > (1 - precision\_factor)$ **then**
    $correction\_detected = false$
    $end\_correction = i$
  end if
end for

output : *beginning_correction* and *end_correction*, designate the first and last point of the correction

This algorithm could also detect many corrections in a single demonstration by storing many values in *beginning_correction* and *end_correction*, therefore the Boolean variable *correction_detected* will switch value many times.

2.1.2 Force applied input

The robot also provides information about the external force applied on the arm for each point. Indeed, when an operator applies a correction, he applies a force on the robot with his hand. The aim of this algorithm is to detect this external force. Thus, by computing the norm (to make the algorithm robust for each axis), it's possible to find very precisely the point when a correction starts or ends, see Figure 10.

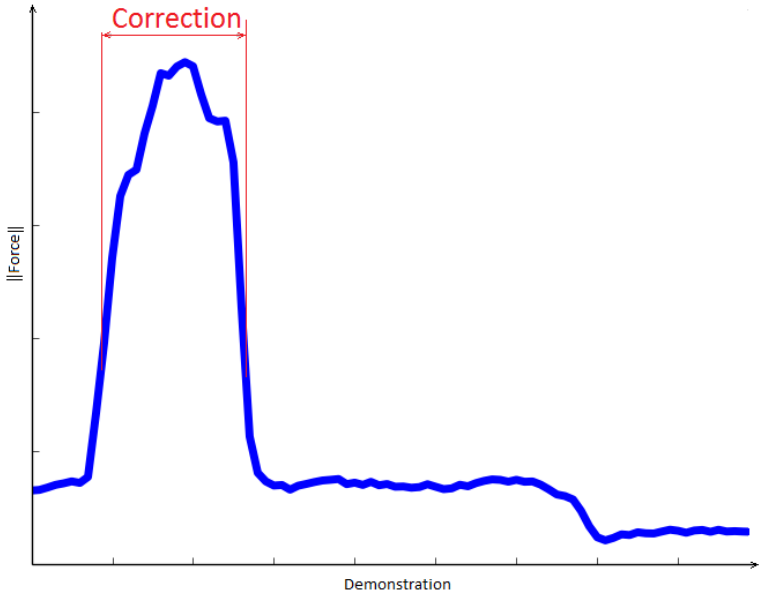


Figure 10: Graph of the norm of external force applied during a demonstration.

The algorithm is exactly the same as Algorithm 1, it's a threshold, when values are going up to a threshold value, it means that they are part of a correction.

This algorithm is very precise but is not applicable on any other robot, indeed not all robots measure the applied force. That's why we will consider it as a ground truth for quantitative analysis, but the position and velocity input algorithm will be selected for this application.

2.2 Algorithm analysis

To do a quantitative analysis, we made many demonstrations on the robot, and using the force algorithm as a ground truth, we tested the precision factor.

Here are the different steps of the analysis: for every demonstration, the force algorithm is applied to find the real correction. The standard velocity algorithm is then applied many times by varying the precision factor from 0 to 1. The result of both algorithms is compared by taking each point that is considered to be in the correction from the velocity, and a percentage of the detected correction is computed with the Equation 1. Finally, the mean of the percentage of all the curves from all corrections was computed.

$$ratio = \frac{Number\ of\ point\ in\ the\ detected\ correction\ and\ also\ in\ the\ real\ correction}{Number\ of\ point\ in\ the\ real\ correction} \tag{1}$$

The detected correction is the output of the correction isolation algorithm based on the velocity and the real one is the algorithm based on the force applied on the robot.

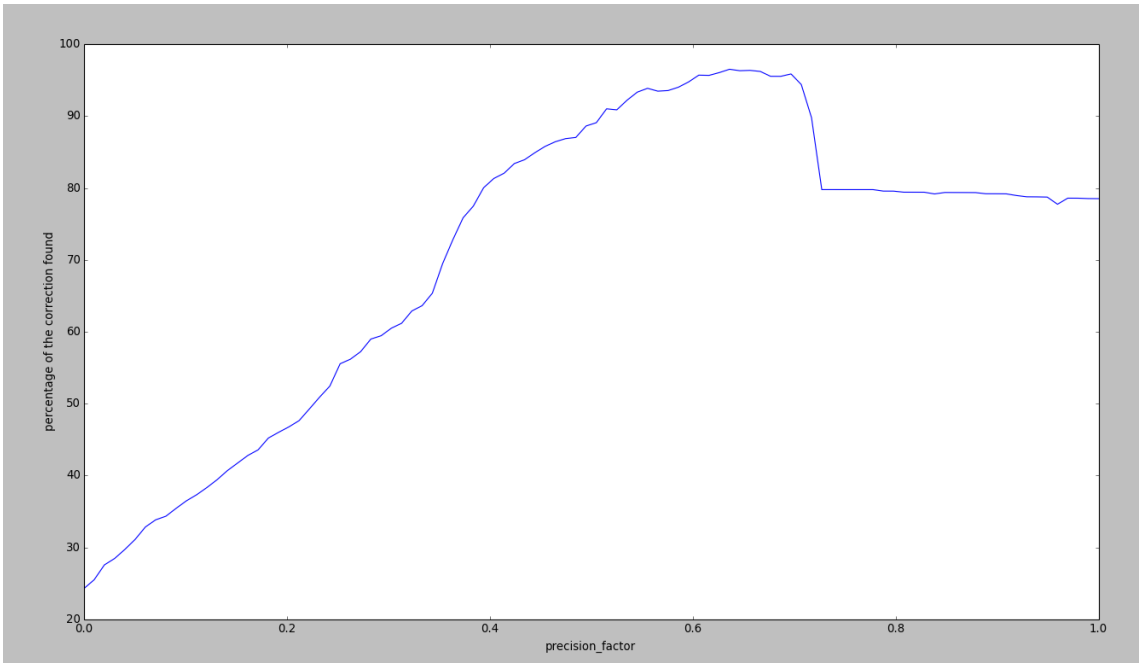


Figure 11: Average of the percentage of the curve detected in terms of the precision factor from 5 demonstrations.

Based on the few demonstrations correction detection results in Figure 11, the precision factor seems to give a good result at a value of $\simeq 0.65$. This result could be used but is absolutely not a reference.

With a precision factor of 0.65, the algorithm will consider that a vector $\vec{v_1}$ is aligned with another vector $\vec{v_2}$ if the cosine is bigger than 0.65: $cos(\vec{v_1}, \vec{v_2}) > 0.65$, in other terms, the angle between $\vec{v_1}$ and $\vec{v_2}$ has to be bigger than 49 *degrees*.

3 Continuous Correction Computation

After correction isolation, in order to give some good data to the learning algorithm, the discrete correction will be interpolated into a continuous mathematical function form. A demonstration is a set of positions with time stamps. In order to represent it as a continuous function, the position coordinate of each point has to be interpolated. To do so, we explored some methods summarized below. The most important criterions are the fitting accuracy and the computational cost.

3.1 Fourier series

It is possible to reconstruct a curve with a sum of cosine and sine functions, which will result to a continuous and derivable function.

$$p(t) = \sum_{k=0}^{n} a_k cos(\omega_k t) + b_k sin(\omega_k t) \tag{2}$$

Advantage

It fits well the data, and it's easily possible to add frequency filtering if there is need for smoothing the correction (for example by removing high frequencies if needed) and this form allows infinite-derivation.

Disadvantage

Fourier decomposition is a very powerful tool but only for frequency analysis. However, in this case, even if high frequencies are filtered, it's possible that the computation results to some peak between training points.

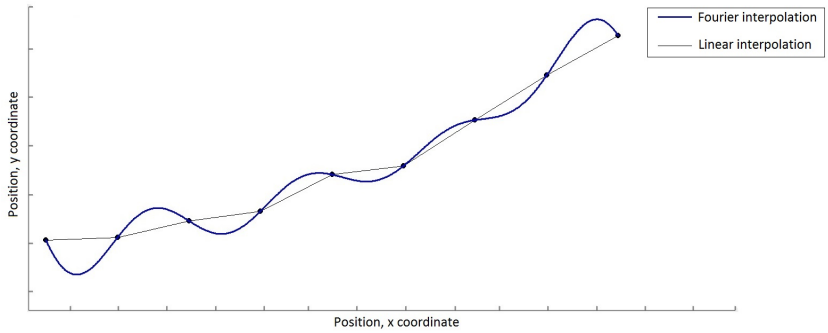


Figure 12: Interpolation example with Fourier series.

In Figure 12, knot points come from a correction (in black) and it's Fourier series interpolation (in blue). The result Fourier curve is oscillating between the points, but still fitting them. This method adds wrong information in the trajectory. It doesn't represent then the correction.

3.2 Non-linear regression with Taylor series

By minimizing the error from discrete data to a n^{th}-order polynomial function, it's possible to interpolate data with a continuous and n-derivative function. This method fits well the knot points. It also keeps the general look of the curve without adding peak or high frequencies, see Figure 13.

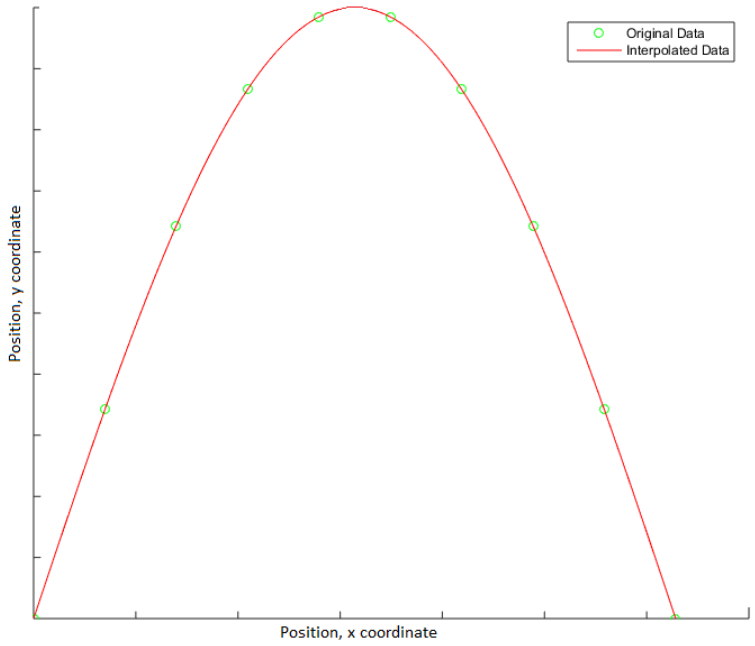


Figure 13: Interpolation example of a 10^{th}order polynomial function.

Advantage

This method is quite fast to compute and represents well the correction. It provides a n-derivative form.

Disadvantage

There are no real disadvantage for this method, only one: this method is not compatible for an analytic distance resolution (which is presented in Continuous Gaussian process regression (4.4.3)).

3.3 Bézier curve

A Bézier curve, also called B-Spline, is a parametric function which is often used in computer graphics.

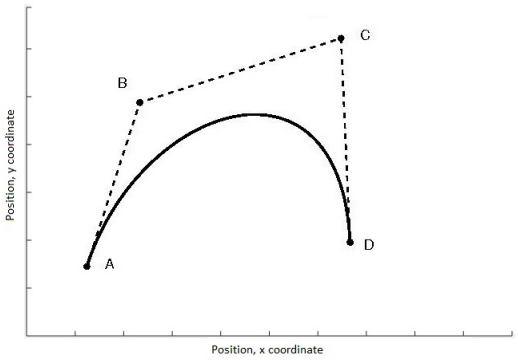


Figure 14: Interpolation example of a B-Spline.[1]

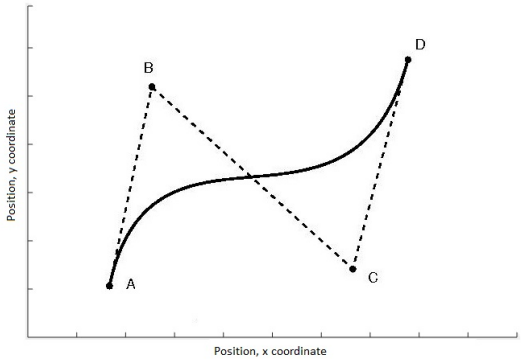


Figure 15: 2^{nd} interpolation example of a B-Spline.[2]

Advantage

This method returns a very smooth result which is, in our case, convenient as there could be some irregularities in the knot points.

Disadvantage

This method does not fit the points at all, the curve is not even going close to the testing points (see in Figure 35 or Figure 15).

[1] Found on: http://pulsar.webshaker.net/2012/08/29/les-courbes-de-bezier-1/
[2] Found on: http://pulsar.webshaker.net/2012/08/29/les-courbes-de-bezier-1/

3.4 Spline

A Spline is composed by a set of many parametric polynomial functions with a given order, cubic spline functions are commonly used in this case. With a cubic spline, it's not possible to fit an entire correction but only 2 position points. For this reason, one Spline is composed of many polynomial functions (called cubic splines), one between each couple of points, see Figure 16.

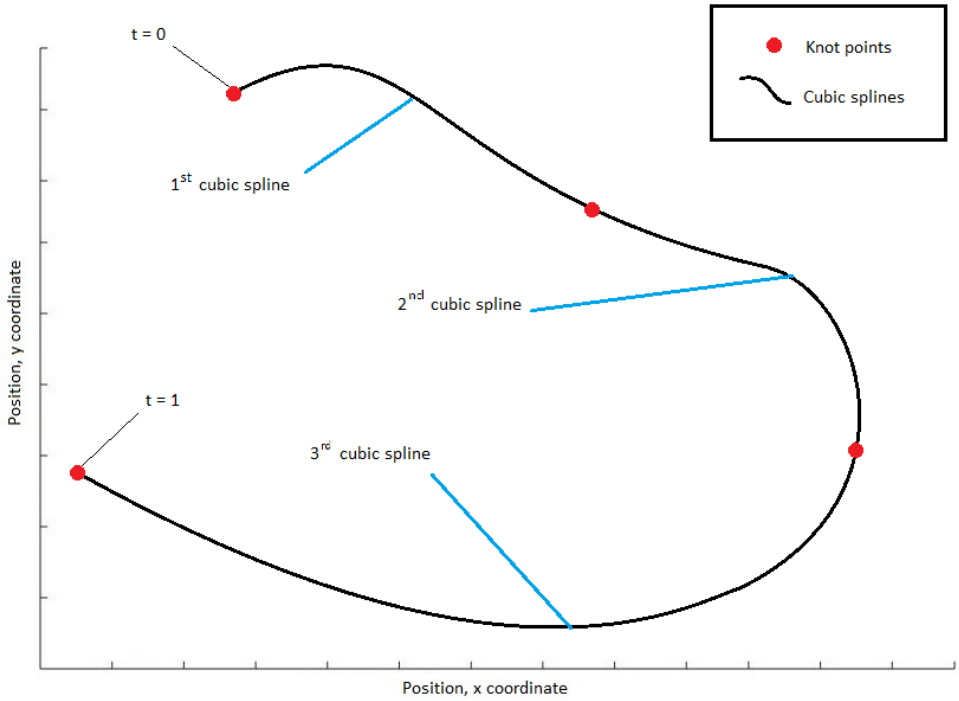


Figure 16: Example of spline interpolation with 4 knot points and 3 polynomial functions.

In our application, the correction is in 3 dimensions. Therefore, there would be 3 Splines, one for each axis (and each spline is composed of many cubic splines). All would be parametrized by the same variable: $t \in [0, 1]$ (t goes from 0 to 1 over the entire correction, not on each polynomial function between points, see Figure 16)
There are many ways to compute the coefficient of the cubic splines that we will see.

3.4.1 Position constraint

It's possible to use natural cubic splines, it is the easiest and most common way to do it. Natural cubic splines $S(x)$ need to follow some properties:

- $S(x)$ should interpolate the position of all data points
- $S(x)$ should be continuous all along the curve
- $S'(x)$ should be continuous all along the curve
- $S''(x)$ should be continuous all along the curve
- For the first and the last points: $S''(x) = 0$

Between every couple of points, in order to follow the 5 rules, there is a system to solve to get the coefficients of the polynomial functions. This is how natural splines are computed, see Figure 17. Indeed, with cubic splines it is very easy to follow the properties enunciate before:

$$\begin{aligned} S(x) &= ax^3 + bx^2 + cx + d \\ S'(x) &= 3ax^2 + 2bx + c \\ S''(x) &= 6ax + 2b \end{aligned} \tag{3}$$

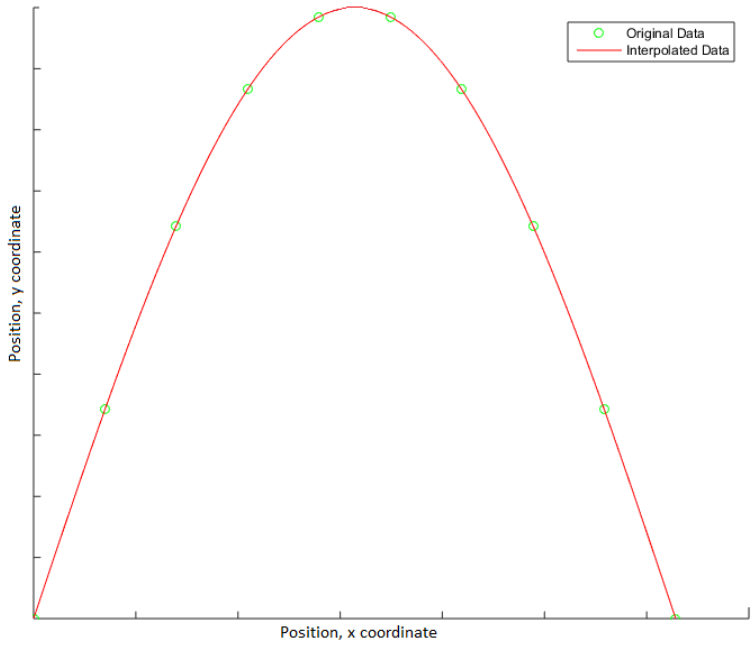


Figure 17: Interpolation example of cubic splines.

Spline allows a good fitting, fast computation time and it appear that the result is the same as the Taylor polynomial fitting. However, Splines and Taylor function are very different, both of them are a polynomial function but, almost all the time, with a different order. Taylor interpolation will be a n^{th}-order polynomial, where n is the number of points and cubic splines are third-order polynomials. To fit the knot points, a spline-interpolation is composed by many cubic spline functions.

3.4.2 Position and velocity constraint

Using cubic splines, it is also possible to compute it with the point's position and velocity. Indeed, By deriving the system (robots provide position and velocity for each point which very convenient in this case) it's possible to solve the system and get the coefficient of the polynomials function.
The properties will be:

- $S(x)$ should interpolate the position of all data points
- $S(x)$ should be continuous all along the curve
- $S'(x)$ should interpolate the derivative of all data points
- $S'(x)$ should be continuous all along the curve

Those conditions are enough to compute the splines.
The curve will be composed by one cubic spline between each point (indeed each data point carries either a position and a velocity).

Advantage

Fitting the position and the velocity gives, with small computation time, the best fit. Indeed, fitting both position and derivative of the position is important for a good continuity of the curve.

Disadvantage

Constraining both position and velocity over-constraint the curve. Sometimes, non-expected results appear, see Figure 18.
In the following graph, each dimension interpolated over a t variable, therefore there 3 Splines.

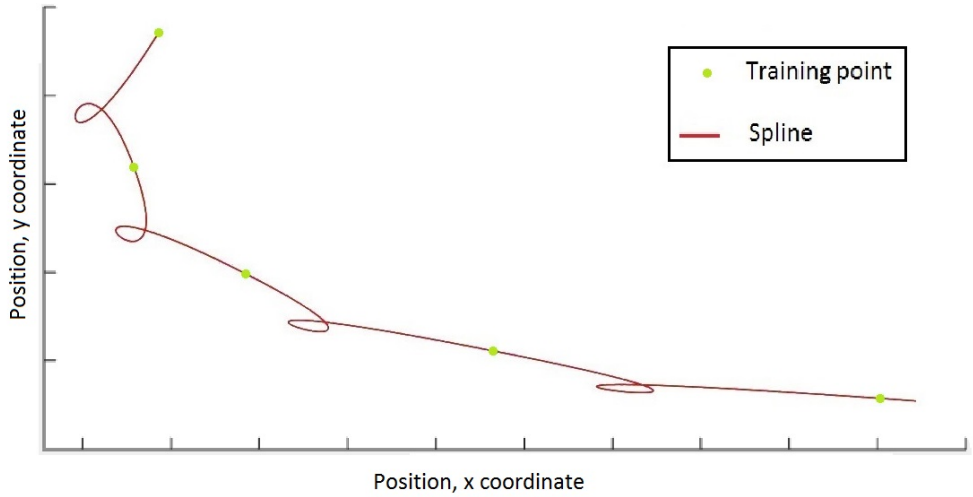


Figure 18: Interpolation example of a problematic over-constraint cubic Spline.

The spline interpolation created some very small loops, this is an effect of over-constraining. This case needs to be avoid because the goal of the curve fitting is to provide clean data, not dirtier data.

3.5 Method chosen

Fourier interpolation could be chosen, but for complexity and general representation of the aspect of the curve this solution wasn't chosen. Furthermore this solution is not compatible for the future distance algorithm.
B-splines do not fit the points which makes the solution interesting for smoothing reason but not usable in our application.
Taylor polynomials is a very good method, but for the future distance algorithm reason, splines are better.
Splines give good results, when constraining them with velocity, some loops appear which is not acceptable, but basic spline (position fitting) is the best method to represent a correction, this method is chosen for next steps.

4 Continuous Gaussian Process Regression

When a correction is added in the dynamical system, it needs to influence trajectories that initially went through. The trajectory *should not* simply go straight forward, and when reaching a correction, following it, see Figure 21. It should anticipate and slowly get around of it in advance in order to have a smooth result, see Figure 19. For this non-linear application, Gaussian Process regression is selected.

4.1 Motivation

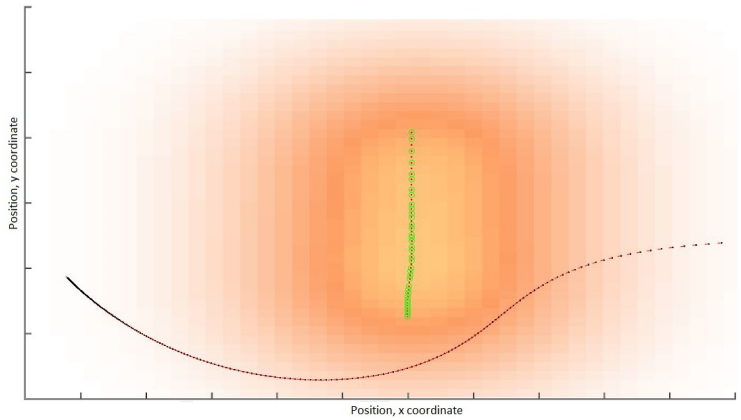


Figure 19: Example of a trajectory that get around a correction. Green circles are the discrete correction, black dot points the trajectory, the red arrows the normalized velocity of each dot and red-orange gradient the amplitude of the influence of the correction.

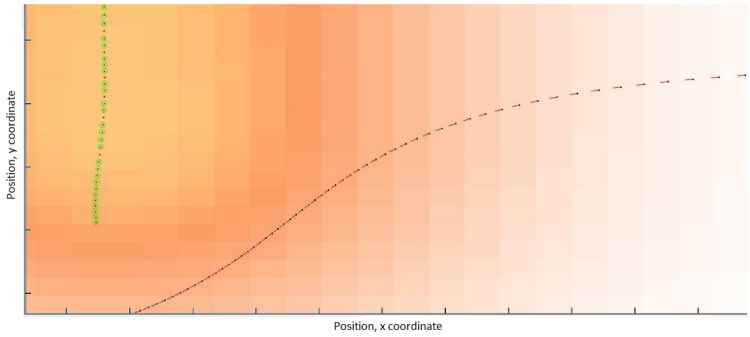


Figure 20: Zoom of Figure 19 system. Example of a trajectory that gets around a correction. Green circles are the discrete correction, black dot points the trajectory, the red arrows the normalized velocity of each dot and red-orange gradient the amplitude of the influence of the correction.

As we can see, the correction needs to get around the correction without simply reaching it and then following it until reaching the attractor, see a bad example in Figure 21;

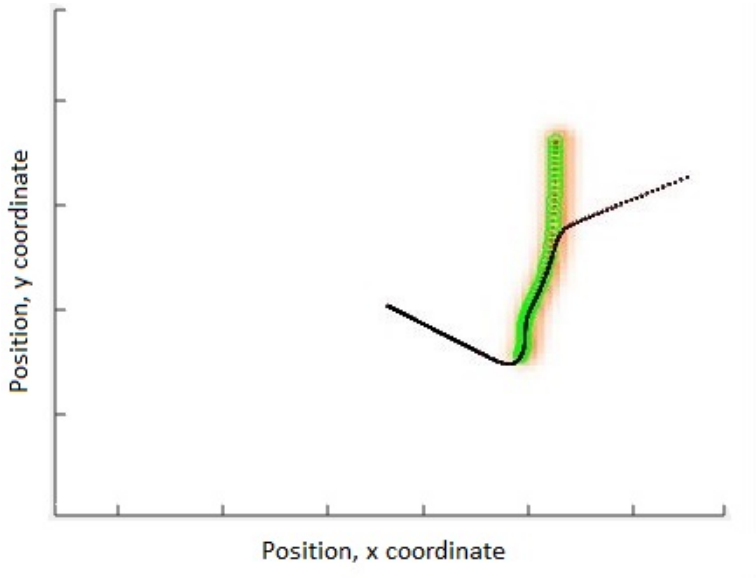


Figure 21: Bad example of avoiding correction.
The trajectory do not get around, it simply follow it. Green circles are the discrete correction, black dot points the trajectory, the red arrows the normalized velocity of each dot.

GPs are used in our application because they encapsulate the property that proximity points have more impact than distant points.

4.2 Quick review on standard GPR

Gaussian Process Regression is a very popular tool for nonlinear regression problems. This tunable algorithm makes powerful and robust applications. However, its biggest drawback is the quadratic computational complexity. Indeed, GPR involves the inversion of an $N \times N$ matrix, where N is the number of training points. That's why this tool becomes relatively slow for application with more than a few thousand of training points in real time.

4.2.1 Definition

The Gaussian Process Regression works by computing estimated data in a defined space. It takes as an input training points and observed value (the training points represent the location of the observed value, there are as many training points as estimated value), and testing points. It returns estimated value at the location of the testing points. Let $\{\mathbf{x_i}\}_{i=1}^{N}$ be a set of N D-dimensional training input vectors $\mathbf{x_i} \in \mathbb{R}^D$ and its associated observed value $\{y_i\}_{i=1}^{N}$ with $y_i \in \mathbb{R}$ for all $i = 1 \dots N$.
The GPR works using a covariance function $f(x_i, x_j)$ and a covariance matrix $\mathbf{K}$ defined as:

$$\mathbf{K} = \begin{bmatrix} k(\mathbf{x}_1, \mathbf{x}_1) & \dots & k(\mathbf{x}_1, \mathbf{x}_N) \\ \vdots & \ddots & \vdots \\ k(\mathbf{x}_N, \mathbf{x}_1) & \dots & k(\mathbf{x}_N, \mathbf{x}_N) \end{bmatrix} \tag{4}$$

4.2.2 Joint distribution

Assuming a mean of 0 and a variance of σ_n^2, the joint distribution of the observed output is:

$$p(\mathbf{y}) = \mathcal{N}(\mathbf{0}, \mathbf{K} + \sigma_n^2 \mathbf{I}) \tag{5}$$

where $\mathbf{y} = [y_1, \ldots, y_N]^T$ are the observed values. We assume that the outputs are independent of their observed values, indeed the covariance function only depends on the input.

By extending our distribution to a x^* testing point and its associated y^* observed value, the distribution becomes:

$$p\left(\begin{bmatrix}\mathbf{y}\\ y^*\end{bmatrix}\right) = \mathcal{N}\left(\mathbf{0}, \begin{bmatrix}\mathbf{K} & \mathbf{k}^*\\ \mathbf{k}^{*T} & k(x^*, x^*)\end{bmatrix} + \sigma_n^2 \mathbf{I}\right) \tag{6}$$

where $\mathbf{k}^* = [k(x_1, x^*), \ldots, k(x_N, x^*)]^T$. This Gaussian process will predict y^* using Equation 6. Standard Gaussian conditioning yields:

$$p(y^*|\mathbf{y}) = \mathcal{N}(\mu^*, c^*) \tag{7}$$

with known forms of μ^* and c^*. For brevity, we only write the form for μ^*:

$$\mu^* = \mathbf{k}^{*T}(\mathbf{K} + \sigma_n^2 \mathbf{I})^{-1}\mathbf{y} \tag{8}$$

The result equation needs an inversion of a $N \times N$ matrix, this is the main problem of computational cost of GPR.

4.2.3 GPR influences of the correction

In our application, see Figure 19, the correction is composed by training points (x_i), the angle between the velocity of each points of the correction and the original trajectory at those points are the observed values (y_i values) and the testing points are the successive points of the trajectory (that's why we use the GPR in real time, each call of the GPR is a new point x^* of the trajectory to get the corresponding value of y^*).
The effect of the GPR is to compute the estimated angle for a given testing point, and rotate its velocity. With this process, trajectories avoid properly correction.
Some problems remain, the computational cost of the GPR becomes too big when many corrections are added to the system (see quantitative analysis Figure 34). It makes the robot being very slow. Also, due to the local influence of each training point, the influence is not globally distributed on the correction, see Figure 22 and Figure 23.

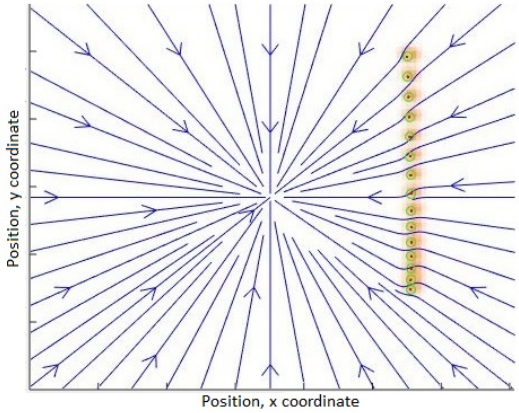


Figure 22: Influence of the correction with a low length-scale

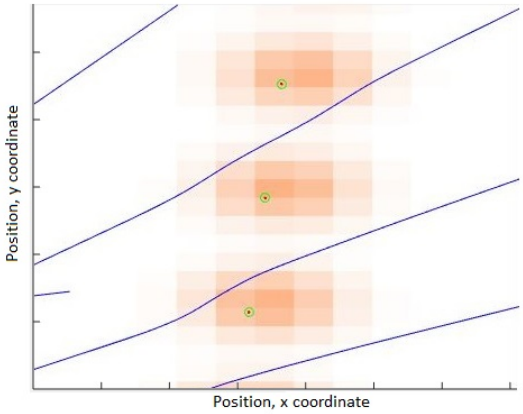


Figure 23: Influence of the correction with a low length-scale zoom

When zooming (and playing with hyper-parameters for more clarity) the effect from two neighbor training points can be smaller than what is necessary and some holes appears in the influence. It results in a trajectory that goes through the correction, see Figure 24.

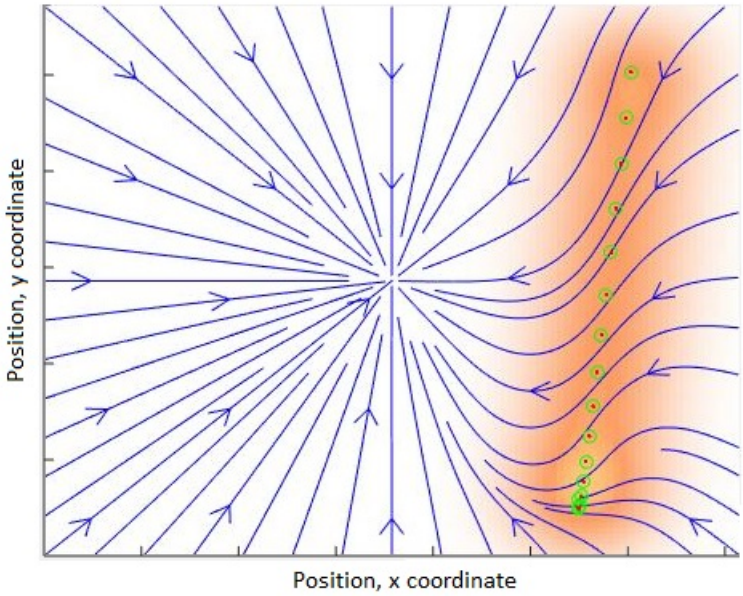


Figure 24: Example of a trajectory going through the correction.

This particular problem mostly happens when training points are far from each other (and when testing it on the real robot, the situation appears quite often which makes this problem important to resolve).

The correction is interpolated into a continuous curve, so one solution would be to generate linearly spaced new interpolated data, and provide it to the GPR. However, to avoid the problem, a lot of data should be generated, which would make explode the computational cost (see quantitative analysis Figure 34).

Because of the slow GPR (using in real time), there are no numeric possible solutions.

But an analytical one could be possible, adapting the GPR to work with a continuous representation of the data.

4.3 Adaptation of continuous GPR

We will propose in this part a new form of GPR which will take instead of training discrete points, a continuous curve. The aim will be to drastically reduce the computational cost and increase the robustness (see Figure 24 problem). It will be done by selecting a particular covariance function that annihilates influence of all the curve but one points and so getting rid of the matrix inversion.

4.3.1 Continuous input curve for GPR

The main difference between a discrete GPR and a continuous GPR is that the training data will be a continuous curve. This curve C is parametrized by a t-value $t \in [0, 1]$ and allows the access of any point on the curve for a defined t, $c(t) \in \mathbb{R}^D$.
In our case, the continuous curve will be the correction (as extracted from a demonstration in section 2) interpolated into a spline (as shown in section 3).

4.3.2 Curve based covariance function

The key of our regression is the covariance function. Let $x' \in \mathbb{R}^D$ be a point of the curve C such as $x' = c(t')$ and x^* a testing point. The covariance function is defined by:

$$k(\mathbf{x}', \mathbf{x}^*) = exp(-(\mathbf{x}^* - \mathbf{x}')^T \mathbf{\Lambda}(\mathbf{x}')(\mathbf{x}^* - \mathbf{x}')) \tag{9}$$

where $\mathbf{\Lambda}(\mathbf{x}') \in \mathbb{R}^{D \times D}$ is a metric that depends on the curve C and the point $\mathbf{x}'$. Let $\mathbf{e}'_t$ denote the tangent vector of C at $\mathbf{x}'$. Then, $\mathbf{\Lambda}$ is defined as:

$$\mathbf{\Lambda}(\mathbf{x}') = \begin{bmatrix} \mathbf{e}'_t & \dots & \mathbf{e}_D \end{bmatrix} \text{diag}([\lambda_1, \dots, \lambda_D]) \begin{bmatrix} \mathbf{e}'_t & \dots & \mathbf{e}_D \end{bmatrix}^T \tag{10}$$

where the vectors $[\mathbf{e}'_t, \dots, \mathbf{e}_D]$ constitute an orthonormal basis for $\mathbb{R}^D$.

4.3.3 Continuous GPR

With Equation 10, by increasing λ_1 towards infinity, the covariance function $k(x', x^*)$ tends to zero and x^* satisfying $(\mathbf{x}^* - \mathbf{x}')^T \mathbf{e}'_t \neq 0$, i.e for any testing point in the space which is aligned to the normal of the tangent of the curve at x', the covariance function will go to 0 and the matrix $\mathbf{K}$ in Equation 4 becomes diagonal. This means that Equation 8 become:

$$\boldsymbol{\mu}^* = \mathbf{k}^{*T} \text{diag}\left(\left[\frac{1}{k(\mathbf{x}_1, \mathbf{x}_1) + \sigma_n^2}, \dots, \frac{1}{k(\mathbf{x}_N, \mathbf{x}_N) + \sigma_n^2} \right] \right) \mathbf{y} \tag{11}$$

Assuming that λ_1 is large and x^* is aligned to the normal of the tangent of the curve at the point x', *there is only one non-zero element in* $\mathbf{k}^*$, it is x'. Equation 11 can be simplified to:

$$\boldsymbol{\mu}^* = \frac{k(\mathbf{x}', \mathbf{x}^*)}{k(\mathbf{x}', \mathbf{x}') + \sigma_n^2} y' \tag{12}$$

Since we only have to compute the kernel with a single point (x'), as long as we can find this point fast enough we reduce an $N \times N$ matrix inversion to 1×1. We will see that x' is simply the closest point of x^* on the curve. It will be the focus of the next section.

4.3.4 Continuous GPR implementation

The continuous GPR makes computation quite fastly and easily by looking at Equation 12. However this equation needs x' which is the point on the curve where the normal of the tangent is aligned to the testing point x^*. This point is the closest point of x^* on the curve, see Figure 25.

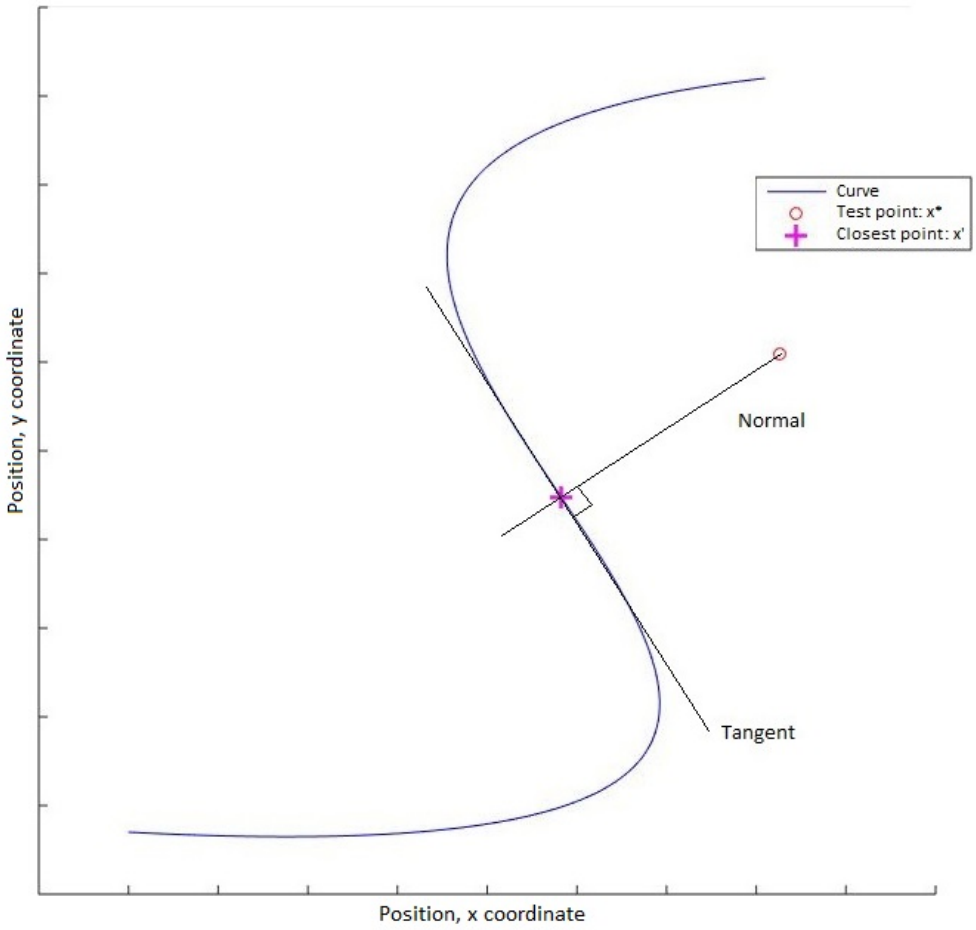


Figure 25: The closest point is the one which has an aligned normal.

For the continuous GPR implementation, an algorithm that searchs for the closest point of a testing point in the space on a continuous curve is required.

4.4 Distance Algorithm

For the continuous GPR adaptation, a distance algorithm is needed. This algorithm has to compute the closest distance from a point (which is anywhere in the space) on the correction (which is a continuous curve).

4.4.1 Brute force

Brute force is a well-known algorithm, not made for optimal-computational cost but easy to implement.
It works by first choosing a fix number of points on the curve which are equally distant. Those points are the interpolated points (compute by the continuous curve), they are not the knot points (knot points are not necessarily linearly spaced, with interpolated point it's possible to get any point anywhere and how much we want). Then the distance is computed for each of the interpolated points. The closest one is the closest point (see Algorithm 2).

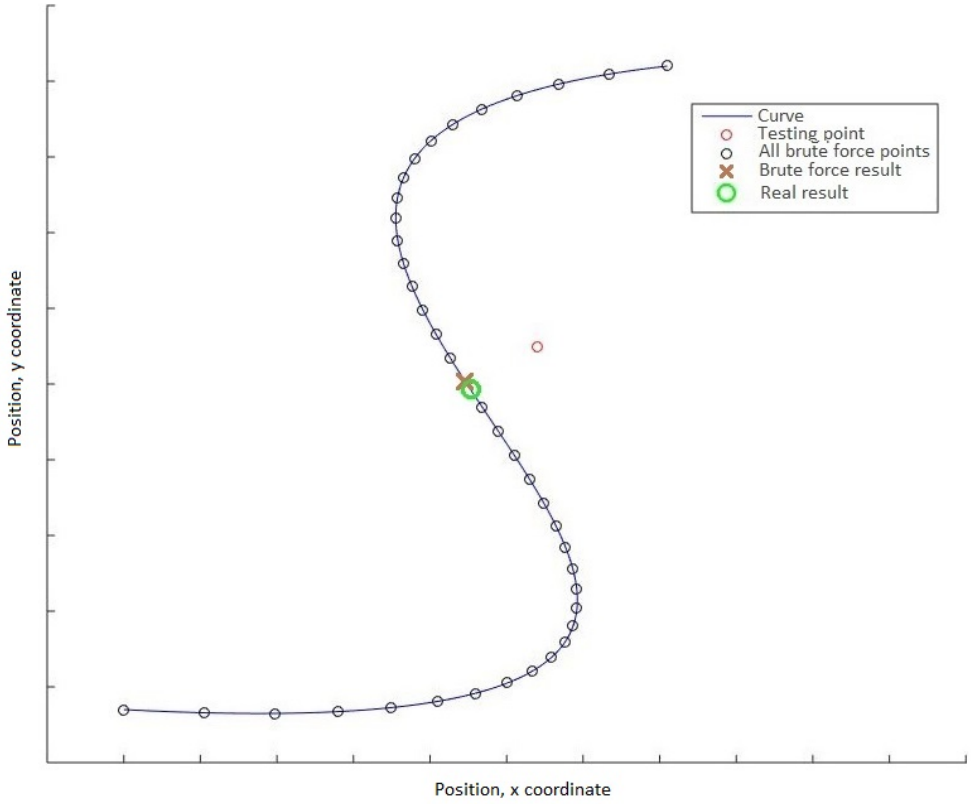


Figure 26: Example of brute force algorithm with 40 points.

Algorithm 2 brute force search

input : continuous correction: $x_{xyz}(t)$, testing point: x^*_{xyz}

$N = 10000$ #N can be given as an input
$indices = linearly\_spaced(0, 1, N)$ #provides N linearly spaced points from 0 to 1
$result = 0$

for i *in* $indices$ **do**
  if $distance(x_{xyz}(i), x^*_{xyz}) < distance(x_{xyz}(result), x^*_{xyz})$ **then**
    $result = i$
  end if
end for

output: the closest point: $x_{xyz}(result)$

Advantage

This algorithm is very easy to implement and gives quite accurate results.

Disadvantage

This algorithm is very heavy, takes a lot of computational cost, $O(n)$ where n is the number of points, and it's precision directly depends on the number of points. Typically, a good number of point for this application is $n = 10000$, which takes too much time to compute.

4.4.2 Dichotomy

This algorithm, also called Bisection-method, is based on selecting successively sub-intervals and testing the distance of the boundary points. Sub-intervals are selected by choosing the middle of the latest sub-intervals until converging to the good point (see Algorithm 3).

Another application for this algorithm is searching a value in an ordered array, by taking the middle of the sub interval until reaching the value or the place where the value should be.

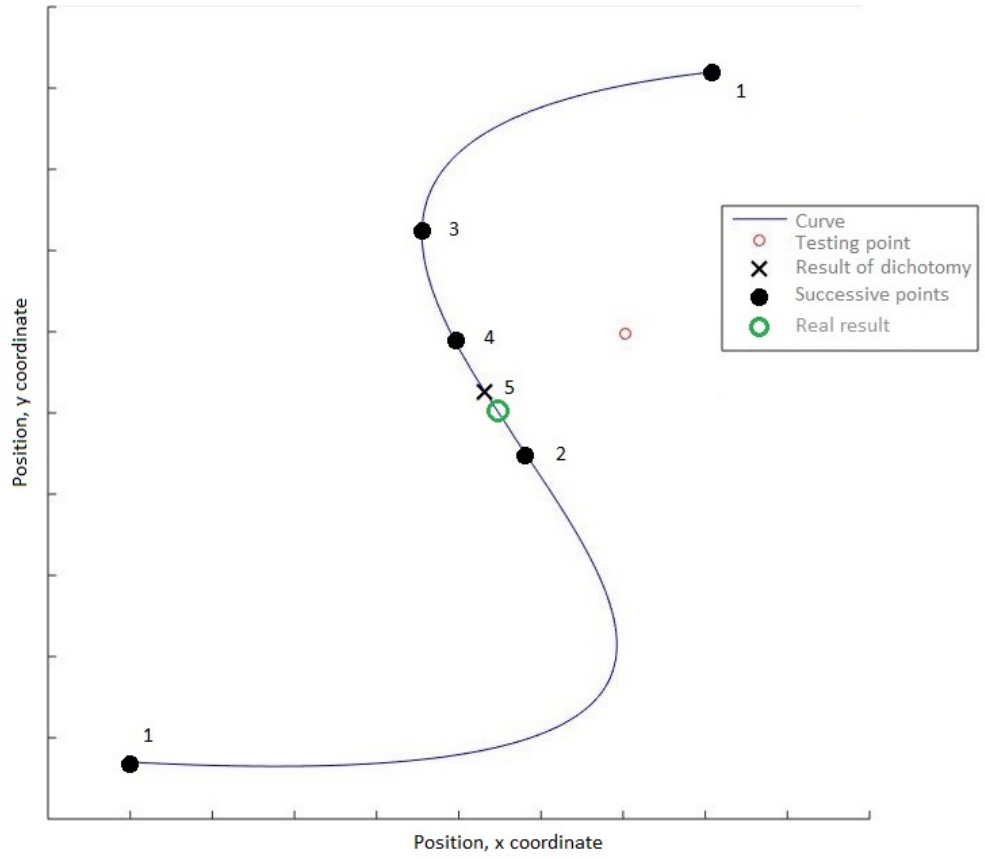


Figure 27: Example of dichotomy algorithm with 5 iterations.

In this example (Figure 27), we can see that this method converges very quickly to the result, even if one or two more computations would have given a good precision in this example.

Algorithm 3 Dichotomic search

input : continuous correction: $x_{xyz}(t)$, testing point: x^*_{xyz}

$N = 40$ #N can be given as an input
$sublimit1 = 0$
$sublimit2 = 1$ #sublimit1-2 are the boundary of the sub-interval

for i *from* 1 *to* N **do**
  if $distance(x_{xyz}(sublimit1), x^*_{xyz}) < distance(x_{xyz}(sublimit2), x^*_{xyz})$ **then**
    $sublimit2 = (sublimit1 + sublimit2)/2$ #sublimit2 is further, need to be changed
  else
    $sublimit1 = (sublimit1 + sublimit2)/2$ #sublimit1 is further, need to be changed
  end if
end for

output: the closest point: $x_{xyz}(sublimit1)$

Advantage

This algorithm can be very precise, it has a very low computational cost, $O(n)$ where n is the number of iteration (in this implementation $O(n)$ but with binary search it is $O(log_2(m))$ where m is the length of the array). Typically, for this kind of application you could choose $n = 40$.
Even if the computational cost is the same for the brute force algorithm, it is said here to be low because of the small required value of n.

Disadvantage

This algorithm can be precise, but can also give a completely wrong result, it detects only local minimal which is a problem, the minimal distance point of all the curve is needed here, see Figure 28.

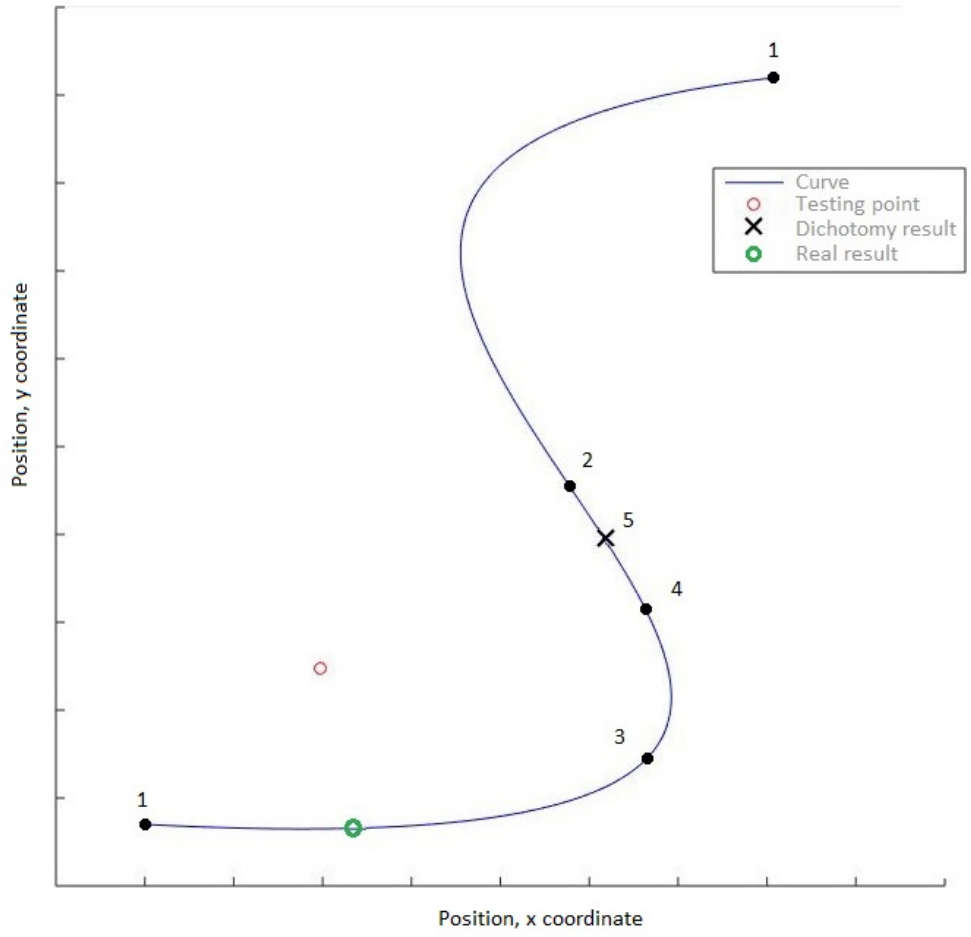


Figure 28: Example of dichotomy algorithm with 5 points which fail.

When testing this method, we figure out that this problem occurs quite often.

4.4.3 Analytic solution

The continuous curve is a spline defined by many cubic splines on each axis in the following form (the robot environment is in 3 dimensions so there are 3 splines parametrized by a t-parameter, $t \in [0, 1]$)

$$\begin{aligned} f_x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ f_y(t) &= a_y t^3 + b_y t^2 + c_y t + d_y \\ f_z(t) &= a_z t^3 + b_z t^2 + c_z t + d_z \end{aligned} \tag{13}$$

Maths

The distance from a point (x^*, y^*, z^*) to another point on the curve is:

$$distance(t) = \sqrt{(x^* - f_x(t))^2 + (y^* - f_y(t))^2 + (z^* - f_z(t))^2}$$

The minimal distance is found by deriving:

$$\frac{d(distance(t))}{dt} = \frac{-2f'_x(t)(x^* - f_x(t)) - 2f'_y(t)(y^* - f_y(t)) - 2f'_z(t)(z^* - f_z(t))}{\sqrt{(x^* - f_x(t))^2 + (y^* - f_y(t))^2 + (z^* - f_z(t))^2}} = 0$$

$$\frac{d(distance(t))}{dt} = -2f'_x(t)(x^* - f_x(t)) - 2f'_y(t)(y^* - f_y(t)) - 2f'_z(t)(z^* - f_z(t)) = 0$$

We focus now on $S_x = -2f'_x(t)(x^* - f_x(t))$ because the calculus are the same for dimension x, y and z.
We know that:

$$\begin{aligned} f_x(t) &= a_x t^3 + b_x t^2 + c_x t + d_x \\ f'_x(t) &= 3a_x t^2 + 2b_x t + c_x \end{aligned}$$

so:

$$S_x = -2f'_x(t)(x^* - f_x(t)) = -2(3a_x t^2 + 2b_x t + c_x)(x^* - a_x t^3 - b_x t^2 - c_x t - d_x)$$

After factorization:

$$S_x = 6a_x^2 t^5 + 10a_x b_x t^4 + (8a_x c_x + 4b_x^2)t^3 + (6a_x d_x + 6b_x c_x - 6a_x x^*)t^2 + (4b_x d_x + 2c_x^2 - 4b_x x^*)t + (2c_x d_x - 2c_x x^*)$$

$$S_x + S_y + S_z = 0$$

Since there are 5 roots to this polynomial, we compute the solution by removing complex roots, and finally taking the closest root of the few remaining.

Result interpretation

This method is very powerful, as an analytically algorithm, its computational cost is low and constant and, by definition, the most accurate possible.

The result will be the absolute closest point on the curve to the testing point, but it will not necessarily be in the interval $t \in [0, 1]$, depending on the testing point. This is not a problem, it is very simple to test if the result point is in the interval, and if not, it means that the trajectory is not going through the given correction, see Figure 29.

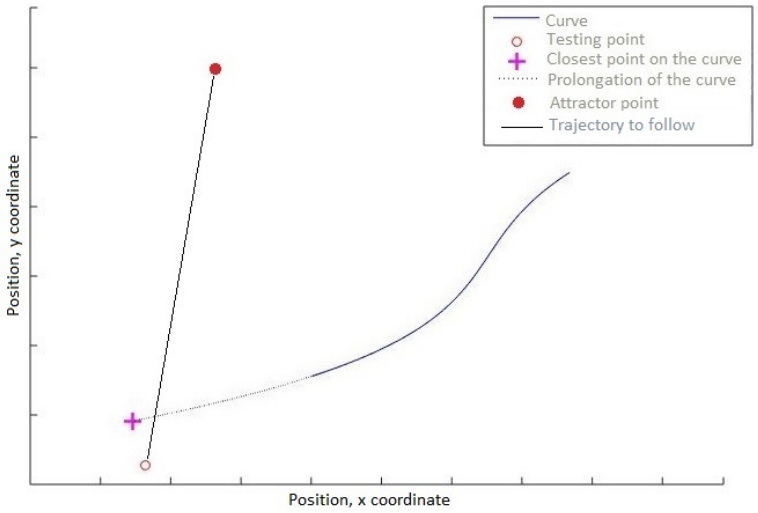


Figure 29: Example of a trajectory which is not going through the correction and therefore shouldn't be influenced by the curve.

In this particular case, the testing point is set on purpose somewhere where the straight trajectory wouldn't cross the correction, so the trajectory from the testing point to the attractor shouldn't be influence by the correction. We can see that the result of the closest point algorithm returns a value (pink +) that is not on the correction, this is the absolute closest point of the analytic curve. This result is not a problem, indeed it's very easy to test if the result is located on the correction and if not, the point is considered to not be influenced by the correction.

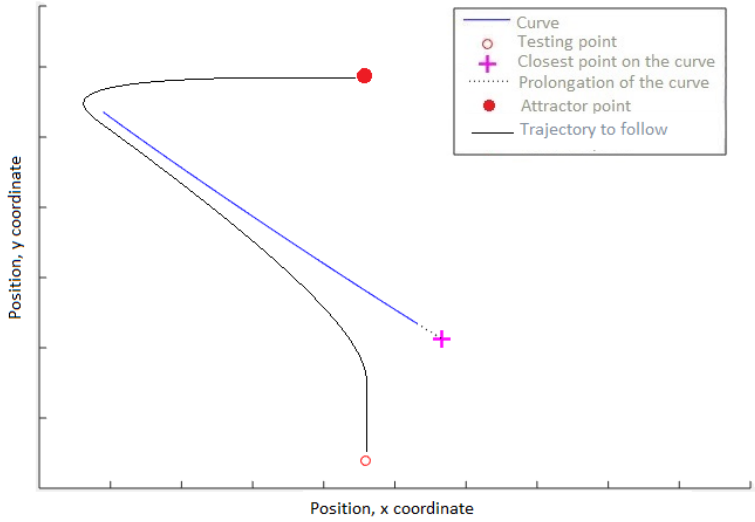


Figure 30: Example of a trajectory which was going through the correction but still out of the interval.

In this example Figure 30, the testing point is set on purpose somewhere where the straight trajectory would cross the correction but where the distance-point algorithm would return a value out of the correction. It's also not a problem, as we can see in black, the trajectory goes forward until the closest point reach the correction, and the GPR will have an effect before that the trajectory cross the correction.

For its constant computational cost and very good accuracy, the analytic algorithm will be chosen for the continuous Gaussian process regression computation. This algorithm will be used in the quantitative analysis.

The correction is usually composed by many splines which is composed by many polynomial functions, therefore, the algorithm will be executed many times. One closest point is compute per polynomials functions and the final closest point is the closer one.

4.5 Analysis

To analyze the result of this adapted algorithm, some tests will be done, all the time by comparing the discrete and the continuous Gaussian Process Regression.
First we will analyze the accuracy by checking if the new algorithm return a good result with a low number of training points.
Then we will analyze the computational cost by varying the number of testing points and the number of training points.

4.5.1 Accuracy of continuous Gaussian Process Regression

To compare the influence of the continuous and discrete GPR, we took a correction made by 3 points, and tested its influence on some testing points.

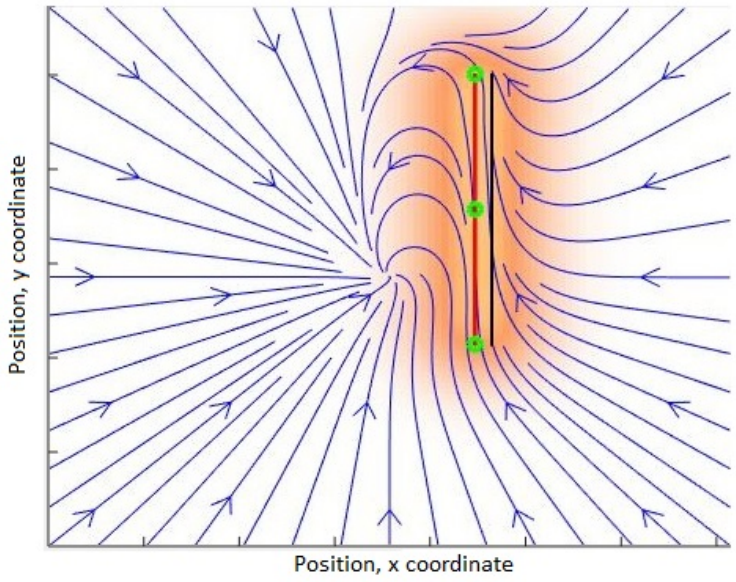


Figure 31: Test for continuous and discrete GPR.

For this test (see Figure 31), 3 training discrete points were taken (green circle), a spline was computed (in red) that will be the training continuous data and many testing points were chosen on a parallel line (in black) so that the influence of the curve on the testing points will be very clear.

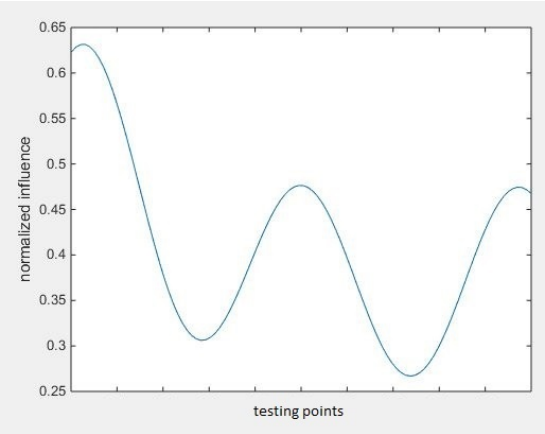


Figure 32: Graph of the normalized influence of the discrete Gaussian process on the test points.

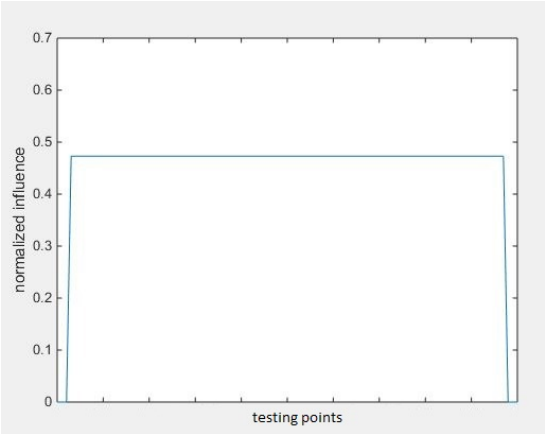


Figure 33: Graph of the normalized influence of the continuous Gaussian process on the test points.

As we can see on Figure 32, only the 3 training points influence the result, that's why there are 3 squared-exponential in the graph. Typically a trajectory could go through the correction when the influence is the lowest (see Figure 24). On Figure 33, the influence is constant, no trajectory will be able to cross the correction. This is exactly what the continuous Gaussian Process Regression was developed for.

This test was done to prove that the influence $|GP|$ is not local to the knot points of the correction but is a general influence of the entire correction curve, due to the continuous GPR.

4.5.2 Computation time

To compare the computational cost of both algorithms, first we varied the number of test points and then the number of training points.
Algorithms were simulated on a computer, the time was numerically computed. The same system as below (Figure 31) is used.

Testing points number variation

First, the computationnal cost of the GPRs algorithms are tested by having the number of testing point varying. As in Figure 31, there is a vertical line on which many testing points linearly spaced were computed.

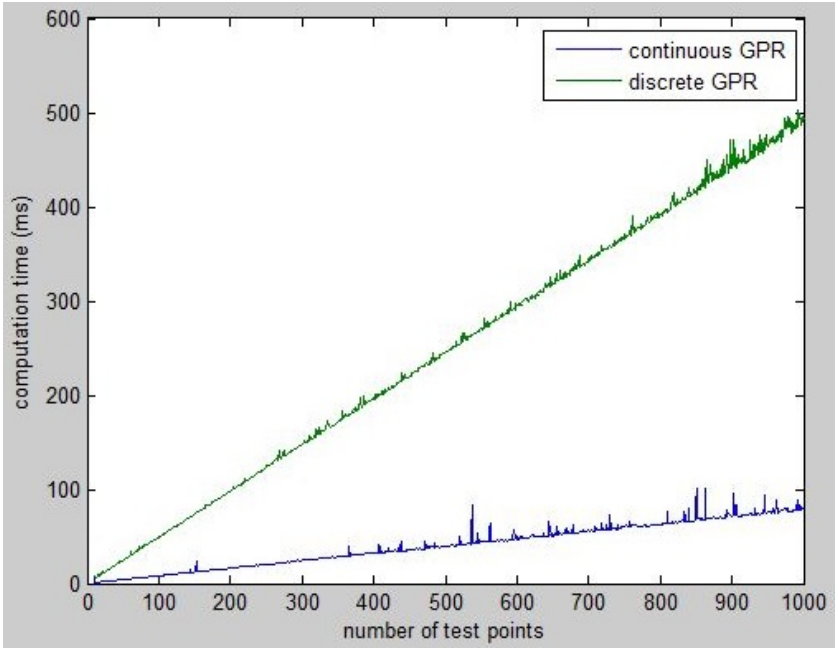


Figure 34: Graph of computational time for GPR algorithms. Testing points vary with 3 training points.

Both algorithm computational times increase linearly, but for the continuous algorithm, it's increasing slower and the values are almost 10 times smaller.

Training points number variation

The computationnal cost of the GPRs algorithm are tested by having the number of training point varying. As in Figure 31, there is a vertical line on which many training points linearly spaced were computed. Another parameter was tested here, the smoothing parameter. Indeed, while using the robot to get data, the robot is sending a lot of data very fastly and the algorithm gets very quickly slow. That's why a decimation algorithm has been used to have a lower number of training point but still a good representation of the correction.

A coefficient of decimation called smoothing was used, it works very simply: if equals to 1, we keep all the data, if equals to 0.5, we keep half of the data (every two points, one is removed) and the same kind of behavior for every values between 0 and 1.

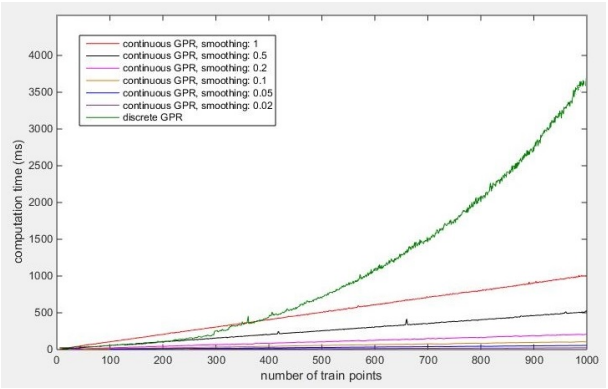


Figure 35: Graph of computational time for discrete and continuous GPR algorithms on 100 testing points with a smoothing of 1, 0.5, 0.2, 0.1, 0.05, 0.02.

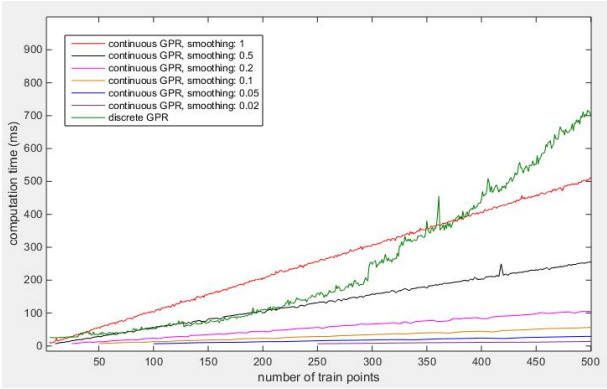


Figure 36: Zoom on the 500 first points of Figure 35.

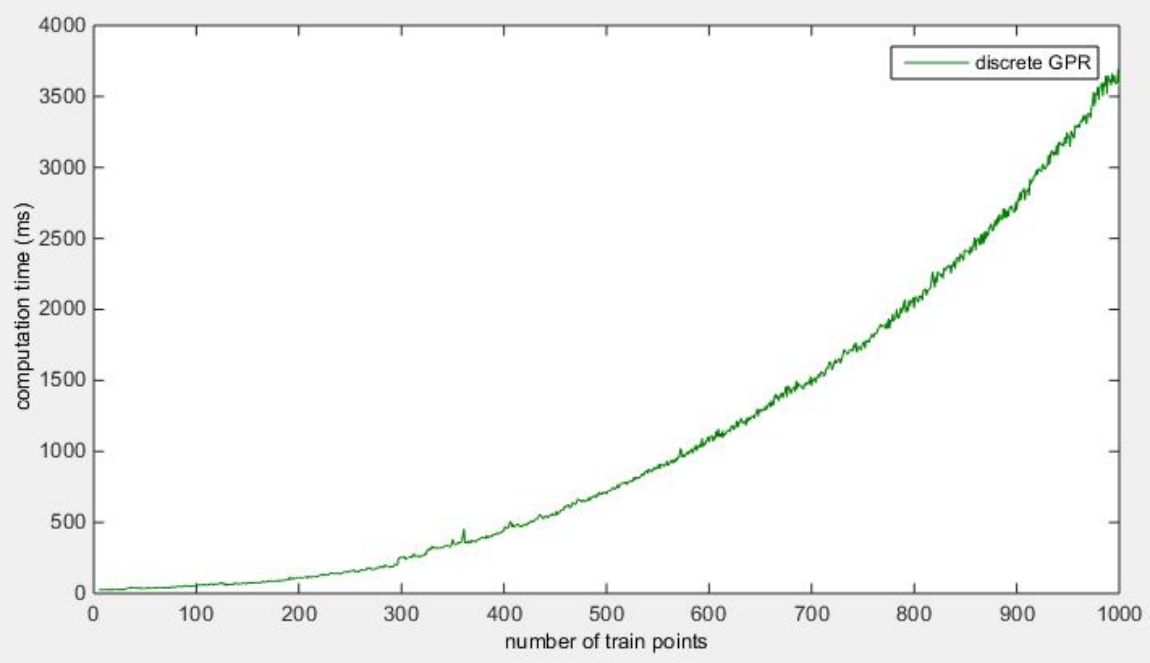


Figure 37: Figure 35 with only the discrete GPR curve.

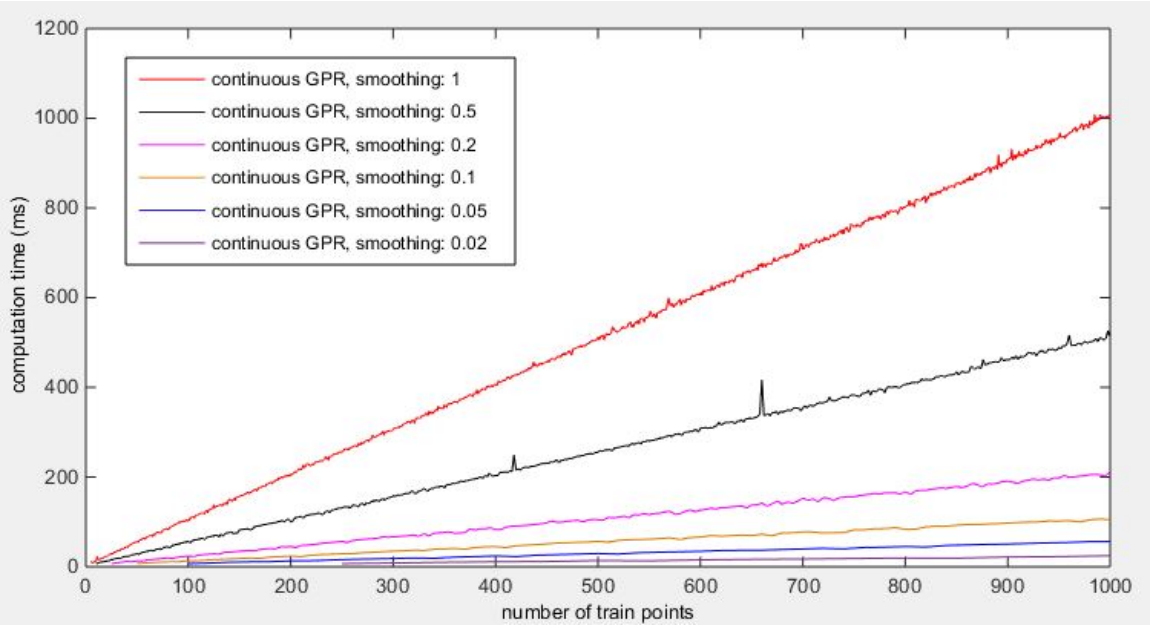


Figure 38: Figure 35 with only the continuous GPR curves.

On Figure 37 we can see the computation time of the discrete GPR algorithm vary quadratically.
The general shape of the curves of the continuous algorithm on Figure 38 are linear. Indeed, if there are many training points, there will be many cubic splines (one between each training point), so the distance algorithm will have to compute the minimal distance point on each cubic splines, and then took the closest one. Even if this algorithm has a constant computational cost for a single cubic splines, the continuous GPR will take linearly more times to be compute. The most the smoothing goes to zero, the faster it will be to compute the continuous GPR.

All those tests were done on a computer, the computational time is not a perfectly linear, quadratic or constant signal because there are always many tasks running at the same time which affects randomly the time values (indeed, we are measuring in milliseconds scale, it is very sensitive). Tests were done in the best possible conditions with the least amount of parallel tasks possible to minimize the problem.

Those analyses show that the continuous GPR gives better result for high number of training points, but the standard GPR is better with a low number of training points. This is directly linked to the inversion of the $N \times N$ matrix (if N is low the inversion will be fast) and the constant computationnal cost of the analytic distance algorithm.

5 Conclusions

The aim of this project is to allow the robot to be able to learn movement from demonstrations.
In this report, we presented the trajectory processing componenent which is the transformation of a demonstration to simple commands on how the robot need to adapt its trajectory.

Here are some steps of the process, the operator applies a kinestethic correction on the robot while doing a movement. This demonstration is provided to an algorithm which isolates the correction using the data (velocity and original trajectory), then transforms the correction into a continuous curve (cubic splines) and finally, when moving again near the same place where the correction was applied, it gets aligned to the shown correction. A continuous Gaussian Process Regression is used by computing a distance from the actual position of the robot to the correction curve.

In this project, algorithms were developed on Matlab and implemented on the robot using ROS library in Python. The codes can all be found in a private repository:
https://github.com/epfl-lasa/nl-dynamics

Discussion

The main developement towards the robotics in this project concerns the continuous Gaussian Process Regression adaptation. Indeed this algorithm provides a robust data estimation solution from a continuous representation of data. It is used and adapted for splines in this application, however it could be generalized for any continuous representation of data. Indeed, the key of this algorithm is the minimal distance from a point to the curve computation, which could be generalized by a numerical solution.

Work on the robot

Finally we tested everything, and the robot learned and moved. Cyril Schmitt has also been working on this project but on the vocal part. When assembling both of our projects, interaction, movement and learning processes of the robot work and gave good results. We were able to control the robot using natural language, and after applying correction and running the learning algorithm, the robot corrected its trajectory itself, see Figure 39, this graph data's directly comes from the robot. In a 3D space it successfully isolates two corrections from a demonstration.

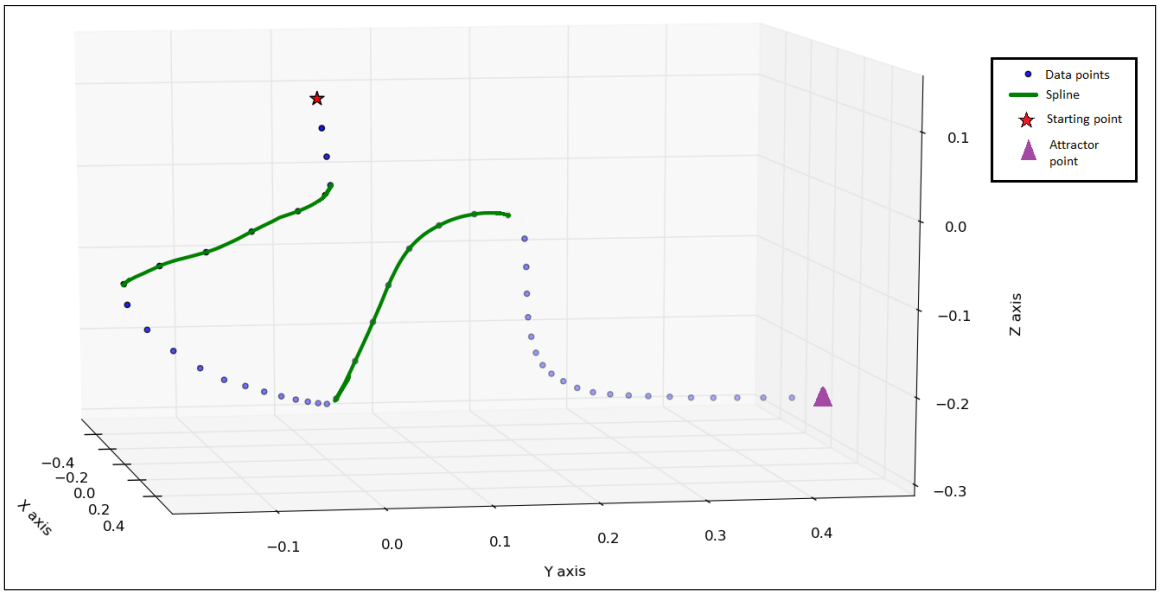


Figure 39: 3D graph of real robot's data. There are two corrections in the demonstration transform in a continuous curve (spline) in green. The starting point is the red star and the attractor point is the purple triangle.

Future work

Some more tests could have been done for the analysis of the isolation correction algorithm, indeed the precision factor of this algorithm can be tuned more precisely. Also, some improvements could be done such as optimization of the algorithms (the research of the closest point on many cubic splines for example) or developping a concept of showing many times the same correction to the robot to have a better learning, algorithms like putting together many corrections in a 3D space would be developped. However, the project is working and the only future work would concern optimization.

6 Acknowledgments

I would like to thank my supervisors Felix Duvallet and Klas Kronander. Both of them were very available and helped me for this project. I learned a lot and appreciated working in the LASA lab. I would also like to thank Professor Aude Billard for her encouragement and advice.

References

[1] Carl Edward Rasmussen and Christopher K. I. Williams, the MIT Press *Gaussian Processes for Machine Learning* http://www.gaussianprocess.org/gpml/ 2006.

[2] ROS documentation, Open source Robotics Foundation http://wiki.ros.org/.

[3] Fourier series interpolation for Equation 2, Wikipedia https://en.wikipedia.org/wiki/Trigonometric_interpolation.

[4] Cyril Schmitt *Simplifying user/robot interface by using speech commands*, EPFL, 2016.

[5] Klas Kronander, *Gaussian Process Regression on Curves*, Technical paper, EPFL, 2016.

[6] Sky McKinley and Megan Levine, *Cubic Spline Interpolation*, http://web.archive.org/web/20090408054627/http://online.redwoods.cc.ca.us/instruct/darnold/laproj/Fall98/SkyMeg/Proj.PDF