Generating Calligraphic Trajectories with Model Predictive Control

Daniel Berio*
Goldsmiths, University of London

Sylvain Calinon†
Idiap Research Institute

Frederic Fol Leymarie‡
Goldsmiths, University of London

ABSTRACT

We describe a methodology for the interactive definition of curves and motion paths using a stochastic formulation of optimal control. We demonstrate how the same optimization framework can be used in different ways to generate curves and traces that are geometrically and dynamically similar to the ones that can be seen in art forms such as *calligraphy* or *graffiti art*. The method provides a probabilistic description of trajectories that can be edited similarly to the control polygon typically used in the popular spline based methods. Furthermore, it also encapsulates movement kinematics, deformations and variability. The user is then provided with a simple interactive interface that can generate multiple movements and traces at once, by visually defining a *distribution* of trajectories rather than a single one. The input to our method is a sparse sequence of targets defined as multivariate Gaussians. The output is a dynamical system generating curves that are natural looking and reflect the kinematics of a movement, similar to that produced by human drawing or writing.

Index Terms: Computer Graphics [I.3.3]: Picture/Image Generation—Line and curve generation; Computer Graphics [I.3.6]: Methodology and Techniques—Interaction techniques; Computer Applications [J.5]: Arts and Humanities—Fine arts

1 INTRODUCTION

The hand drawn curves that can be seen in certain art forms such as calligraphy and graffiti art are often the result of skillful and expressive movements that require years to master [29]. The resulting traces often possess subtle qualitative features which are difficult to reproduce through traditional computer graphics methods such as B-splines or Bézier curves. We adopt the hypothesis that some of the aesthetic qualities observable in hand drawn traces are closely associated to the way such traces were created, and in particular with respect to their dynamic properties (i.e. velocity, acceleration, and associated curving behaviour). Such an hypothesis is commonly held amongst visual artists, and is further supported by academic work in theories of art history [17, 34], as well as by studies stemming from the fields of psychology and neuroscience. Such studies indicate that the visual perception of marks made by a drawing hand trigger activity in the motor areas of the brain [18, 26], and further induce an approximate mental recovery of the movements and gestures underlying the artistic production [19, 32].

In computer graphics applications requiring the simulation of hand drawn traces, it then looks advantageous to follow a *movement centric* approach, in which a shape is defined by the movement underlying its production rather than by an explicit definition of its geometry. We argue that doing so simplifies the process of computationally capturing the inherent qualities of human made traces, such as smoothness and variability, which stem from the properties of the motor system and from the type of movements used when drawing. In this paper, we explore this approach with a trajectory generation methodology based on *stochastic optimal control*.

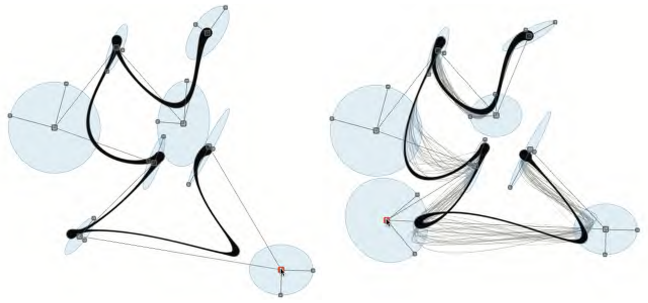

Figure 1: **Left:** Interactively editing of a trajectory through the manipulation of Gaussians. **Right:** interactive editing and visualisation of the trajectory distribution (gray).

Optimal control consists of a family of methods aimed at determining the trajectories of a dynamical system that minimise a given cost or performance criterion [10]. In our study we solve the optimal control problem numerically with a stochastic formulation of *Model Predictive Control* (MPC), a technique that is popular in robotics and industrial related applications. While MPC is typically used in a control system setting, we demonstrate how it can be used as a flexible curve generation tool for computer graphics applications. The control system produces trajectories that have desirable smoothness properties, while sharing common ground with conventional curve generation methods [13].

We formulate the optimisation objective as a discrete sequence of targets that are defined probabilistically in the form of multivariate Gaussian distributions. The centres of each Gaussian define a coarse *control-polygon* which permits interactive manipulation (Fig. 1), behaving similarly to traditional spline/polynomial interpolation methods. In addition, the covariances explicitly describe the *variability* of different parts of the movement, as well as to describe curvilinear features that take into account human movement *coordination* [8]. The optimisation process results in smooth curves and motion paths that are, by design, dynamically and kinematically similar to the ones that would result from a human movement.

The contributions of this paper are two-fold. First, we describe a flexible and interactive trajectory generation method that can be used to synthesise curves for computer graphics applications, while also being able to generate motion paths for computer animation or robotics applications. Second, we show that this methodology is particularly useful for applications requiring the simulation of *human made artistic traces*. We focus on examples involving the production of letter forms as the ones that can be seen in traditional calligraphy as well as contemporary graffiti (street) art. We show that our method is well suited to capture the visual features of such traces, with the additional benefits of facilitating a realistic animation of the trajectory evolution, and providing rich dynamic information that can be exploited to facilitate expressive rendering methods.

The rest of the paper is organised as follows. The next section gives a brief background on related work and further explains the concepts that form the basis for this study. In Sect. 3 we describe the optimisation framework used for the trajectory generation meth-

*e-mail: d.berio@gold.ac.uk
†sylvain.calinon@idiap.ch
‡e-mail:ffl@gold.ac.uk

16-19 May, Edmonton, Alberta, Canada

ods, which are then outlined in Sect. 4. These include a stochastic definition of trajectories Sect. 4.1, which can also be reformulated to describe interpolating trajectories (Sect. 4.2), as well as to approximate Bézier curves (Sect. 4.3). Finally, in Sect. 5 we describe how our system can be used in an interactive setting for the purposed goal of generating instances of synthetic calligraphy. Additional mathematical and implementation details are given in the Appendices.

2 Related work and Background

While movement synthesis has not been commonly used for the synthesis of artistic imagery (e.g. see [24]), a few examples exist that have proposed approaches that are related to our method. Haeberli implemented DynaDraw, a computer program that allows the user to interactively generate strokes evocative of calligraphy by simulating a mass attached to the mouse position [20]. House et al. [22] as well as Liu et al. [25] generate sketch based renderings by using a *Proportional Integral Derivative* (PID) controller, which controls the evolution of a 2nd order system describing the trajectory of a pen. Shinoda et al. minimize the jerk (time derivative of acceleration) of calligraphic traces defined as B-Splines in order to mimic the effect of running-style Japanese calligraphy [36]. AlMeraj et al. [1], use the minimum-jerk model of human reaching movements [16] to mimic the visual qualities of hand drawn lines. With a motivation similar to ours, Berio and Leymarie [3] apply the Sigma Lognormal handwriting model [33] to interactively generate variations of graffiti like traces. Similarly to our method, letter forms can be defined interactively by defining a control polygon made of a sparse number of targets. In this communication, we propose a method based on stochastic optimal control that, in its different formulations, is applicable to the same type of problem domains as the ones tackled in the previously mentioned studies. The main advantages of our method is that: (i) it encompasses variability and coordination at the descriptive level; and (ii) its generality and flexibility allow a user to experiment with different types of trajectory generation methods and dynamical systems with a single framework.

Typically, hand drawn curves are interactively specified by using a sketch based interface, in which a user traces a curve with a trackpad, mouse or tablet, and the trace is then processed to remove discontinuities and imperfections caused by the digitising device. This process is referred to as "beautification", "neatening" or "fairing", and has been implemented in a number of methods, e.g. [27, 28, 41]. We propose an alternative method to the definition of hand-drawn curves, in which a user defines a coarse series of targets with a point-and-click procedure and a control system defines a trajectory that tracks the targets, resembling traces produced by a human movement.

In this study, we rely on principles that have been observed in the movement science and handwriting analysis/synthesis domains. The velocity profile of rapid and straight reaching motions are characterised by "bell shaped" velocity profiles [30]. Such bell shaped velocity profiles have been modeled with a variety of techniques, including sinusoidal functions [31], Beta functions [5], optimisation methods [16], and lognormals [33]. Many studies propose that smooth arm motions can be described as the space time superposition of a number of "ballistic" movement primitives [40], which are commonly referred to as *strokes*. Each stroke can be represented with the characteristic bell-shaped speed profile. Complex movements tend to show an inverse power relationship between absolute curvature and speed, where curvature extrema usually correspond to minima in the speed profile [44] . Various experiments have exposed the tendency of humans to keep the time of movements relatively independent across different size ranges, aka *isochrony* [44]. Isochrony can be *global*, i.e. for movements and trajectories as a whole, or *local*, for parts of a movement [23]. Complex hand and arm motions tend to be executed in a smooth manner with trajectories that seem to minimise a cost or performance objective [15]. Our method shares relations with a number of models that formulate this objective with the minimisation of the square magnitude of higher order derivatives of position, such as the minimum *jerk* (3rd order) [16], minimum *snap* (4th order) [12] and minimum *crackle* (5th order) [11] models. Humans movements show inherent variability [4], where the variability tends to be higher in parts of a movement that are not critical to the required precision of a task, which is known as the *minimal intervention principle* [42]. Our method is consistent with this principle, and allows a user to explicitly define the required task precision in different parts of a movement through the manipulation of covariances.

We can identify in the literature two dominant representations that are used to describe the spatial evolution of hand-movement paths: *virtual targets* and *via-points*. Virtual targets imply a ballistic stroke representation of movement, and describe the "imaginary" loci at which each stroke is aimed. As a result, these positions do not directly lie along the corresponding motion path. Such a representation has been adopted in a variety of models of handwriting, e.g. [5, 33]. As the name implies, via-points represent landmark positions *along* the trajectory, which are the basis for a number of optimisation-based models of movement (e.g. [12, 16]. In this paper we describe a representation that functions as a "hybrid" between via-points and virtual-targets, in which the evolution of a trajectory is encoded in terms of *multivariate Gaussians*. This hybrid representation is capable of describing ballistic virtual target positions, via-points, as well as more complex spatial constraints, such as forcing a movement to pass through a narrow region of space.

The proposed method also shares relations with interpolating and smoothing splines. Egerstedt and Martin [13] discussed the equivalence between several forms of splines and control theoretic formulations of dynamical systems. The authors cover the case of interpolating splines in which the curve passes through user-defined keypoints, as well as smoothing splines in which a trade-off is found between curve smoothness and curve fitting. It is shown that smoothing splines correspond to the output of a controller found by minimizing quadratic cost functions similar to the ones used in our method. Our method extends this principle to a more generic case, in which each covariance matrix encodes the precision as well as coordination patterns in the movement.

3 Optimal control framework

Model Predictive Control (MPC) encompasses a series of numerical methods used to predict the behaviour of a dynamical system, and compute a series of optimal feedback or feedforward commands that will minimise a given cost function and constraints over a given time horizon. In a typical control setting, only the first optimal command is fed to the system, and the optimisation process is repeated iteratively by shifting the time horizon forward to the next time step. As a result, MPC is also commonly referred to as *receding horizon control*. Because our application is focused on curve/trajectory generation rather than control, we perform only a single optimisation step with a time horizon that corresponds to the duration of the trajectory as a whole. From a control perspective, this corresponds to the assumption of a perfect reproduction of trajectory and no external disturbance. As we shall show, this assumption allows us to exploit MPC as a flexible motion and curve synthesis tool.

For the task at hand, we use the simplest form of MPC, such that the objective is unconstrained, the system is linear and the cost function is quadratic. In this case the problem is equivalent to the control problems known as discrete Linear Quadratic Tracking (dLQT). While MPC is based on a well understood control-theoretical background, in our application we focus on a step by step description, which follows and is complemented by mathematical and implementation details in the Appendices. Thanks to the availability of powerful linear algebra libraries and software packages, the method can be implemented in a straightforward manner by following the equations described in the text. For more detailed theory and deriva-

tions, the interested reader is referred to [7].

3.1 Dynamical system

We model the movement of a pen trajectory by optimising the evolution of a discrete linear time invariant (dLTI) system of order n, which results in a discrete sequence of positions $\boldsymbol{x}_t \in \mathbb{R}^D$, where the index t denotes the t-th time step. The dynamical system is described with the state space representation

$$\boldsymbol{\xi}_{t+1} = \boldsymbol{A}\boldsymbol{\xi}_t + \boldsymbol{B}\boldsymbol{u}_t, \tag{1}$$

where the system state

$$\boldsymbol{\xi}_t = \left[\boldsymbol{x}_t^\top, \dot{\boldsymbol{x}}_t^\top, \ldots, \overset{(n-2)\top}{\boldsymbol{x}_t}, \overset{(n-1)\top}{\boldsymbol{x}_t}\right]^\top \in \mathbb{R}^{nD}, \tag{2}$$

is given by the position concatenated with its derivatives up to the order $n-1$. The dynamics of the system are determined by the matrices $\boldsymbol{A} \in \mathbb{R}^{nD\times nD}$ and $\boldsymbol{B} \in \mathbb{R}^{nD\times D}$, which fully describe the response of the system to an input command $\boldsymbol{u}_t$. The methods described in this paper function for higher dimensions, but for the scope of this study we focus on planar trajectories, so we assume $D = 2$. While the optimisation framework we will describe can function with arbitrary linear systems, in the examples demonstrated here, we use a chain of n integrators that is controlled by its nth order derivative. For example, a system of order 2 will correspond to a system controlled with acceleration commands. The reader is referred to Appendix A for further mathematical details.

3.2 Quadratic cost

An optimal trajectory is computed by minimising a quadratic cost function that, for each time step, tries to reduce deviations from a reference state sequence while keeping the amplitude of the control commands low. For a trajectory of N time steps, the cost is given by

$$J = \sum_{t=1}^{N} \left(\hat{\boldsymbol{\xi}}_t - \boldsymbol{\xi}_t\right)^\top \boldsymbol{Q}_t \left(\hat{\boldsymbol{\xi}}_t - \boldsymbol{\xi}_t\right) + \sum_{t=1}^{N-1} \boldsymbol{u}_t^\top \boldsymbol{R}_t \boldsymbol{u}_t, \tag{3}$$

where for each time step, $\hat{\boldsymbol{\xi}}_t$ is the desired state (position and optionally consecutive order derivatives) and $\boldsymbol{Q}_t \in^{nD\times nD}$ and $\boldsymbol{R}_t \in^{D\times D}$ are positive semidefinite *weight* matrices. The weight matrices respectively define the tradeoff between tracking accuracy (*state cost*) and limiting the amplitude of the control commands (*control cost*). In our application, we keep the control cost fixed with a diagonal regularization term $\boldsymbol{R}_t = r\boldsymbol{I}$, where larger values of r produce smoother trajectories.

The optimisation problem can be solved either: (i) in a *batch* form, by solving a large regularized least squares problem; or (ii) *iteratively*, by using dynamic programming and resulting in the a series of time-varying gains. While the latter method is faster, the batch approach directly provides a probabilistic interpretation of the result, which we exploit for stochastic sampling. For implementation details of both methods, the reader is referred to Appendix B.

4 Trajectory Generation

In the following paragraphs we describe different trajectory generation methods that can be achieved with MPC. All the proposed methods run at interactive rates and produce trajectories that are smooth and are kinematically similar to the ones that can be seen in movements made by a human when drawing or writing. The control term of the cost function enforces a smooth and continuous trajectory, regardless of continuity of the desired state sequence. As a result, it is possible to define the reference as a sparse sequence of states. This results in a concise and easily manipulable representation of the trajectory geometry that can be exploited in a user interaction scenario.

4.1 Minimal intervention trajectories

It has been demonstrated that the combination of MPC with a probabilistic representation can be exploited to capture the variability of multiple human demonstrations in a human-robot interaction scenario [9]. Here, we show how a similar approach can be exploited for the task of trajectory and curve generation.

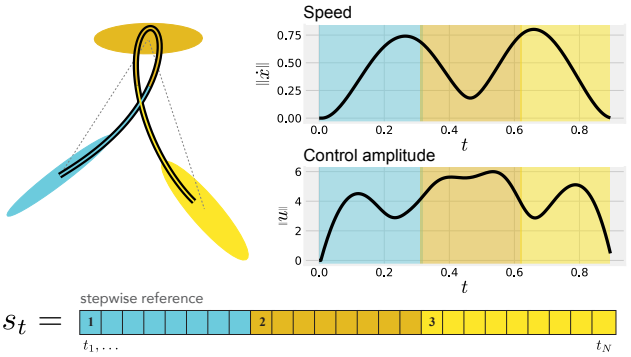

Figure 2: 4th order trajectory generated with Gaussian targets, and the corresponding speed and command (snap) magnitude profiles. Below, schematic visualisation of the state vector for a stepwise reference. The duration of each state is color coded with the corresponding Gaussian.

We describe a trajectory with an ordered sequence of m states, where each state is defined with a multivariate Gaussian distribution $\mathcal{N}_D(\boldsymbol{\mu}_i, \boldsymbol{\Sigma}_i)$. This representation can be seen as a ballistic decomposition of the movement in $m-1$ strokes, where each stroke i is aimed at the center of the $(i+1)$th Gaussian. With an assumption of perfect *local isochrony* (i.e. a fixed duration for each state), we define a state vector $\boldsymbol{s} \in \mathbb{Z}^N$ in which each consecutive state is indexed N/m times in a stepwise manner (e.g. $\boldsymbol{s} = \{1, 1, 2, 2, 2, 3, \ldots, m\}$, see Fig. 2). The reference state sequence and tracking weights for the whole optimisation horizon are then given by

$$\hat{\boldsymbol{\xi}}_t = \boldsymbol{\mu}_{s_1} \forall t \quad \text{and} \quad \boldsymbol{Q}_t = \boldsymbol{C}^\top \boldsymbol{\Sigma}_{s_t}^{-1} \boldsymbol{C} \text{ for } t < N, \tag{4}$$

where the $\boldsymbol{C}$ is the *sensor* matrix, the block entries of which determine what components of the state should be considered in the optimisation objective. For this use case, we only consider the position components, which is described with a sensor matrix

$$\boldsymbol{C} = [\boldsymbol{I}, \boldsymbol{0}, \ldots, \boldsymbol{0}] \in \mathbb{R}^{D\times nD}. \tag{5}$$

This produces zero entries of $\boldsymbol{Q}_t$ for the state derivative terms, which are consequently ignored in the cost function[1]. For the last time step, we set the matrix $\boldsymbol{Q}_N$ identically to Equation 5, but we then add a large constant diagonal value to the derivative terms of matrix that corresponds to a high precision and low variance (we used 1×10^{10} in our examples, which for our use case provided consistent results across different system orders without numerical issues). This enforces a zero boundary condition on the state derivatives and brings the movement to a smooth stop.

With this formulation the tracking weights are defined in terms of *required precisions*, and the penalty of deviating from a given state is given by the Mahalanobis distance to the centre of the corresponding Gaussian. The resulting trajectory formation method is therefore consistent with the minimal intervention principle [42]. Each Gaussian functions as a *stochastic target*: as the variance of the Gaussian approaches zero, it increasingly constrains the trajectory to pass

[1] Note that a sensor matrix with all block entries set to $\boldsymbol{I}$ would correspond with an optimisation objective that takes all state derivatives into consideration.

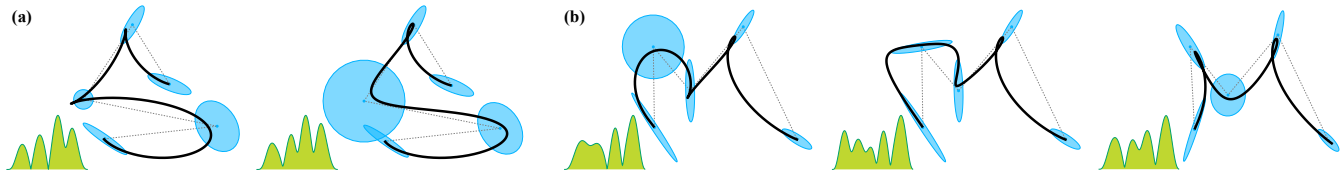

Figure 3: **(a)**, smoothing effect of increasing the variance of a Gaussian. **(b)**, manipulating the trajectory evolution with full covariances. Below each trajectory, its corresponding speed profile.

through its location, effectively behaving like a via-point. A higher variance produces a smoothing effect (similar to smoothing splines), resulting in a behavior that is similar to a virtual target (Fig. 3a). In addition, using full covariances can be exploited to mimic calligraphic effects by allowing the definition of *coordinations* and *directional trends* in the trajectory Fig. 3b).

4.1.1 Stochastic sampling

If we consider a Bayesian formulation of the batch version of the minimisation problem, we can interpret the generated trajectory as the center of a *trajectory distribution* [8]. We can then stochastically sample this distribution in order to generate a possibly infinite number of variations over the mean trajectory (Fig. 4, mathematical details in Appendix C). While variations could also be achieved by randomly perturbing the means and covariances of each state, this method allows the user to define the spatial evolution, as well as the variability of the trajectory within a single compact representation. This property is useful for the intended goal of generating calligraphic traces, since it mimics the variability that can be observed in human handwriting, calligraphy and drawing.

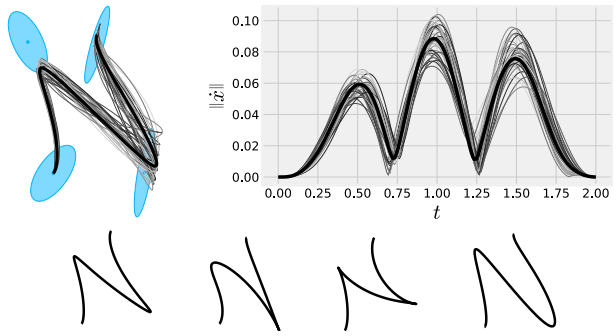

Figure 4: Stochastic sampling of the trajectory distribution for a letter "N". **Top-left:** the mean trajectory (black), random samples from the trajectory distribution (gray) and the corresponding Gaussians. **Top-right:** The corresponding speed profiles. **Bottom:** Selected random samples from the trajectory distribution.

4.2 Interpolation with MPC

With a slightly different formulation of the same optimisation framework, we can generate various types of interpolating trajectories. For this task, we define a series of *key points*[2] $\{\boldsymbol{v}_i\}_{i=1}^{m}$ and the corresponding time steps $\{t_i\}_{i=1}^{m}$. While in the previously described formulation, we have specified the tracking costs in a stepwise fashion for the purpose of interpolation, we use here a *sparse* reference (Fig. 5, top). Intuitively, this corresponds to an optimisation objective that prioritises trajectory smoothness and only enforces tracking the given states at the given time steps. To achieve this objective,

[2]We adopt the term *key-points* rather than *via-points* here, because the latter term does not describe the initial and final point of the trajectory.

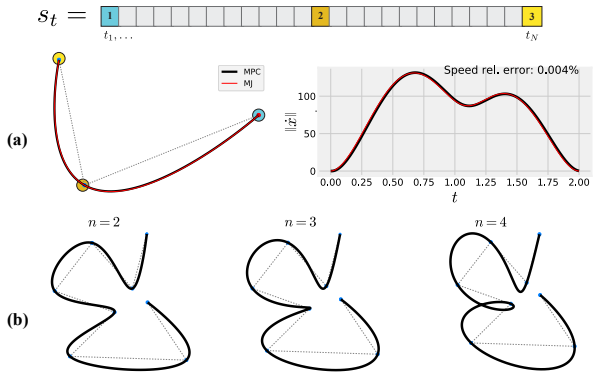

Figure 5: Interpolation with MPC. **a**, comparison of MPC (black) with a closed form solution of a minimum jerk trajectory with 1 via-point (red). On top, a color coded schematic of the corresponding state vector. **b**, examples of interpolating trajectories of increasing order $(2, 3, 4)$.

we set all desired states and weights to zero except for the time occurrence of each key-point with

$$\hat{\boldsymbol{\xi}}_t = \begin{cases} \boldsymbol{C}^\top \boldsymbol{v}_{t_i} & \text{if } t_i = k \\ \boldsymbol{0}, & \text{otherwise.} \end{cases}, \tag{6}$$

and

$$\boldsymbol{Q}_t = \begin{cases} \boldsymbol{C}^\top \boldsymbol{I}\boldsymbol{C}, & \text{if } t_i = k \\ \boldsymbol{I}, & \text{if } t = N \\ \boldsymbol{0}, & \text{otherwise.} \end{cases} \tag{7}$$

This expresses the penalty of deviating from key-point positions at a given time step k with an identity covariance, which reduces to a state cost given by the Euclidean norm to the key-point position. In order to minimize the command amplitude across the whole movement, we then set $\boldsymbol{R}_t = \lambda \boldsymbol{I}$, where λ is a very small regularisation term (1×10^{-15} for the examples given in this section). Note that the matrix $\boldsymbol{C}$ is again used to determine the number of derivatives that influence the cost of each via-point. The last entry of $\boldsymbol{Q}_t$ is always set to a full identity matrix, which again enforces a zero boundary condition for the whole state of the system. This formulation allows us to use MPC to produce close numerical approximations of a number of trajectory generation methods, such as polynomial interpolation/splines [14] as well as "minimum square derivative" (MSD) methods such as the minimum jerk model [38]. As a comparative example, we demonstrate that our method can closely approximate curved minimum jerk trajectories with 1 via-point (Fig. 5a), the closed form solution for which is defined by Flash and Hogan in the form of a quintic polynomial [16]. It should be noted that while the MSD methods predict a time difference between consecutive via-points that is *approximately* equal across a movement, the exact time occurrences of each via-point are predicted by the models as part of the optimisation. This will result in the via-point occurring in the proximity of a curvature extrema of the

trajectory. In the example shown in Fig. 5a, we compute the exact time occurrence as predicted by the minimum jerk model, which can be done by numerically finding the real roots of a polynomial of degree 9 [16]. For more complex trajectories, we currently assume a uniform spacing in time between via-points (Fig. 5b), leaving the optimization of this timing information for further work.

4.3 Mimicking Bézier curves

In the following paragraphs we describe how the same optimisation framework can be used to mimic the shape and behavior of (cubic) Bézier curves. The resulting trajectory/curve generation method provides means of interaction almost identical to its parametric counterpart. At the same time, it provides the flexibility of MPC (such as the ability to easily adjust the trajectory smoothness) and also guarantees trajectory smoothness regardless of the configuration of control points. This is particularly useful for calligraphy generation, where the desired trajectories are per se smooth.

It has been shown that cubic Bézier curves [14] and splines [13] can be interpreted as the trajectories of a 2nd order dynamical system which minimise acceleration commands. Indeed, we can see that with the previously described key-point formulation, it is possible to closely approximate a Bézier curve. This can be done by setting also the first order derivative entry of the sensor matrix $\boldsymbol{C}$ to $\boldsymbol{I}$, and then augmenting the key-points with the derivative computed according to the cubic Bézier formulation (Fig. 6a). This method allows to closely approximate a Bézier curve, and can be used with the same constraints and a higher order systems, which results in a higher degree of continuity (Fig. 6b). However, the method has limitations in the definition of piecewise curves, which for example require setting the same velocity where consecutive curve segments connect.

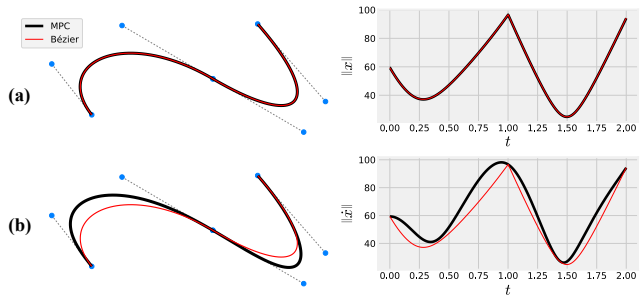

Figure 6: Approximating Bézier curves with a sparse reference and states augmented with velocity. **a**, 2nd order system, which results in an identical curve and speed profile (as shown in [14]). **b**, the same states with a 3rd order system. This produces a slightly different curve with a higher degree of continuity.

We observe that we can also mimic the behavior and shape of a Bézier curve by using a stepwise tracking reference. This can be done by placing isotropic covariance Gaussians centered at each control point of the curve, and then adjusting the influence of intermediate control points on the trajectory by uniformly increasing the variance of each corresponding Gaussian (Fig. 7). The variances are currently set empirically, but we plan to explore methods for automatising this process in further iterations of the study.

At the cost of a less precise approximation, we obtain a curve generation method that produces similar shapes to Bézier curves with a similar representation, and with the additional flexibility of the Gaussian representation and the benefit of always maintaining smooth and physiologically plausible kinematics. The utility of this property in our application is emphasized if we randomly perturb the control points of a letter form and compare the result with the one produced with a Bézier curve (Fig. 8).

While the Bézier curve becomes discontinuous due to the differently oriented tangents at the places where the curve segments

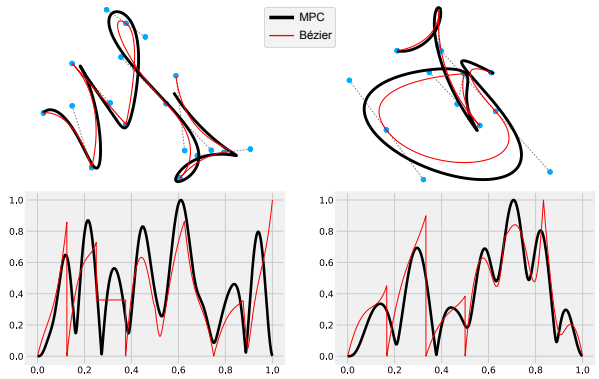

Figure 7: Mimicking Bézier curves (red) with MPC (black) using a stepwise reference, isotropic Gaussians and a 4th order system. Below, the corresponding speed profiles normalised and superposed for comparison.

Figure 8: Effect of randomly displacing control point positions with Bézier curves (red) and MPC (black).

meet, the MPC formulation tends to maintain a smooth trajectory regardless of the positions of the control points. This can be exploited as an additional method to generate synthetic variations of a handwriting or calligraphy trajectory, which can be interactively edited with a traditional control point and tangent interface. The same smoothness property can be used to concatenate multiple letter forms with ligatures that evoke a smooth and natural motion, which can be easily achieved by treating the control points of the letters as a single trajectory (Fig. 9).

Figure 9: Automatic ligature generation by concatenating the control points of two letters. On the right, a comparative example using Bézier curves.

5 Interactive Trajectory Specification and Rendering

The MPC based methods described above are well suited for the interactive definition of trajectories. It is in fact trivial to drag the key-points (Sect. 4.2) or control points (Sect. 4.3) with a typical point-and-click procedure, and it is also simple to interactively manipulate the Gaussians (Sect. 4.1) for the probabilistic case (Fig. 1). Each Gaussian can be edited interactively by manipulating an ellipsoid, where the centre of the ellipsoid defines the mean $\boldsymbol{\mu}_i$, and the axes are used to manipulate the covariance $\boldsymbol{\Sigma}_i$ through its eigendecomposition. The latter can be described with

$$\boldsymbol{\Sigma}_i = \boldsymbol{\Theta}_i \boldsymbol{S}_i^2 \boldsymbol{\Theta}_i^\top, \tag{8}$$

where $\boldsymbol{\Theta}_i$ corresponds to an orthogonal (rotation) matrix, and $\boldsymbol{S}_i$ is a scaling matrix. Here, we describe the 2D case in which the rotation

Figure 10: Top left, a graffiti script (tag) made with a marker by Los Angeles artist "Trixter". Top right, a user defined "motor plan" for the tag, generated by placing targets near salient positions along the original trace and then adjusting the covariances to follow the original trajectory. Second row, the reconstructed trajectory (left) and one variation made by increasing the regularisation parameter r. Bottom row, two random samples from the corresponding trajectory distribution.

and scaling matrices are given by

$$\boldsymbol{\Theta}_i = \begin{bmatrix} \cos\theta & -\sin\theta \\ \sin\theta & \cos\theta \end{bmatrix}, \quad \theta = \tan^{-1}\frac{a_2}{a_1}, \quad \boldsymbol{S}_i = \begin{bmatrix} \frac{\|\boldsymbol{a}\|}{2} & 0 \\ 0 & \frac{\|\boldsymbol{b}\|}{2} \end{bmatrix}, \tag{9}$$

where $\boldsymbol{a}$ and $\boldsymbol{b}$ are the major and minor axes of an ellipse, which can be interactively dragged to manipulate the shape of the distribution. Isotropic Gaussians influence the trajectory evolution, in a similar manner to a virtual target or control point, where a small variance will force the trajectory to pass close to the centre. Thinner Gaussians influence the curvilinear evolution of the trajectory, forcing it to follow the direction of the major axis of the ellipse. While the 3D case is currently not implemented, it would be straightforward to adapt the described technique to an arc-ball interface [37], in order to manipulate the 3D rotation components.

In most of our examples we settle with a dynamical of order 4, which we evaluate to give the best balance between trajectory smoothness and a precise control on the trajectory evolution. In order to achieve approximately equal tracking performance across different system orders, we express the control cost in terms of a *maximum displacement* d, and compute R based on the low frequency gain of the system with

$$\boldsymbol{R} = \frac{1}{(\omega^n d)^2}\boldsymbol{I} \qquad \text{and} \qquad \omega = 2\pi\frac{T}{m-1}, \tag{10}$$

with ω empirically chosen, corresponding to the average duration of a stroke. Higher values of d will produce sharper trajectories, while lower values will result in smoother trajectories. On the other hand, because the cost function is defined as a tradeoff between accurate tracking and smooth control, it is possible to keep the value of d fixed depending on the size of the working area, and then interact with the trajectory by only manipulating the Gaussians.

By utilising the stochastic sampling technique described in Sect. 4.1.1, we can interactively visualise the variability of the generated trajectories, while manipulating the Gaussians. This results in a

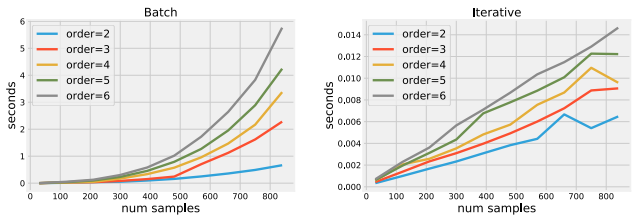

Figure 11: Comparison of performance for the batch and iterative approach.

novel form of interactive trajectory editing that, due to its generative nature, directly describes a family of trajectories rather than a single one. As an example, the same interface can be used to rapidly reconstruct, and generate variations of, an existing instance of human made calligraphy (Fig. 10). In this use case, the user first defines a coarse sequence of Gaussians over salient positions along the input trace (approximately in correspondence with curvature extrema), and then adjusts the covariances to modify the trajectory and mimic the curvature and smoothness of the original trace. Variations of the input can then be generated by stochastic sampling.

The proposed methods generate trajectories with smooth derivatives up to the order of the system used in the optimisation. This can be exploited to facilitate painterly and expressive renderings of the trajectory. In this study, we limit ourselves to a simple brush model (see Fig. 1 and 10), which assumes that the amount of paint deposited is inversely proportional to the speed of the pen. To mimic this effect, we sweep a pre-selected texture image along the trajectory with a size that varies as an inverse power function of the instantaneous speed. While this is obviously not an accurate model of a brush or pen, it results in a trajectory rendering that accentuates the perceived dynamics of the trace (see also accompanying video).

Furthermore, the distances between consecutive points along the generated trajectory reflect the smooth kinematics generated by the model. As a result, it is possible to easily generate realistic and natural looking animations, by incrementally sampling points along the trajectory with a fixed time-step. This approach can be used to either generate stroke animations, or to guide the hand motion of a virtual character or robotic arm.

5.1 Performance

We have tested our method on a 2,5 GHz Intel Core i7 machine and used OpenGL for hardware accelerated rendering; We have implemented the optimisation code in Python, using the NumPy [43] linear algebra package, as well as in C, using the Armadillo library [35]. Both the batch and the iterative approaches run at interactive rates up to a limit of time steps that depends on the order of the system used in the optimisation. For the examples given in this paper we use an optimisation horizon of approximately 200 time steps, which results in trajectories that are perceived as smooth, and for which the batch and iterative solutions take in average (across orders) 70 ms and 3 ms respectively. The batch approach requires the inversion of a matrix with a dimension which is directly proportional to the number of time steps, and obviously this results in a rapid performance drop as the latter increases (Fig. 11, left). This problem is overcome with the iterative solution, the complexity of which is approximately linear to the number of time steps (Fig. 11, right). On the other hand, the batch solution is more compact and allows the intuitive formulation of probabilistic interpretation of the output and stochastic sampling of the trajectory distribution.

6 Conclusion

We have proposed the use of model predictive control (MPC) as a curve generation tool, which can be used in a manner similar to

conventional interfaces for polynomial curve generation. Our main contribution is an application to computer graphics of a probabilistic formulation of MPC that explicitly describes intra-movement variability and coordination. We have shown that the same framework can also be used to compute numerical approximations for polynomial interpolation methods such as Bézier curves, or well known trajectory formation methods such as the minimum jerk model. This results in a flexible and general trajectory generation method, that we consider particularly useful for applications that necessitate the simulation of traces such as the ones made by a human when drawing or writing.

While in this paper we focused on the generation of 2D trajectories, the proposed methodology can be generalised to higher dimensions. This opens up the possibility to extend the method to 3D trajectories, as well as taking in consideration the evolution of additional variables, such as the drawing tool orientation or pressure.

In this paper, we have focused on the application of MPC for the generation of movements and traces that mimic the visual qualities of human made graffiti and calligraphy. At this stage, we have relied on the qualitative evaluation by a number of experienced artists and designers ($n = 5$), who have characterised the output of our system as valid instantiations of the targeted hand-styles. In future work, we intend to perform a series of controlled user studies in order to further evaluate the quality of the generated traces and motions. It should be noted that evaluating the aesthetic quality of a visual mark or trace is not a well defined problem, and the notion of "style similarity" can be subjective to the viewer and depends on factors such as cultural and artistic background. On the other hand, we propose, in line with others before us such as Hertzmann [21] or Stacey [39], that the computational study of style is worthwhile, and that a procedure that generates patterns that are perceived as similar to a given artistic style, provides the grounds to establish a potential and computational theory of art that is both *generative* and descriptive [21].

7 ACKNOWLEDGMENTS

We would like to thank Trixter and David Chang for kindly providing calligraphy samples. This work has been partly supported by UK's EPSRC Centre for Doctoral Training in Intelligent Games and Game Intelligence (IGGI; grant EP/L015846/1).

A DYNAMICAL SYSTEM

The continuous-time system matrices for the chain of integrators are given by

$$\bar{\boldsymbol{A}} = \begin{bmatrix} \mathbf{0} & \boldsymbol{I} & \mathbf{0} & \cdots & \mathbf{0} \\ \mathbf{0} & \mathbf{0} & \boldsymbol{I} & \cdots & \mathbf{0} \\ \vdots & \vdots & \vdots & \ddots & \vdots \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \boldsymbol{I} \\ \mathbf{0} & \mathbf{0} & \mathbf{0} & \cdots & \mathbf{0} \end{bmatrix}, \; \bar{\boldsymbol{B}} = \begin{bmatrix} \mathbf{0} \\ \mathbf{0} \\ \vdots \\ \mathbf{0} \\ \boldsymbol{I} \end{bmatrix}. \tag{11}$$

The discrete time state matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ used in Equation 1 are then computed using a Zero Order Hold (ZOH) or forward Euler discretisation of $\bar{\boldsymbol{A}}$ and $\bar{\boldsymbol{A}}$ with a sampling period Δt, where for the Euler case we simply have

$$\boldsymbol{A} = \Delta t \bar{\boldsymbol{A}} + \boldsymbol{I} \quad \text{and} \quad \boldsymbol{B} = \Delta t \bar{\boldsymbol{B}}. \tag{12}$$

B COST FUNCTION SOLUTIONS

B.1 Batch solution

For the batch solution, we express the cost function with

$$J = \left(\hat{\boldsymbol{\xi}} - \boldsymbol{\xi}\right)^\top \boldsymbol{Q} \left(\hat{\boldsymbol{\xi}} - \boldsymbol{\xi}\right) + \boldsymbol{u}^\top \boldsymbol{R} \boldsymbol{u}, \tag{13}$$

where $\boldsymbol{Q} = \text{blockdiag}\left(\boldsymbol{Q}_1, \boldsymbol{Q}_2, ..., \boldsymbol{Q}_N\right) \in \mathbb{R}^{nDN \times nDN}$, $\boldsymbol{R} = \text{blockdiag}\left(\boldsymbol{R}_1, \boldsymbol{R}_2, ..., \boldsymbol{R}_{N-1}\right) \in \mathbb{R}^{DN \times DN}$, and the desired state, current state and control commands are respectively stacked in in large column vectors $\hat{\boldsymbol{\xi}}$, $\boldsymbol{\xi}$ and $\boldsymbol{u}$. We then express all future states as a function of the initial state $\boldsymbol{\xi}_1$, which can be compactly represented in matrix form as

$$\boldsymbol{\xi} = \boldsymbol{S}_{\boldsymbol{\xi}} \boldsymbol{\xi}_1 + \boldsymbol{S}_{\boldsymbol{u}} \boldsymbol{u}, \tag{14}$$

with

$$\boldsymbol{S}_{\boldsymbol{\xi}} = \begin{bmatrix} \boldsymbol{I} \\ \boldsymbol{A} \\ \boldsymbol{A}^2 \\ \vdots \\ \boldsymbol{A}^N \end{bmatrix} \quad \text{and} \quad \boldsymbol{S}_{\boldsymbol{u}} = \begin{bmatrix} \mathbf{0} & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{B} & \mathbf{0} & \dots & \mathbf{0} \\ \boldsymbol{A}\boldsymbol{B} & \boldsymbol{B} & \dots & \mathbf{0} \\ \vdots & \vdots & \ddots & \vdots \\ \boldsymbol{A}^{N-1}\boldsymbol{B} & \boldsymbol{A}^{N-2}\boldsymbol{B} & \dots & \boldsymbol{B} \end{bmatrix}, \tag{15}$$

where $\boldsymbol{S}_{\boldsymbol{\xi}} \in \mathbb{R}^{nND \times D}$ and $\boldsymbol{S}_{\boldsymbol{u}} \in \mathbb{R}^{nND \times (N-1)D}$. Substituting Equation 14 into Equation 13, differentiating with respect to $\boldsymbol{u}$ and setting to zero results in a regularized least squares estimate of the optimal command sequence, given by

$$\boldsymbol{u} = \left(\boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^\top \boldsymbol{Q} \left(\hat{\boldsymbol{\xi}} - \boldsymbol{S}_{\boldsymbol{\xi}} \boldsymbol{\xi}_1\right), \tag{16}$$

where the control weight matrix $\boldsymbol{R}$ effectively acts as a Tikhonov regularization term, see e.g. [6]. The command sequence $\boldsymbol{u}$ is then substituted back into (14), resulting in the optimal trajectory $\boldsymbol{\xi}$.

We note that with increasing system orders the amplitude of the commands will grow exponentially, which can result in a badly scaled matrix in the inverse term of Equation 16. To overcome this problem, we first scale the desired states $\hat{\boldsymbol{\xi}}$ by a factor, and the stacked weight matrices $\boldsymbol{Q}$ by the same factor squared. This value is chosen in order to limit the standard deviation of the components of $\hat{\boldsymbol{\xi}}$ below a maximum range. In the examples given in this paper we choose a factor that keeps the standard deviation below 1×10^{-6}, which gives numerically stable results up to a system order of 5.

B.2 Iterative solution

A more efficient solution to the optimisation problem can be derived using dynamic programming or an extension of variational calculus known as Pontryagin's Maximum Principle. We refer the interested reader to the work of Bryson [7] for the details of the derivations. It follows that the optimal solution is given in the form of a feedback controller with time varying weighting matrix $\boldsymbol{K}_t$, and the commands are for each time step t given by

$$\boldsymbol{u}_t = \underbrace{\left(\boldsymbol{B}^\top \boldsymbol{P}_t \boldsymbol{B} + \boldsymbol{R}_t\right)^{-1} \boldsymbol{B}^\top \boldsymbol{P} \boldsymbol{A}}_{\boldsymbol{K}_t} \tilde{\boldsymbol{\xi}}_t, \tag{17}$$

where

$$\boldsymbol{P}_t = \tilde{\boldsymbol{Q}}_t - \boldsymbol{A}^\top \left(\boldsymbol{P}_{t+1} \boldsymbol{B} \left(\boldsymbol{B}^\top \boldsymbol{P}_{t+1} \boldsymbol{B} + \boldsymbol{R}_t\right)^{-1} \boldsymbol{B}^\top \boldsymbol{P}_{t+1} - \boldsymbol{P}_{t+1}\right) \boldsymbol{A} \tag{18}$$

is a *Riccati difference equation*, which can be solved backwards in time by setting a terminal condition $\boldsymbol{P}_N = \boldsymbol{Q}_N$. In Equation 17 and Equation 18, we introduce the symbols $\tilde{\boldsymbol{\xi}}_t$ and $\tilde{\boldsymbol{Q}}_t$. These respectively denote an *augmented* state vector

$$\tilde{\boldsymbol{\xi}}_t = \left[\boldsymbol{\xi}_t^\top, 1\right]^\top, \tag{19}$$

and an augmented tracking weight

$$\tilde{\boldsymbol{Q}}_t = \begin{bmatrix} \boldsymbol{Q}_t^{-1} + \hat{\boldsymbol{\xi}}_t \hat{\boldsymbol{\xi}}_t^\top & \hat{\boldsymbol{\xi}}_t \\ \hat{\boldsymbol{\xi}}_t^\top & 1 \end{bmatrix}^{-1}, \tag{20}$$

which permit us to treat the tracking problem more compactly and efficiently as a *regulation* problem, resulting in a formulation that is equivalent to a Linear Quadratic Regulator (LQR).

C STOCHASTIC SAMPLING

The optimal trajectory resulting from Equation 14 in the batch solution can be interpreted probabilistically as a trajectory distribution

$$\boldsymbol{\xi} \sim \mathcal{N}(\boldsymbol{\mu}_{\boldsymbol{\xi}}, \boldsymbol{\Sigma}_{\boldsymbol{\xi}}) \tag{21}$$

with

$$\boldsymbol{\mu}_{\boldsymbol{\xi}} = \boldsymbol{S}_{\boldsymbol{\xi}}\boldsymbol{\xi}_1 + \boldsymbol{S}_{\boldsymbol{u}}\boldsymbol{u} \quad \text{and} \quad \boldsymbol{\Sigma}_{\boldsymbol{\xi}} = \sigma \boldsymbol{S}_{\boldsymbol{u}} \left(\boldsymbol{S}_{\boldsymbol{u}}^{\top}\boldsymbol{Q}\boldsymbol{S}_{\boldsymbol{u}} + \boldsymbol{R}\right)^{-1} \boldsymbol{S}_{\boldsymbol{u}}^{\top}, \tag{22}$$

where σ is a scaling factor proportional to the mean squared error of the least square estimate in Equation 16, which can be computed automatically in Matlab with the *lscov* command. Additional details on the derivations are given in [8].

This permits the generation of an infinite number of trajectories through stochastic sampling of the distribution, which can be done with the eigendecomposition $\boldsymbol{\Sigma}_{\boldsymbol{\xi}} = \boldsymbol{V}_{\boldsymbol{\xi}}\boldsymbol{\Lambda}_{\boldsymbol{\xi}}\boldsymbol{V}_{\boldsymbol{\xi}}^{\top}$, where $\boldsymbol{V}_{\boldsymbol{\xi}}$ is a matrix of eigenvectors of the symmetric matrix $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}$, and $\boldsymbol{\Lambda}_{\boldsymbol{\xi}}$ is a matrix with the respective eigenvalues along the diagonal. We can then stochastically generate variations around the mean trajectory with

$$\boldsymbol{\xi} \sim \boldsymbol{\mu}_{\boldsymbol{\xi}} + \boldsymbol{V}_{\boldsymbol{\xi}}\boldsymbol{\Lambda}_{\boldsymbol{\xi}}^{\frac{1}{2}}\mathcal{N}(\boldsymbol{0}, \boldsymbol{I}). \tag{23}$$

In practice, the covariance matrix $\boldsymbol{\Sigma}_{\boldsymbol{\xi}}$ will be of high dimension, which will result in slow computation if all eigencomponents are evaluated. It is sufficient here to utilise a reduced subset of eigencomponents with the largest eigenvalues (between 5 and 9 in the provided examples). This can be done at an interactive rate by using the Arnoldi iteration technique [2], which is readily implemented in commonly used linear algebra packages (Matlab®, NumPy in Python, and ARPACK in C).

REFERENCES

[1] Z. AlMeraj, B. Wyvill, T. Isenberg, A. A. Gooch, and R. Guy. Automatically mimicking unique hand-drawn pencil lines. *Computers & Graphics*, 33(4):496–508, 2009.

[2] W. E. Arnoldi. The principle of min. iterations in the soln. of the matrix eigenvalue problem. *Quat. of App. Maths*, 9(1):17–29, 1951.

[3] D. Berio and F. F. Leymarie. Computational Models for the Analysis and Synthesis of Graffiti Tag Strokes. In P. Rosin, ed., *Computational Aesthetics (CAe)*, pp. 35–47. Eurographics Association, June 2015.

[4] N. A. Bernstein, M. L. Latash, and M. Turvey. *Dexterity and its development*. Taylor & Francis, 1996.

[5] H. Bezine, A. M. Alimi, and N. Sherkat. Generation and analysis of handwriting script with the beta-elliptic model. *Int'l Workshop on Frontiers in Handwriting Recognition (IWFHR)*, 8(2):515–20, 2004.

[6] C. M. Bishop. *Pattern Recognition and Machine Learning (Information Science and Statistics)*. Springer, 2006.

[7] A. E. Bryson. *Dynamic optimization*. Addison Wesley, 1999.

[8] S. Calinon. Stochastic learning and control in multiple coordinate systems. In *Intl Workshop on Human-Friendly Robotics*, 2016.

[9] S. Calinon. A tutorial on task-parameterized movement learning and retrieval. *Intelligent Service Robotics*, 9(1):1–29, 2016.

[10] M. J. T. Da Silva. *Pre-computation for controlling character behavior in interactive physical simulations*. PhD thesis, Citeseer, 2010.

[11] J. B. Dingwell, C. D. Mah, and F. A. Mussa-Ivaldi. Experimentally confirmed mathematical model for human control of a non-rigid object. *Journal of Neurophysiology*, 91(3):1158–1170, 2004.

[12] S. Edelman and T. Flash. A model of handwriting. *Biological cybernetics*, 57(1-2):25–36, 1987.

[13] M. Egerstedt and C. Martin. *Control Theoretic Splines: Optimal Control, Statistics, and Path Planning*. Princeton Univ. Press, 2009.

[14] M. B. Egerstedt, C. F. Martin, et al. A note on the connection between Bezier curves and linear optimal control. *IEEE Transactions on Automatic Control*, 49(10):1728–31, 2004.

[15] S. E. Engelbrecht. Minimum principles in motor control. *Journal of Mathematical Psychology*, 45(3):497–542, 2001.

[16] T. Flash and N. Hogan. The coordination of arm movements. *Journal of Neuroscience*, 5(7):1688–1703, 1985.

[17] W. C. Fong. Why Chinese painting is history. *The Art Bulletin*, 85(2):258–80, 2003.

[18] D. Freedberg and V. Gallese. Motion, emotion and empathy in esthetic experience. *Trends in cognitive sciences*, 11(5):197–203, 2007.

[19] J. J. Freyd. Representing the dynamics of a static form. *Memory & cognition*, 11(4):342–346, 1983.

[20] P. Haeberli. Dynadraw: A dynamic drawing technique, 1989. http://www.graficaobscura.com/dyna/.

[21] A. Hertzmann. Non-photorealistic rendering and the science of art. In *8th Int'l Symp. on Non-Photorealistic Animation and Rendering (NPAR)*, pp. 147–57. ACM, 2010.

[22] D. H. House and M. Singh. Line drawing as a dynamic process. In *Computer Graphics and Applications, 2007. PG'07. 15th Pacific Conference on*, pp. 351–360. IEEE, 2007.

[23] M. I. Jordan and D. M. Wolpert. Computational motor control. In M. Gazzaniga, ed., *The Cognitive Neurosciences*. MIT Press, 2nd ed., 1999.

[24] J. E. Kyprianidis, J. Collomosse, T. Wang, and T. Isenberg. State of the "art": A taxonomy of artistic stylization techniques for images and video. *IEEE Trans. on Vis. & C.G.*, 19(5):866–85, 2013.

[25] J. Liu, H. Fu, and C.-L. Tai. Dynamic sketching: Simulating the process of observational drawing. In *Proceedings of the Workshop on Computational Aesthetics*, pp. 15–22. ACM, 2014.

[26] M. Longcamp, J. L. Anton, M. Roth, and J. L. Velay. Visual Presentation of Single Letters Activates a Premotor Area Involved in Writing. *NeuroImage*, 19(4):1492–1500, 2003.

[27] J. Lu, F. Yu, A. Finkelstein, and S. DiVerdi. Helpinghand: Example-based stroke stylization. *ACM Trans. on Graphics*, 31(4):46, 2012.

[28] J. McCrae and K. Singh. Sketching piecewise clothoid curves. *Computers and Graphics (Pergamon)*, 33(4):452–461, 2009.

[29] C. Mediavilla. *Calligraphy: From Calligraphy to Abstract Painting*. Scirpus, 1996.

[30] P. Morasso. Spatial control of arm movements. *Experimental Brain Research*, 42(2):223–7, 1981.

[31] P. Morasso and F. Mussa Ivaldi. Trajectory formation and handwriting: a computational model. *Biological cybernetics*, 45(2):131–142, 1982.

[32] A. Pignocchi. How the Intentions of the Draftsman Shape Perception of a Drawing. *Consciousness and Cognition*, 19(4):887–898, 2010.

[33] R. Plamondon et al. Recent developments in the study of rapid human movements with the kinematic theory. *Pattern Recognition Letters*, 35:225–35, 2014.

[34] D. Rosand. *Drawing acts: Studies in graphic expression and representation*. Cambridge University Press, 2002.

[35] C. Sanderson and R. Curtin. Armadillo: A template-based C++ library for linear algebra. *Journal of Open Source Software*, 1(2), 2016.

[36] H. Shinoda et al. Generation of cursive characters using minimum jerk model. In *IEEE Proc. SICE*, vol. 1, pp. 730–3, 2003.

[37] K. Shoemake. ARCBALL: A user interface for specifying three-dimensional orientation using a mouse. In *Graphics Interface*, pp. 151–6, 1992.

[38] G. Simmons and Y. Demiris. Optimal robot arm control using the minimum variance model. *Journal of Robotic Systems*, 22(11):677–690, 2005.

[39] M. Stacey. Psychological challenges for the analysis of style. *Artificial Intelligence for Engineering Design, Analysis and Manufacturing*, 20(3):167–84, 2006.

[40] H.-L. Teulings and L. Schomaker. Invariant properties between stroke features in handwriting. *Acta psychologica*, 82(1):69–88, 1993.

[41] Y. Thiel, K. Singh, and R. Balakrishnan. Elasticurves: Exploiting stroke dynamics and inertia for the real-time neatening of sketched 2D curves. In *Proc. 24th ACM Symp. on User Interface Software & Technology (UIST)*, pp. 383–92, 2011.

[42] E. Todorov and M. I. Jordan. Optimal feedback control as a theory of motor coordination. *Nature neuroscience*, 5(11):1226–1235, 2002.

[43] S. Van Der Walt, S. C. Colbert, and G. Varoquaux. The NumPy array: A structure for efficient numerical computation. *Computing in Science & Engineering*, 13(2):22–30, 2011.

[44] P. Viviani and G. Mccollum. The relation between linear extent and velocity in drawing movements. *Neuroscience*, 10(1):211–8, 1983.