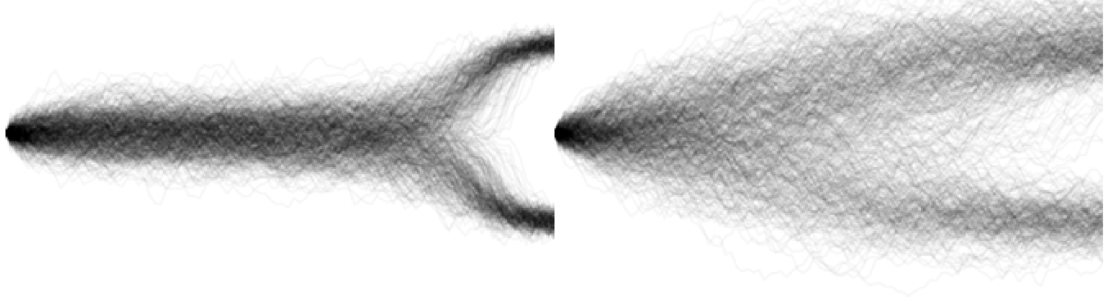
Path Integral Control with history dependent basis functions

Anton de Groot

August 4, 2020



Contents

1. Introduction

In general the optimal control is time dependent for a finite horizon problem. A finite horizon problem is an optimization problem where you want to find an optimal solution within a specific time frame. In this thesis I will look if the time dependent control can be approximated by using a time independent feature based controller using delayed inputs.

To find the optimal control Path Integral Control theory (PIC) will be used. PIC formulates an optimal control in terms of a cost associated with a path taken and an end-cost, the successfulness of a taken path. In our daily lives, we would like to avoid obstacles that are in our paths, but we also do not want to swerve excessively. PIC theory will provide an algorithmic description to accomplish this.

In this thesis we can imagine a drone trying to fly through a gap in a wall. The algorithm in appendix B will compute a control function, such that the drone will fly through one of two gaps with a high success rate.

The model with basis functions has inputs that are only dependent on its current position to determine what the control function should be. This means that it does not explicitly depend on time. We will call this model the Stationary Delayed Control or SDC.

2. Theory

2.1. Basis functions

If you want to approximate a certain function by the superposition of other functions, then given some inputs x and weights ω we can write it in the form: [1]

$$u(\vec{x}, \vec{\omega}) = \sum_{i=1}^{D} \omega_i h_i(x) \tag{1}$$

where h_i is a so called basis function and $u(x)$ will be our control function. There is a plethora of basis functions; in Fourier series a combination of sines and cosines can approximate certain functions. So in this case the basis functions could be sines, but other options also include polynomials such that $h_i(x) = x^i$. In this thesis I will look at the hyperbolic tangent function of the form:

$$h_i(x(t), x(t-d\tau), \ldots, x(t-n\cdot d\tau)) = \tanh\left(\sum_{k=0}^{n} x(t-k\cdot d\tau) b_{ik} + c_i\right) \tag{2}$$

This $d\tau$ is not necessarily the same as dt from the dynamics, it can be chosen to be much larger than dt. This basis function is chosen because we would like to have a basis function that does not result in a control term that can grow infinitely large, as it is not physically possible to exert such a force. This can be the case for polynomial basis functions. Another reason to choose hyperbolic tangents as basis functions is because there is only one fixed point per basis function for hyperbolic tangents, but infinitely many stable and unstable fixed points if we choose sinusoidal basis functions.

2.2. Delay lines and history dependent basis functions

Delay lines repeat what the past D positions were. For example the current position $x(t_i)$ for N different paths will be a vector of length N, a generalized position X will then be a matrix $\mathbb{R}^{N\times D}$

$$X(t_i) = \begin{pmatrix} \vec{x}(t_i) & \vec{x}(t_{i-1}) & \ldots & \vec{x}(t_{i-D}) \end{pmatrix} \tag{3}$$

The matrix $X(t_i)$ is then used in basis functions of the form $\tanh(Xb_j + c_j)$.

2.3. The algorithm

In order to compute the trajectory of a certain path in discrete time, we can express the displacement dX by the following dynamics:

$$dX = udt + dW \tag{4}$$

If not a single path ends up through a gap, that is to say, not a single target is reached, then the algorithm in section B cannot be applied and is also not able to learn.[1] This is the only constraint on the magnitude of the drift term. U is the aforementioned control term. This can be any arbitrary Markov control [2], i.e. a stochastic control in discrete time. And dW is a stochastic variable. This can be rewritten in the form:

$$\dot{X} = u + \xi \tag{5}$$

where ξ equals:

$$\xi = \frac{dW}{dt} \tag{6}$$

To be able to use Path Integral Control theory we must use a certain path cost, defined as:

$$S^u(t) = \int_{t_i}^{t_f} V(\tau, X^u) + \frac{1}{2} u(\tau, X^u)^2 d\tau + \int_{t_i}^{t_f} u(\tau, X^u) dW \tag{7}$$

Note that the cost S depends on future values of X and is therefore not adaptive. [2] It also includes a stochastic integral with respect to Brownian motion. This is somewhat unusual because the stochastic integral vanishes when taking the expected value, as is done in equation (11). However, such a term appears naturally in the theory when you write the most general form of the equations. [2]

The function $V(\tau, X(\tau))$ is a cost-function associated with a certain position. This cost-function will be infinite when it coincides with the wall at time t_f and zero otherwise, this will also be zero when $t \neq t_f$. This means that a path is only punished by this term by the result it yields: success or failure. This cost term is called the end-cost $\Phi(X(t_f))$.

$$S^u(t) = \Phi(X^u(t_f)) + \int_{t_i}^{t_f} \frac{1}{2} u(\tau, X^u)^2 d\tau + \int_{t_i}^{t_f} u(\tau, X^u) dW \tag{8}$$

The second term in the above equation: $\int u^2 d\tau$, means that any large control functions will be punished. The last term in the equation indicates that the control udt is disturbed by the noise dW. If u and dW point in opposite directions then the noise counteracts the control, which is unlucky. If they point in the same direction then it enhances the control, which can be considered as good luck. Whether a path is lucky or not is essentially accounted for in the cost of the path because $\int udW$ is negative and reduces the cost if

[1]Since all end-costs will be infinite and there will not even be a measure of 'better performance'.

u and dW are in opposite directions and the integral is positive and increases the cost if they both point in the same direction. This makes sure that for the optimal control the cost is always the same regardless of luck. [2]

It can be shown that if you assume the optimal controller $u^\star$ to be of the parametrized form: [3]

$$u^\star(x,t) = A^\star h(x,t) \tag{9}$$

Where h is the k-dimensional basis functions applied to x and the parameters $A \in \mathbb{R}^{N\times k}$. Solving this equation will then lead to:

$$\sum_{k'=1}^{K} \left(\hat{\theta}_{k'} - \theta_{k'}\right) \langle h_{k'}(X_s) h_k(X_s)\rangle_u = \langle dW h_k(Xs)\rangle_u \tag{10}$$

this is solved by:

$$A = \left(\langle uh'\rangle + \langle h(dW+u)\rangle\right) \langle hh'\rangle^{-1} \tag{11}$$

Where h is computed using delay lines using equation 2 and its implementation can be found in figure 5.

$$u(x,t) = A\cdot h(x,t) \tag{12}$$

This control will not be the optimal control, but after applying the algorithm iteratively, it will approach the optimal control. We now have almost everything we need to start applying the algorithm. We also need a measure of performance, for this we first need the weight of a path: [2]

$$\alpha^u = \frac{e^{-S^u_{t_0}}}{\mathbb{E}\left[e^{-S^u_{t_0}}\right]} \tag{13}$$

Where we can introduce a function called ψ

$$\psi(x,t) = e^{-S^u_t} \tag{14}$$

Where the cost S is the cost of paths originating from (x,t) ending at time T. The fraction of effective samples is defined as

$$\lambda^u = \frac{1}{\mathbb{E}\left[(\alpha^u)^2\right]} = \frac{1}{1+var\left((\alpha^u)^2\right)} \qquad 0 \le \lambda^u \le 1 \tag{15}$$

In the algorithm this will also be called the effective sample size (EffSS) but there it varies $0 \le$ EffSS $\le N$, i.e. the number of paths. If we redefine the path weight $w = e^{-S}$ as the entire path cost $t \in [0,T]$, we can write the effective sample size as:

$$\text{Effss} = N\cdot\lambda = \frac{N}{\mathbb{E}[(\alpha)^2]} = \frac{(\sum w)^2}{\sum w^2} \qquad 0 \le \text{EffSS} \le N \tag{16}$$

using the fact that $\mathbb{E}\left[w^x\right] = \frac{\sum w^x}{N}$. We will also be looking at the success rate of the algorithms, this is the fraction of paths that end up in the desired end-state. So if we can compute A we can also compute the control $u(x,t)$ after which we can compute a new A.

3. Cost-to-go and the optimal control

The path integral control has an exact solution in the form of the following equation:

$$\psi(x,t) = \int dzq(z,T|x,t)\exp\left(-\phi(z)/\lambda\right) \tag{17}$$

The solution to which can be found in appendix A.2 in equations 38 and 39. The only thing that remains is to determine $\sigma(t)$ the variance of the distribution at time t. Realize that the noise is picked from a normal distribution with mean 0 and a standard deviation of 1. This means that for a single discrete time step $\sigma^2 = 1$. We will be looking at a distribution of paths originating from point (x,t) at the final time T. This will entail $\frac{T-t}{dt}$ discrete time steps. Furthermore, we will demand that at time $t = T - dt$ this will then have a standard deviation of 1 since there is only 1 step left to take.

For a time t such that $t \in (0,T)$ we can then write the standard deviation as:

$$\sigma^2(t) = v \cdot \frac{T-(t-dt)}{dt} \tag{18}$$

for a given constant v. If we plug in the condition that at time $t = T - dt$ σ^2 must equal 1 we then get:

$$\sigma^2(t) = \frac{T-(t-dt)}{2dt} \tag{19}$$

To compute the optimal control we will introduce the so called cost-to-go. This is a cost function that indicates how expensive the control will be to reach a desired end position from a start position (x,t). This can be expressed as

$$J(x,t) = -\log(\psi(x,t)) \tag{20}$$

This is the case because $\psi(x,t) \propto e^{-\text{pathcost}}$. As was introduced in equation 14. The optimal control $u^*(x,t)$ is then the $u(x,t)$ that minimizes the cost function at $J(x+f(x,u),t+dt)$ or to put it differently:

$$u_*(x,t) = \underset{u}{\operatorname{argmin}}\left(J(x+f(x,u),t+dt)\right) = \underset{u}{\operatorname{argmin}}\left(J(x+u\cdot dt,t+dt)\right) \tag{21}$$

where $f(x,u)$ is the dynamics of the system. We can now compute the Cost-to-go for any (x,t)

4. Results

For the SDC we start with a completely uncontrolled scenario. Only the noise generated by zero mean random variables will determine $x(t)$ of the paths. Then for each moment t a matrix A will be computed using X,u and the noise.[2] Then the paths will be redrawn, but at each t the control $u(x,t)$ will be computed using equation 12.

For the optimal control we only need to determine the paths once. We just start at time $t = 0$, noise does influence the paths, but now for each time step a control u is calculated for each path using equation 21. The goal of both approaches is to arrive at a hole in the wall positioned at $[1,2]$ and $[-1,-2]$.

If we calculate the cost-to-go, we can see in figure 1 the contour plot and a few intersections of that contour map at specific times. We can clearly see a bifurcation happening at the end. At first, the future is still uncertain, but after a certain delay a choice has been made and it will steer towards one of the endpoints.

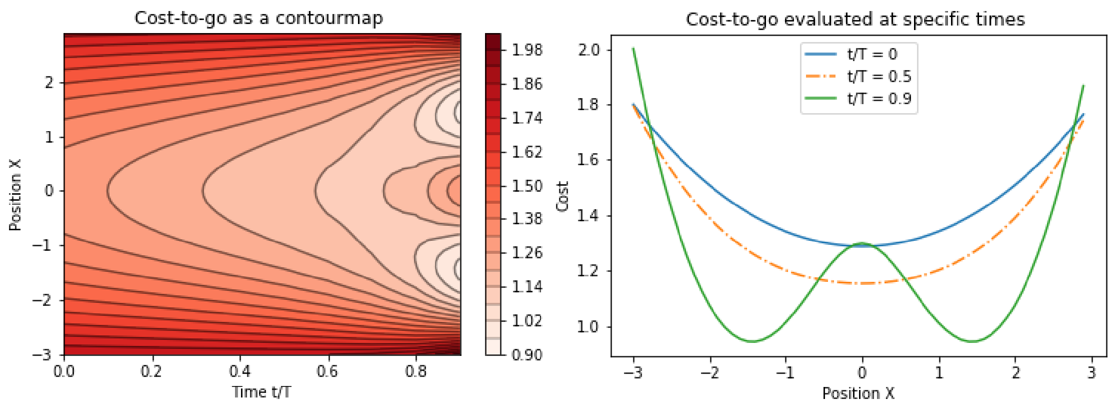


Figure 1: The cost-to-go plotted as a function of x and t

In figure 2 we can see that under an optimal control it first tries to steer towards the center but near the end it tries to steer toward one of the desired endpoints. We can also see that in the optimal control case the paths end tightly around 1.5 whereas for the SDC they end between [1,2]. The SDC's case looks more like a gaussian distribution albeit more guided towards the target. It is evident that the control is much stronger for the optimal controller than in the SDC's case.

The effective sample size in figure 3 is normalized because the two methods had different sample sizes, which in this case means simply that they had a different number of paths.[3] The reason for this is simply because computing the optimal control is more costly than computing the controller by using SDC. The effective sample size of the optimal control is 99.86%. It should be noted that the optimal control does not depend on the algorithm's iteration as it is not an iterative algorithm. The effective sample size is not 100% because of noise. This way they do not take the exact same paths with exactly the same path weights. If you set the noise to zero all the paths will deterministically end up at the same point with exactly the same path weight. This can be shown by the following example with fictional path weights w:

[2]The implementation of the algorithm can be found in section B
[3]The optimal control had 500 paths whereas the SDC had 10.000 paths

$$w = \left[\tfrac{1}{3}, \tfrac{1}{3}\right] \quad \text{Effss} = \frac{\left(\frac{2}{3}\right)^2}{\left(\frac{1}{3}\right)^2 + \left(\frac{1}{3}\right)^2} = 2 \quad \rightarrow 100\%$$
$$w = [1, 1.1] \quad \text{Effss} = \frac{2.1^2}{1 + 1.1^2} \approx 1.995 \quad \rightarrow 99.77\%$$

Given the effective sample size of the optimal control the path weights must be fairly similar. In figure 3 we can see that the effective sample size goes up from the uncontrolled situation which is the case for the first iteration. This means that the algorithm is learning. After the first two iterations it remains fairly constant. If it were to go down it would be a case of overfitting. The control would then be unnecessarily large without getting a much higher success rate.

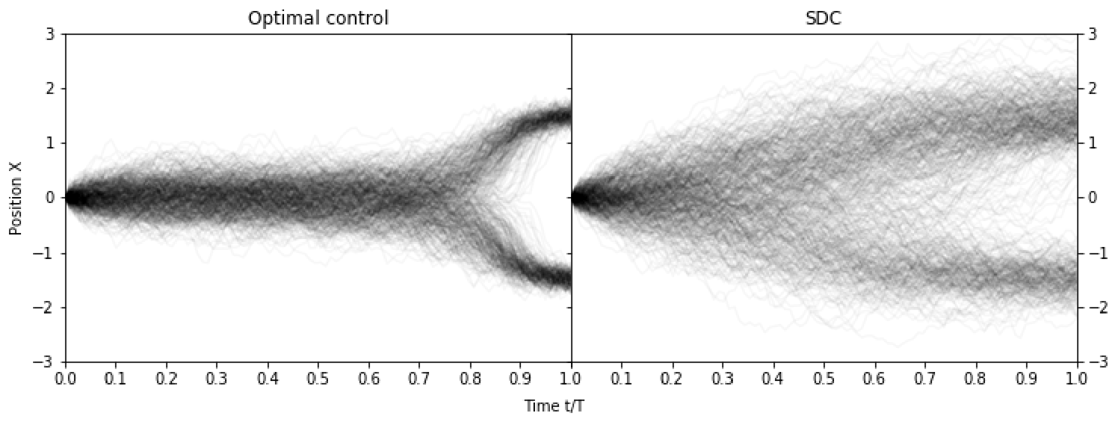


Figure 2: The paths when they are controlled by the optimal controller versus delay lines

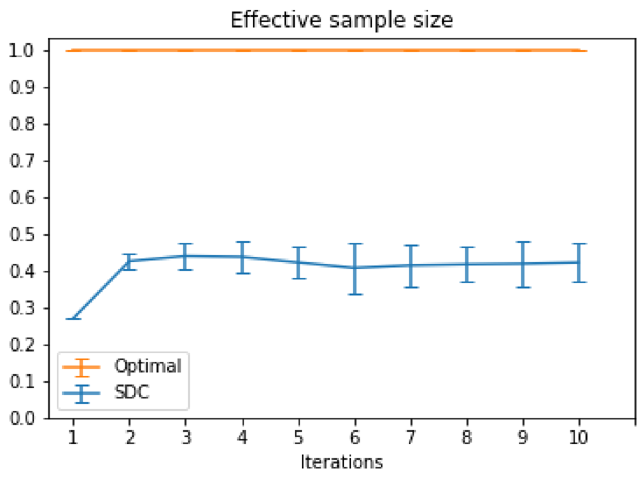


Figure 3: Comparison of the effective sample size of the optimal controller and the SDC

In figure 4 we can see that the optimal control has a success rate of 100%. The SDC quickly reaches his best performance after the third iteration. The iterations thereafter show no change in performance, which is consistent with the effective sample size. A worsening success rate would be reflected in a declining effective sample size.

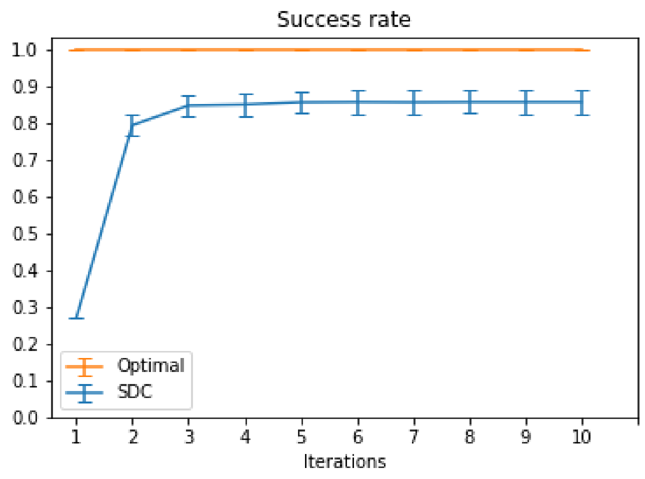


Figure 4: Comparison of the success rate of the optimal controller and the SDC

5. Discussion of the results

As was shown in figures 3 and 4 a time dependent control can be approximated by using a time independent feature based controller using delayed inputs.

It should also be noted that the number of delay lines and the number of basis functions determine how well the algorithm performs and how big the effective sample size is. The fine tuning of these variables could improve the success rate and the effective sample size. In the algorithm the delay lines were updated each discrete time step dt. In reality it would be more realistic to choose dt to be very small and $d\tau$, the time steps at which the delay lines are updated, to be much larger than dt. This way the random paths would seem almost continuous while, for example, a drone determines each N milliseconds its position in order to compute a control. This might increase the success rate but further research would be needed to determine if this is the case.

In figure 3 the effective sample size is normalized since the optimal control takes more time to compute. The time it took to calculate 10.000 paths using the delay lines algorithm was about the same as calculating 500 paths using the optimal control. Admittedly this is mostly because the function to find $\underset{u}{\operatorname{argmin}}()$ was rather naive. It looked at N values between a certain maximum value and minimum value. Where the maximum value should be big enough to not be picked, as it would indicate that the true minimum would fall outside of this range.[4] And with a N large enough to have a precise enough list of possible values, so that a theoretical 'true value' of 4.524 is approximated by 4.51 and not for example 5.2. A possible solution is to use a stopping condition so that it does not check all values of u, but that it stops when the found answer for the next u is larger than the previous value. Another solution would be to use a gradient descent approach, but for this the derivative for equations 38 and 39 needs to be calculated. It would need to account for the fact that there are at most two values of u that minimize the cost-to-go but that it needs to return the lowest value of these two. Otherwise it would steer all paths to 1 endpoint even if it is not the closest one.

[4]If it did fall outside this range the naive argmin function just increased the maximum value and searched again.

Appendices

A. Solution to the path integral

A.1. Path integral with infinitesimal small holes

Two infinitesimal holes are located at $Z = \pm 1$. Calculate the path integral:

$$\psi(x,t) = \int dzq(z,T|x,t)\exp\left(-\phi(z)/\lambda\right) \tag{22}$$

where $q(Z,T|x,t) = \mathcal{N}(z|X,\sqrt{T})$ is a gaussian and

$$\phi(z) = \begin{cases} 0 & \text{if } z = \pm 1 \\ \infty & \text{else} \end{cases} \tag{23}$$

a cost term which equals 0 at the holes in the wall. The solution is then:

$$\psi(x,t) = q(1,T|x,t) + q(-1,T|x,t) \tag{24}$$

A.2. Path integral with finite holes

if we want to solve equation 22 then we will be assuming that the cost term can be written as:

$$\exp\left(-\phi(z)\right) = \exp\left(-\beta(z+p)^2\right) + \exp\left(-\beta(z-p)^2\right) \tag{25}$$

so that there are gaussians centered over the holes positioned at points $\pm p$. This means it is preferable to go through the center of the gaps instead of further away from the center. If we substitute this we get:

$$\psi(x,t) = \int dzq(z,T|x,T)\left(\exp\left(-\beta(z-p)^2\right) + \exp\left(-\beta(z+p)^2\right)\right) \tag{26}$$

where $q(z,T|x,T)$ a gaussian

$$\int dz\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)\left(\exp\left(-\beta(z-p)^2\right) + \exp\left(-\beta(z+p)^2\right)\right) \tag{27}$$

Let's solve the part: $\int dz\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)\exp\left(-\beta(z\pm p)^2\right)$

$$\begin{aligned} &= \int dz\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(z-\mu)^2}{2\sigma^2}\right)\exp\left(-\beta(z\pm p)^2\right) \\ &= \int dz\frac{1}{\sqrt{2\pi\sigma^2}}\exp\left(-\frac{(z-\mu)^2}{2\sigma^2} - \beta(z\pm p)^2\right) \end{aligned} \tag{28}$$

call $\gamma \equiv 2\sigma^2$ with $\gamma \in \mathscr{R}_{\geq 0}$ then:

$$= \int dz \frac{1}{\sqrt{\pi\gamma}} \exp\left(-\frac{(z-\mu)^2 + \gamma\beta(z \pm p)^2}{\gamma}\right) \tag{29}$$

Let's rewrite the exponent by completing the square. If we look at the numerator in the exponent:

$$\begin{aligned} \Pi \equiv (z-\mu)^2 + \gamma\beta(z \pm p)^2 &= z^2 + \mu^2 - 2\mu z + \gamma\beta(z^2 + p^2 \pm 2zp) \\ &= (1+\gamma\beta)z^2 + \mu^2 + \gamma\beta p^2 - 2(\mu \mp \gamma\beta p)z \end{aligned} \tag{30}$$

call $\nu \equiv \gamma\beta$ with $\nu \in \mathscr{R}_{\geq 0}$ then:

$$\Pi = (1+\nu)z^2 - 2(\mu \mp \nu p)z + \mu^2 + \nu p^2 \tag{31}$$

call $\varepsilon \equiv \frac{1}{1+\nu}$ with $\varepsilon \in \mathscr{R}_{\geq 0}$ then multiply Π with ε:

$$\Pi\varepsilon = z^2 - 2(\mu \mp \nu p)\varepsilon z + \varepsilon\mu^2 + \varepsilon\nu p^2 \tag{32}$$

Solving by completing the square

$$(z - (\mu \mp \nu p)\varepsilon)^2 = z^2 - 2(\mu \mp \nu p)\varepsilon z + (\mu \mp \nu p)^2\varepsilon^2 \tag{33}$$

$$z^2 - 2(\mu \mp \nu p)\varepsilon z = (z - (\mu \mp \nu p)\varepsilon)^2 - (\mu \mp \nu p)^2\varepsilon^2 \tag{34}$$

if we compare to equation 32 we can rewrite it as:

$$\Pi\varepsilon = (z - (\mu \mp \nu p)\varepsilon)^2 - \left((\mu \mp \nu p)^2\varepsilon + \mu^2 + \nu p^2\right)\varepsilon \tag{35}$$

if we then divide by ε and put the numerator back we get:

$$\begin{aligned} &\exp\left(-\frac{(1+\nu)(z-(\mu \mp \nu p)\varepsilon)^2 - (\mu \mp \nu p)^2\varepsilon + \mu^2 + \nu p^2}{\gamma}\right) \\ &\exp\left(-\frac{(1+\nu)(z-(\mu \mp \nu p)\varepsilon)^2}{\gamma}\right)\exp\left(-\frac{-(\mu \mp \nu p)^2\varepsilon + \mu^2 + \nu p^2}{\gamma}\right) \end{aligned} \tag{36}$$

if we then use this in equation 26:

$$\begin{aligned} \psi(x,t) = &\frac{1}{\sqrt{\pi\gamma}} \exp\left(-\frac{-(\mu - \nu p)^2\varepsilon + \mu^2 + \nu p^2}{\gamma}\right) \int dz \exp\left(-\frac{(1+\nu)(z-(\mu - \nu p)\varepsilon)^2}{\gamma}\right) \\ &+ \frac{1}{\sqrt{\pi\gamma}} \exp\left(-\frac{-(\mu + \nu p)^2\varepsilon + \mu^2 + \nu p^2}{\gamma}\right) \int dz \exp\left(-\frac{(1+\nu)(z-(\mu + \nu p)\varepsilon)^2}{\gamma}\right) \end{aligned} \tag{37}$$

substitute $(\mu \pm \nu p)\varepsilon = C_{x\pm}$ and $\frac{1+\nu}{\gamma} = \frac{1}{N}$

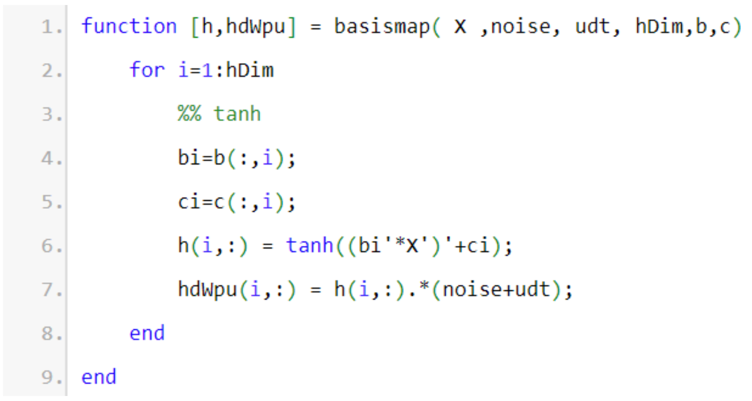
$$\begin{aligned}\psi(x,t) = &\frac{1}{\sqrt{\pi\gamma}}\exp\left(-\frac{-(\mu-\nu p)^2\varepsilon+\mu^2+\nu p^2}{\gamma}\right)\int_a^b dz\exp\left(-\frac{(z-C_{x-})^2}{N}\right)\\ &+\frac{1}{\sqrt{\pi\gamma}}\exp\left(-\frac{-(\mu+\nu p)^2\varepsilon+\mu^2+\nu p^2}{\gamma}\right)\int_c^d dz\exp\left(-\frac{(z-C_{x+})^2}{N}\right)\end{aligned} \tag{38}$$

$$\int_a^b \exp\left(-\frac{(z-C_\pm)^2}{N}\right) = \frac{1}{2}\sqrt{N\pi}\left(\operatorname{erf}\left(\frac{C_\pm-a}{\sqrt{N}}\right)-\operatorname{erf}\left(\frac{C_\pm-b}{\sqrt{N}}\right)\right) \tag{39}$$

where $C_{x\pm}$ and μ depend on x. In this case $\mu = x$ as the noise comes from zero mean centered random variables.

B. The algorithm written in Matlab

The function in figure 5 is used to calculate the terms $h(dW+u)$ and the term h in equation (11). With these variables a new A can be calculated which in turn can be used to calculate the new control. The term $h(x)$ is a matrix with dimensions (*"the number of paths"*, *"the number of basis functions"*). Furthermore the terms b,c are zero mean random variables. The variables c_{ij} are the same for every path i but different per basis function j. It should be noted that $b = (rand(Nlines, hDim)*2-1)/10$ where Nlines is the number $Ndelay+1$ as it also includes the current x position, and hDim is the number of basis functions.

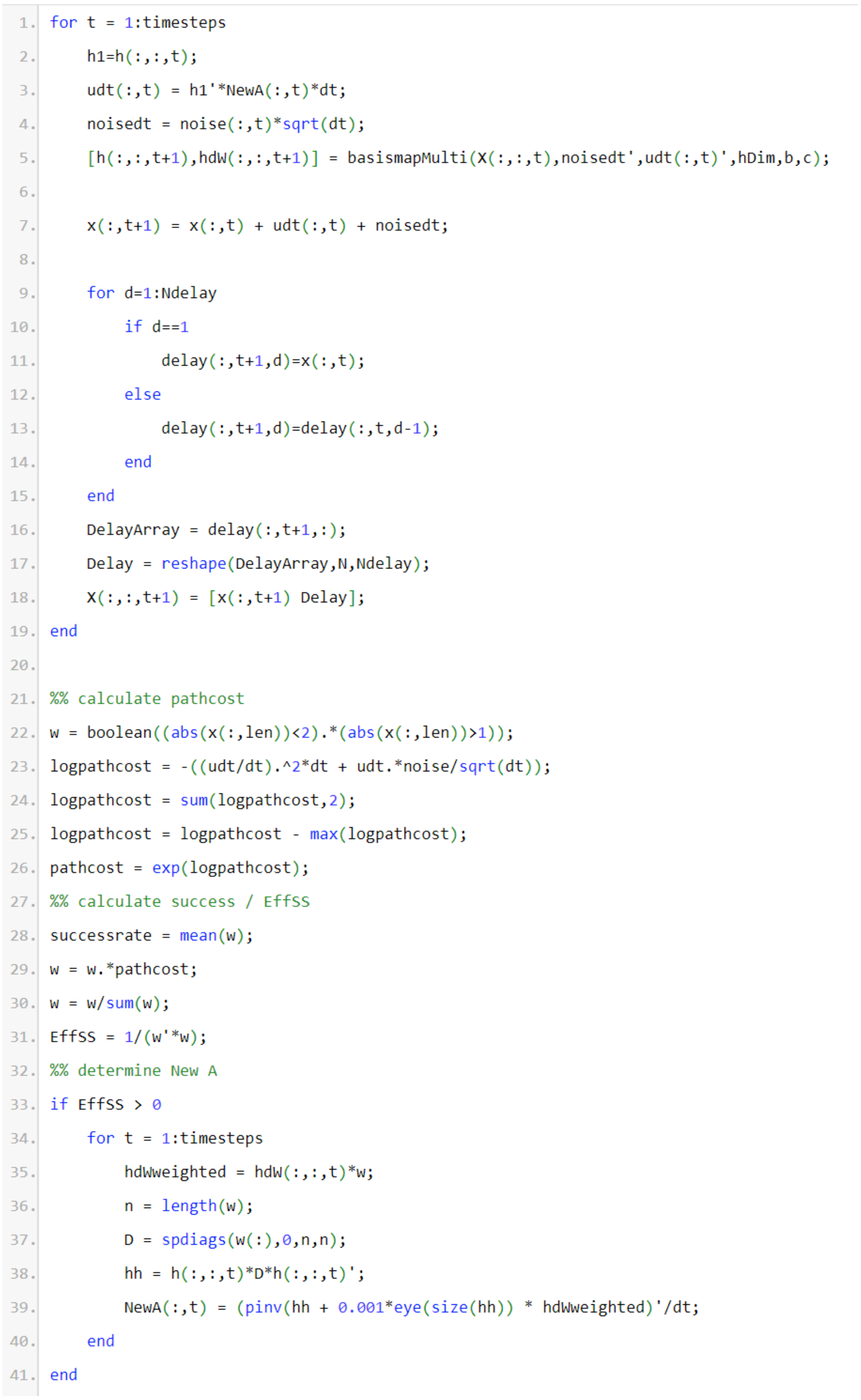


```
function [h,hdWpu] = basismap( X ,noise, udt, hDim,b,c)
    for i=1:hDim
        %% tanh
        bi=b(:,i);
        ci=c(:,i);
        h(i,:) = tanh((bi'*X')'+ci);
        hdWpu(i,:) = h(i,:).*(noise+udt);
    end
end
```

Figure 5: The function used in the algorithm

The algorithm described in figure 6 is only one iteration of the algorithm. The whole algorithm needs to be put in a loop to keep on finding new expressions for A in equation (11) which can be used to find a new and hopefully better control term in equation (12).

The matrix X in line 18 is a matrix consisting of delay lines as expressed in section 2.2. Only the successful paths ending in one of the gaps are important. So in line 22 we only look if a path ends in a gap and give it the value 1, otherwise it is 0. In line 29 we multiply this with the pathcosts. Doing it this way we can avoid using infinities in the end-cost $\Phi(X(t_f))$ as we saw in equation (8). Also note that in equation (11) $\langle hh'\rangle^{-1}$ is calculated by the Moore-Penrose pseudo inverse *pinv()*. A further discussion of the pseudo inverse can be found in section B.1.



```
for t = 1:timesteps
    h1=h(:,:,t);
    udt(:,t) = h1'*NewA(:,t)*dt;
    noisedt = noise(:,t)*sqrt(dt);
    [h(:,:,t+1),hdW(:,:,t+1)] = basismapMulti(X(:,:,t),noisedt',udt(:,t)',hDim,b,c);

    x(:,t+1) = x(:,t) + udt(:,t) + noisedt;

    for d=1:Ndelay
        if d==1
            delay(:,t+1,d)=x(:,t);
        else
            delay(:,t+1,d)=delay(:,t,d-1);
        end
    end
    DelayArray = delay(:,t+1,:);
    Delay = reshape(DelayArray,N,Ndelay);
    X(:,:,t+1) = [x(:,t+1) Delay];
end

%% calculate pathcost
w = boolean((abs(x(:,len))<2).*(abs(x(:,len))>1));
logpathcost = -((udt/dt).^2*dt + udt.*noise/sqrt(dt));
logpathcost = sum(logpathcost,2);
logpathcost = logpathcost - max(logpathcost);
pathcost = exp(logpathcost);
%% calculate success / EffSS
successrate = mean(w);
w = w.*pathcost;
w = w/sum(w);
EffSS = 1/(w'*w);
%% determine New A
if EffSS > 0
    for t = 1:timesteps
        hdWweighted = hdW(:,:,t)*w;
        n = length(w);
        D = spdiags(w(:),0,n,n);
        hh = h(:,:,t)*D*h(:,:,t)';
        NewA(:,t) = (pinv(hh + 0.001*eye(size(hh)) * hdWweighted)'/dt;
    end
end
```

Figure 6: The main algorithm.

B.1. Modifications to the algorithm

The original algorithm used equation 11 to determine the matrix A which is used to compute the control u. This would be equivalent to matlab code as:

$$\text{A = (pinv(hh)*hdWweighted)}'\text{/dt} \tag{40}$$

However this leads to a few extraordinarily large values in A. This does not change the algorithm's ability to learn but some paths will be steered off-course to undesirable large values of x. An easy fix is to add an identity matrix multiplied by a small number. So that it no longer tries to invert

$$\text{A = (pinv(hh + 0.001*eye(size(hh)))*hdWweighted)}'\text{/dt} \tag{41}$$

where $eye()$ is the matlab function for an identity matrix.

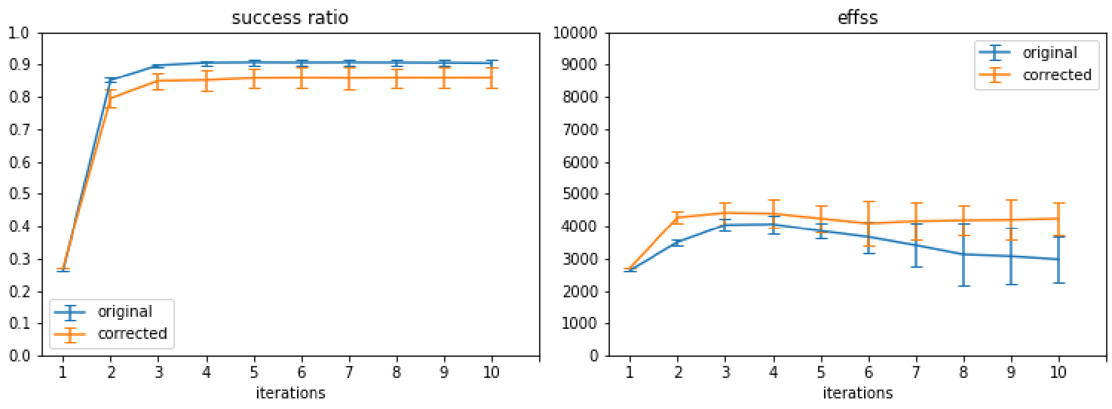


Figure 7: Comparing the EffSS and success at $x = 0$

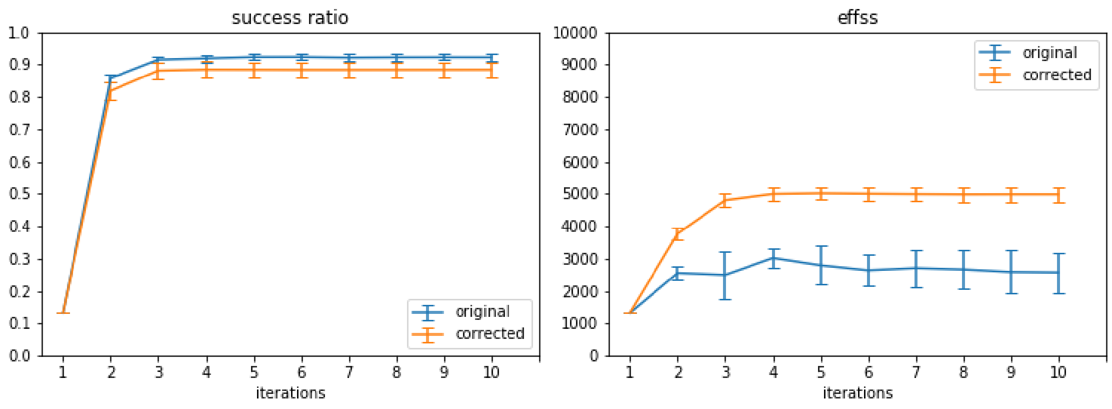


Figure 8: Comparing the EffSS and success at $x = -3$

We can see that the corrected version has a little lower success rate ($85,9 \pm 3,1\%$ versus $90,6 \pm 0,8\%$), but that for values $x \neq 0$, and in this specific case $x = -3$ the effective sample size is significantly higher for the corrected version. Now if we specifically look at the values of A in figure 9, represented by logarithmic boxplots, we can see that in

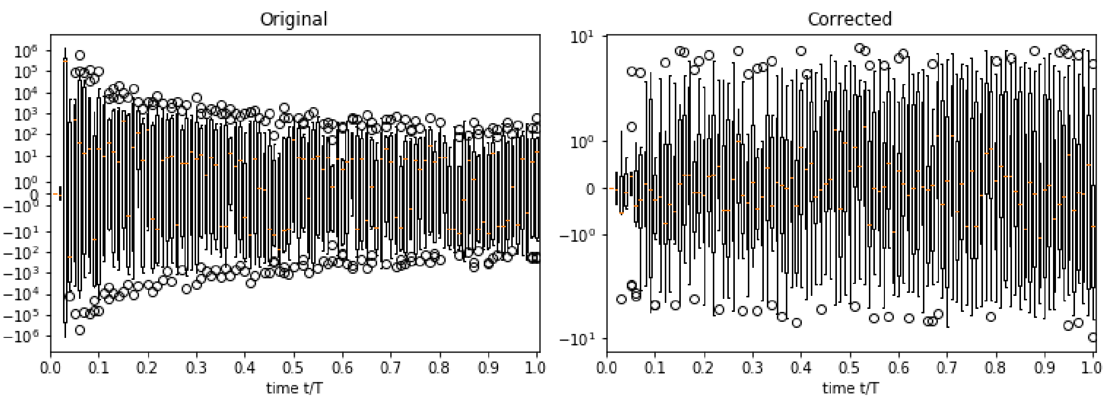


Figure 9: comparing values of A over time using boxplots

the original version the outliers and even the majority of the values differ many orders of magnitude. Whereas in the corrected version it is only about one order of magnitude.

C. About the cover page

The image on the cover page shows what the result is when Path Integral Control is applied. On the left side we see the result of the optimal control, and on the right side we see the result of applying the algorithm.

References

[1] C. Bishop, *Pattern recognition and machine learning*, ch. 3, p. 138. New York: Springer, 2006.

[2] S. Thijssen, *Path Integral Control*. PhD thesis, Radboud University, 28 nov 2016.

[3] S. Thijssen and H. J. Kappen, “Path integral control and state-dependent feedback,” *Phys. Rev. E*, vol. 91, p. 032104, Mar 2015.