studywolf

a blog for things I encounter while coding and researching neuroscience, motor control, and learning

NOV 16 2013

60 COMMENTS

BY TRAVISDEWOLF DYNAMIC MOVEMENT PRIMITIVE, MOTOR CONTROL, ROBOTICS

Dynamic movement primitives part 1: The basics

Dynamic movement primitives (DMPs) are a method of trajectory control / planning from Stefan Schaal's (http://www-clmc.usc.edu/~sschaal/) lab. They were presented way back in 2002 in this paper (http://citeseerx.ist.psu.edu/viewdoc/summary?doi=10.1.1.142.3886), and then updated in 2013 by Auke Ijspeert in this paper (http://www-clmc.usc.edu/publications/I/ijspeert-NC2013.pdf). This work was motivated by the desire to find a way to represent complex motor actions that can be flexibly adjusted without manual parameter tuning or having to worry about instability.

Complex movements have long been thought to be composed of sets of primitive action 'building blocks' executed in sequence and \ or in parallel, and DMPs are a proposed mathematical formalization of these primitives. The difference between DMPs and previously proposed building blocks is that each DMP is a nonlinear dynamical system. The basic idea is that you take a dynamical system with well specified, stable behaviour and add another term that makes it follow some interesting trajectory as it goes about its business. There are two kinds of DMPs: discrete and rhythmic. For discrete movements the base system is a point attractor, and for rhythmic movements a limit cycle is used. In this post we're only going to worry about discrete DMPs, because by the time we get through all the basics this will already be a long post.

Imagine that you have two systems: An imaginary system where you plan trajectories, and a real system where you carry them out. When you use a DMP what you're doing is planning a trajectory for your real system to follow. A DMP has its own set of dynamics, and by setting up your DMP properly you can get the control signal for your actual system to follow. If our DMP system is planing a path for the hand to follow, then what gets sent to the real system is the set of forces that need to be applied to the hand. It's up to the real system to take these hand forces and apply them, by converting them down to joint torques or muscle activations (through something like the operation space control framework) or whatever. That's pretty much all I'll say here about the real system, what we're going to focus on here is the DMP system. But keep in mind that the whole DMP framework is for generating a trajectory \ control signal to guide the real system.

I've got the code for the basic discrete DMP setup and examples I work through in this post up on my github (https://github.com/studywolf/pydmps), so if you want to jump straight to that, there's the link! You can run test code for each class just by executing that file.

Discrete DMPs

Let's start out with point attractor dynamics:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}),$$

where y is our system state, g is the goal, and α and β are gain terms. This should look very familiar, it's a PD control signal, all this is going to do is draw our system to the target. Now what we'll do is add on a *forcing* term that will let us modify this trajectory:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f.$$

How to define a nonlinear function f such that you get the desire behaviour is a non-trivial question. The crux of the DMP framework is an additional nonlinear system used to define the forcing function f over time, giving the problem a well defined structure that can be solved in a straight-forward way and easily generalizes. The introduced system is called the *canonical dynamical system*, is denoted x, and has very simple dynamics:

$$\dot{x} = -\alpha_x x.$$

The forcing function f is defined as a function of the canonical system:

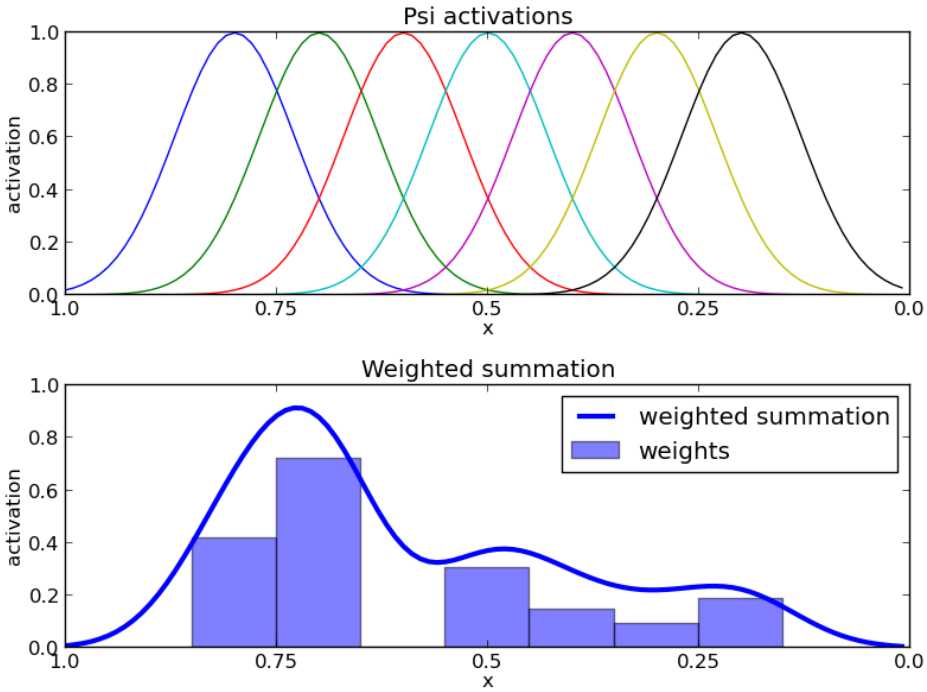
$$f(x, g) = \frac{\Sigma_{i=1}^{N} \psi_i w_i}{\Sigma_{i=1}^{N} \psi_i} x(g - y_0),$$

where y_0 is the initial position of the system,

$$\psi_i = \exp\left(-h_i \left(x - c_i\right)^2\right),$$

and w_i is a weighting for a given basis function ψ_i. You may recognize that the ψ_i equation above defines a Gaussian centered at c_i, where h_i is the variance. So our forcing function is a set of Gaussians that are 'activated' as the canonical system x converges to its target. Their weighted summation is normalized, and then multiplied by the $x(g - y_0)$ term, which is both a 'diminishing' and spatial scaling term.

Let's break this down a bit. The canonical system starts at some arbitrary value, throughout this post $x_0 = 1$, and goes to 0 as time goes to infinity. For right now, let's pretend that x decays linearly to 0. The first main point is that there are some basis functions which are activated as a function of x, this is displayed in the top figure below. As the value of x decreases from 1 to 0, each of the Gaussians are activated (or centered) around different x values. The second thing is that each of these basis functions are also assigned a weight, w_i. These weights are displayed in the lower figure in the bar plot. The output of the forcing function f is then the summation of the activations of these basis functions multiplied by their weight, also displayed in the lower figure below.



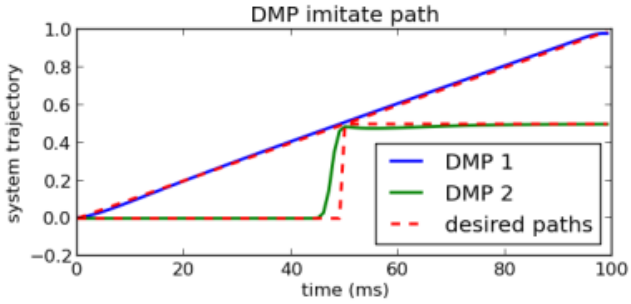
(https://studywolf.files.wordpress.com/2013/11/psi3.png)

The diminishing term

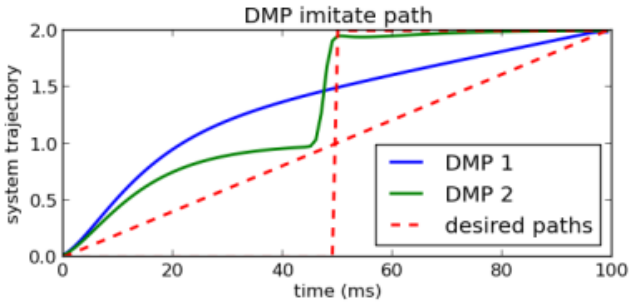
Incorporating the x term into the forcing function guarantees that the contribution of the forcing term goes to zero over time, as the canonical system does. This means that we can sleep easy at night knowing that our system can trace out some crazy path, and regardless will eventually return to its simpler point attractor dynamics and converge to the target.

Spatial scaling

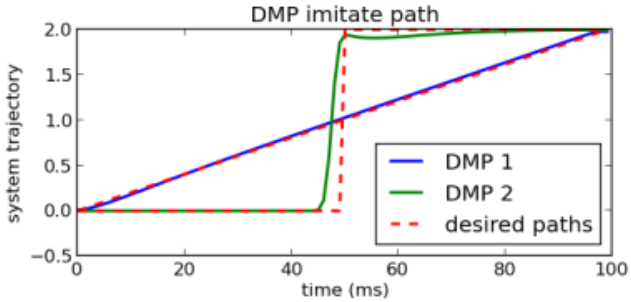
Spatial scaling means that once we've set up the system to follow a desired trajectory to a specific goal we would like to be able to move that goal farther away or closer in and get a scaled version of our trajectory. This is what the $(g - y_0)$ term of the forcing function handles, by scaling the activation of each of these basis functions to be relative to the distance to the target, causing the system to cover more or less distance. For example, let's say that we have a set of discrete DMPs set up to follow a given trajectory:



(https://studywolf.files.wordpress.com/2013/11/dmpimitatedpath1.png)
The goals in this case are 1 and .5, which you can see is where the DMPs end up. Now, we've specified everything in this case for these particular goals (1 and .5), but let's say we'd like to now generalize and get a scaled up version of this trajectory for moving by DMPs to a goal of 2. If we don't appropriately scale our forcing function, with the $(g - y_0)$ term, then we end up with this:

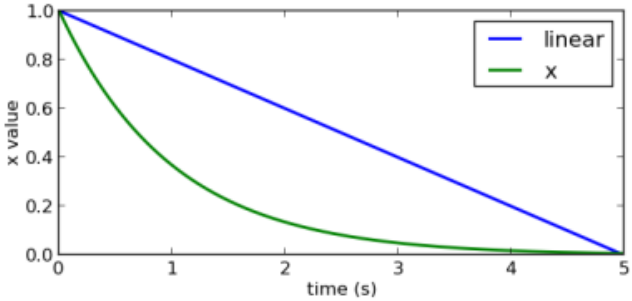


(https://studywolf.files.wordpress.com/2013/11/dmpnogscaling1.png)
Basically what's happened is that for these new goals the same weightings of the basis functions were too weak to get the system to follow or desired trajectory. Once the $(g - y_0)$ term included in the forcing function, however, we get:



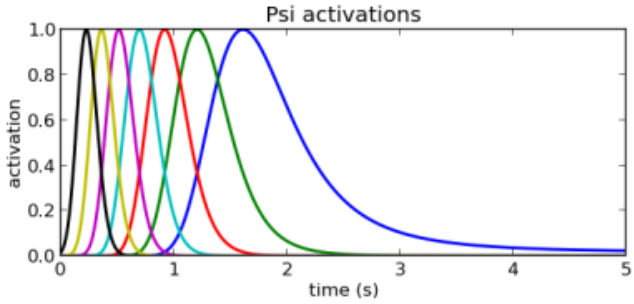
(https://studywolf.files.wordpress.com/2013/11/dmpwithgscaling1.png)
which is exactly what we want! Our movements now scale spatially. Awesome.

Spreading basis function centers
Alright, now, unfortunately for us, our canonical system does not converge linearly to the target, as we assumed above. Here's a comparison of a linear decay vs the exponential decay of actual system:



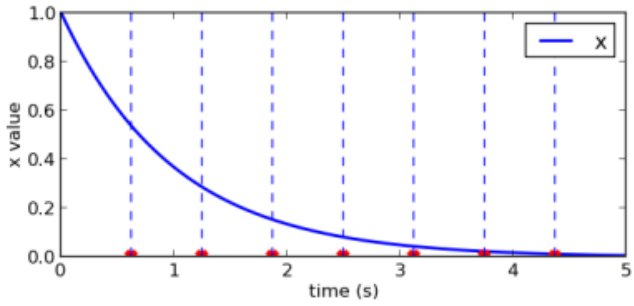
(https://studywolf.files.wordpress.com/2013/11/xvspd2.png)
This is an issue because our basis functions activate dependent on x. If the system was linear then we would be fine and the basis

function activations would be well spread out as the system converged to the target. But, with the actual dynamics, x is not a linear function of time. When we plot the basis function activations as a function of time, we see that the majority are activated immediately as x moves quickly at the beginning, and then the activations stretch out as the x slows down at the end:



(https://studywolf.files.wordpress.com/2013/11/gauss_over_time2.png)

In the interest of having the basis functions spaced out more evenly through time (so that our forcing function can still move the system along interesting paths as it nears the target, we need to choose our Gaussian center points more shrewdly. If we look at the values of x over time, we can choose the times that we want the Gaussians to be activated, and then work backwards to find the corresponding x value that will give us activation at that time. So, let's look at a picture:

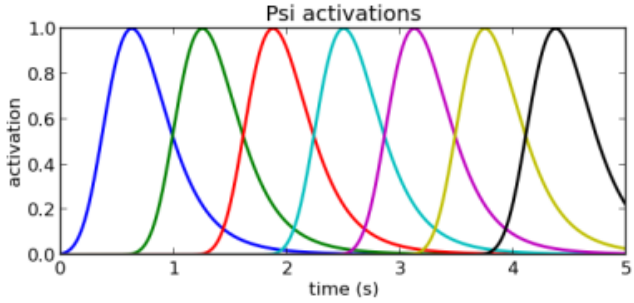


(https://studywolf.files.wordpress.com/2013/11/des_gauss_centers1.png)

The red dots are the times we'd like the Gaussians to be centered around, and the blue line is our canonical system x. Following the dotted lines up to the corresponding x values we see what values of x the Gaussians need to be centered around. Additionally, we need to worry a bit about the width of each of the Gaussians, because those activated later will be activated for longer periods of time. To even it out the later basis function widths should be smaller. Through the very nonanalytical method of trial and error I've come to calculate the variance as

$$h_i = \frac{\#BFs}{c_i}.$$

Which reads the variance of basis function i is equal to the number of basis functions divided by the center of that basis function. When we do this, we can now generate centers for our basis functions that are well spaced out:



(https://studywolf.files.wordpress.com/2013/11/gauss_over_time_spaced_well2.png)

Temporal scaling

Again, generalizability is one of the really important things that we want out of this system. There are two obvious kinds, temporal and spatial. Spatial scaling we discussed above, in the temporal case we'd like to be able to follow this same trajectory at different speeds. Sometimes quick, sometimes slow, but always tracing out the same path. To do that we're going to add another term to our system dynamics, τ, our temporal scaling term. Given that our system dynamics are:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}) + f,$$
$$\dot{x} = -\alpha_x x,$$

to give us temporal flexibility we can add the τ term:

$$\dot{y} \mathrel{+}= \tau \ddot{y},$$

$$y \mathrel{+}= \tau \dot{y},$$

$$x \mathrel{+}= \tau \dot{x},$$

and that's all we have to do! Now to slow down the system you set τ between 0 and 1, and to speed it up you set τ greater than 1.

Imitating a desired path

Alright, great. We have a forcing term that can make the system take a weird path as it converges to a target point, and temporal and spatial scalability. How do we set up the system to follow a path that we specify? That would be ideal, to show the system the path to follow, and have it be able to work backwards and figure out the forces and then be able to generate that trajectory whenever we want. This ends up being a pretty straight forward process.

We have control over the forcing term, which affects the system acceleration. So we first need to take our desired trajectory, $\mathbf{y}_d$ (where bold denotes a vector, in this case the time series of desired points in the trajectory), and differentiate it twice to get the accelerations:

$$\ddot{\mathbf{y}}_d = \frac{\partial}{\partial t}\dot{\mathbf{y}}_d = \frac{\partial}{\partial t}\frac{\partial}{\partial t}\mathbf{y}_d.$$

Once we have the desired acceleration trajectory, we need to remove the effect of the base point attractor system. We have the equation above for exactly what the acceleration induced by the point attractor system at each point in time is:

$$\ddot{y} = \alpha_y(\beta_y(g - y) - \dot{y}),$$

so then to calculate what the forcing term needs to be generate this trajectory we have:

$$\mathbf{f}_d = \ddot{\mathbf{y}}_d - \alpha_y(\beta_y(g - \mathbf{y}) - \dot{\mathbf{y}}).$$

From here we know that the forcing term is comprised of a weighted summation of basis functions which are activated through time, so we can use an optimization technique like locally weighted regression to choose the weights over our basis functions such that the forcing function matches the desired trajectory $\mathbf{f}_d$. In locally weighted regression sets up to minimize:

$$\Sigma_t \psi_i(t)(f_d(t) - w_i(x(t)(g - y_0)))^2$$

and the solution (which I won't derive here, but is worked through in Schaal's 1998 paper (http://roland.pri.ee/doktor/papers/LWL/LWPRreference.pdf) (https://docs.google.com/viewer?url=http%3A%2F%2Froland.pri.ee%2Fdoktor%2Fpapers%2FLWL%2FLWPRreference.pdf&embedded=true&chrome=false&dov=1)) is

$$w_i = \frac{\mathbf{s}^T \boldsymbol{\psi}_i \mathbf{f}_d}{\mathbf{s}^T \boldsymbol{\psi}_i \mathbf{s}},$$

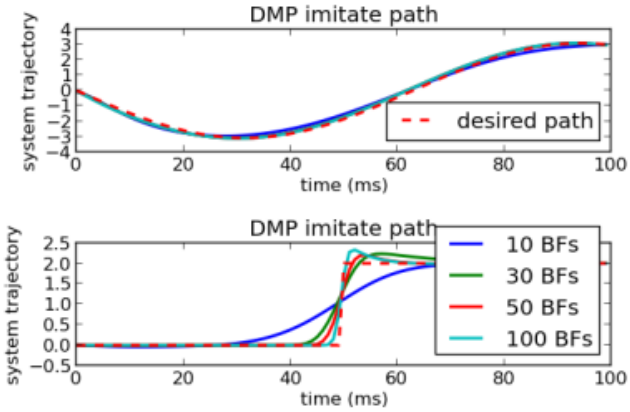
where

$$\mathbf{s} = \begin{pmatrix} x_{t_0}(g - y_0) \\ \vdots \\ x_{t_N}(g - y_0) \end{pmatrix}, \quad \boldsymbol{\psi}_i = \begin{pmatrix} \psi_i(t_0) & \dots & 0 \\ 0 & \ddots & 0 \\ 0 & \dots & \psi_i(t_n) \end{pmatrix}$$

Great! Now we have everything we need to start making some basic discrete DMPs!

Different numbers of basis functions

One of the things you'll notice right off the bat when imitating paths, is that as the complexity of the trajectory increases, so does the required number of basis functions. For example, below, the system is trying to follow a sine wave and a highly nonlinear piecewise function:



(https://studywolf.files.wordpress.com/2013/11/dmpdiffbfnums1.png)
We can see in the second case that although the DMP is never able to exactly reproduce the desired trajectory, the approximation continues to get better as the number of basis functions increases. This kind of slow improvement in certain nonlinear areas is to be expected from how the basis functions are being placed. An even spreading of the centers of the basis functions through time was used, but for imitation there is another method out of Dr. Schaal's lab that places the basis functions more strategically. Need is determined by the function complexity is in that region, and basis function centers and widths are defined accordingly. In highly nonlinear areas we would expect there to be many narrow basis functions, and in linear areas we would expect fewer basis functions, but ones that are wider. The method is called locally weighted projection regression, which I plan on writing about and applying in a future post!

Conclusions \ thoughts

There's really a lot of power in this framework, and there are a ton of expansions on this basic setup, including things like incorporating system feedback, spatio-temporal coupling of DMPs, using DMPs for gain control as well as trajectory control, incorporating a cost function and reinforcement learning, identifying action types, and other really exciting stuff.

I deviated from the terminology used in the papers here in a couple of places. First, I didn't see a reason to reduce the second order systems to two first order systems. When working through it I found it more confusing than helpful, so I left the dynamics as a second order systems. Second, I also moved the τ term to the right hand side, and that's just so that it matches the code, it doesn't really matter. Neither of these were big changes, but in case you're reading the papers and wondering.

Something that I kind of skirted above is planning along multiple dimensions. It's actually very simple; the DMP framework simply assigns one DMP per degree of freedom being controlled. But, it's definitely worth explicitly stating at some point.

I also mentioned this above, but this is a great trajectory control system to throw on top of the previously discussed operational space control framework. With the DMP framework on top to plan robust, generalizable movements, and the OSCs underneath to carry out those commands we can start to get some really neat applications. For use on real systems the incorporation of feedback and spatio-temporal coupling terms is going to be important, so the next post will likely be working through those and then we can start looking at some exciting implementations!

Speaking of implementations, there's a DMP and canonical system code up on my github (https://github.com/studywolf/pydmps), please feel free to explore it, run it, send me questions about it. Whatever. I should also mention that there's this and a lot more all up and implemented on Stefan Schaal's lab website (http://www-clmc.usc.edu/Resources/Software).
Tagged Dynamic movement primitives, high-level controller, motor control, robot control, trajectory planning

60 thoughts on "Dynamic movement primitives part 1: The basics"

Anne says:

March 20, 2014 at 10:19 am

Hi,
I found this blog really helpful, thanks for posting! I'm working on my master's thesis and used this to define my basis functions in a DMP.
I actually noticed one thing. Instead of using h_{i} = #BFs/c_{i}, I found out that for my code h_{i} = #BFs/c_{i}^2 works much better to even out the activations. I was actually wondering if you have any explanation why two problems can both have a clear relation which is different?

Reply

travisdewolf says:

March 20, 2014 at 12:00 pm

Ah! This is definitely the least well-tested part of my implementation. I've gone back and forth testing different variances, the one that I settled on and am actually using is something like h_{i} = #BFs^1.5 / c_{i} in the code on my github. But it really would be useful to have a better reason for that than 'it works pretty good'. I've added it (again) to my to do list, but I'd be very interested to hear about any tests that you've run!

As far as why so many mostly work, it's just a numbers matter If you throw enough basis functions at the problem as long as they sufficiently tile the space you'll be able to do a good approximation of a lot of functions (that are smooth and continuous etc etc). Dr. Schaal has addressed the optimal placement of basis function centers based on the non-linearity of what you're trying to approximate, but for the general case I'm not sure.

Reply

Fotis says:

September 5, 2017 at 2:52 am

Your explanation and examples made me comprehend DMPs! Is there any reference for optimal placement of basis functions? I was also wondering if there is any approach to add BFs when needed, while training online a DMP to a trajectory that is being captured by sensors.

travisdewolf says:

October 26, 2017 at 8:54 am

Hello! Hmm I remember playing around with basis function placement based on the derivative of the trajectory (so you place more basis functions where there's a large derivative to better model the changes), and I believe there is a bunch of work looking at this but I don't know anything specific to cite. Along those lines, for the training you could solve the least squares problem (i.e. the same method done now) and then check out the difference between the desired trajectory and the DMP generated trajectory and place more basis functions using that information. For online training of the DMPs I've heard that Dr. Freek Stulp has done work training with stochastic gradient descent, and there's the PIPI method from Dr. Evangelos Theodorou that you might find interesting!
Cheers,

Arne says:

April 11, 2014 at 10:00 am

Wow! This is by far the best explanation that I have found on DMPs. Far better than the original papers. Thanks!

Reply

travisdewolf says:

April 11, 2014 at 11:05 am

Glad that you found the posts useful! 😀

Reply

javier says:

April 11, 2014 at 1:47 pm

i have a doubt with the \tau temporal scaling term. In the Ijspeert reference they add the \tau to the lef term of the equation and you add it to the right term, this is ok,is just a constant, it doesn' t matter if what you call \tau is equivalent to their \tau^{-1}. My problem is when they use the first order notation. They call z= \tau dy.-> dz=tau ddy. Substitutng this in equation dz gives \tau^2{ddy} (a constant multiplying ddy) but also a \tau dy. So in they second order notation they have a temporal constant multiplying ddy but in they first order notation they have a constant x ddy but also a constant x dy.

Reply

travisdewolf says:

April 11, 2014 at 2:15 pm

Ah, you're right, I should have specified that in the equations, it's in the code! Thanks for the catch, I'll change it now!

Reply

javier says:

April 11, 2014 at 3:11 pm

my thought is? is their second order notation equivalent to their first order notation? In their second order notation the critical dumping contidion is alpha_z= 4 beta_z tau (tau dependent). But in their first order notation the critical dumping condition is alfa_z=4 beta_z (tau invariant).

Reply

travisdewolf says:

April 11, 2014 at 4:28 pm

Hmm, I'm not sure I see why the gains need to be set up differently in the two cases. When I'm implementing this in code it doesn't make a difference if I multiply by \tau when determining \dot{z} and then multiply by \tau again when I add it to \dot{y} or if I just multiply \ddot{y} by \tau^2 in the second order implementation.

Reply

javier says:

April 12, 2014 at 4:32 am

On the other hand, how do you choose the value for alpha_z. A bigger value will make the system follow faster the input, however for a given yd will make the |fd| higher and the |w_i| will be higher too. Has bigger |w_i| a drawback?

Reply

travisdewolf says:

April 14, 2014 at 11:02 am

I haven't found the particular values of \alpha_z to be too important, since when you're calculating the path that you want the system to follow you explicitly take them into account and compensate for them. There's also \tau for temporal scaling, so I really haven't explored different \alpha_z values all too much. If you end up looking at it please let me know what you find!

Reply

abbas says:

March 10, 2015 at 11:23 am

Hi,

Thanks for the Post. The basic version of DMP assumes "Zero Velocity" at the target goal, is it straightforward to assume "non Zero velocity" at the target goal for the system? I want to learn a kick behavior for a humanoid soccer robot using DMPs, But this "Zero Velocity" assumption limits the potenitial of the method for my application.

Thanks
Abbas

Reply

travisdewolf says:

March 11, 2015 at 10:50 am

Hi Abbas,

thanks for the question! This is really interesting. So, in this paper on pages 5 and 6 they talk about some different examples of similar situations, bouncing a ball, hitting a tennis ball, etc. There are more details for the bouncing ball here and the details for robot table tennis here, which is more applicable because kicking a ball isn't a rhythmic movement usually. Actually, this paper from Jan Peters might be the most appropriate, seems like fortunately there are a few detailed papers about similar situations!

I would also think that since you're doing kicking balance is an issue, so you might be interested in looking at controlling in the null space to prioritize balance and then do the kicking motion without falling over.
Hope this helps, good luck! And please let me know how it goes!
Thanks!

Reply

Matteo says:

April 28, 2015 at 11:10 am

Hi,congratulations for your blog. I'm doing a thesis about this topic and your explanation is really helpful. I have two question about this part:
1 – at this line -> where we use \tau^2 for \ddot{y} because it's the second derivative
I don't understand. If I have used tau and not tau^2,what would change in the behaviour?
2 – at this line -> such that the forcing function matches the desired trajectory {f}_d
It's an error?It could be yd,right?

Reply

travisdewolf says:

April 29, 2015 at 1:11 pm

Hi Matteo, thanks! For your questions:

1) If you pull up the Ijspeert paper and look at equation 2.8 you can see how the system is formed with temporal scaling when you've broken it up into two first-order equations. For simplicity I merged the two equations into a second order equation here, and when you do that you end up with the acceleration being multiplied by tau squared. Does that help?

2) Ah this is a little confusing, the desired trajectory is now in terms of a desired _force_ trajectory, since that's what the contribution of the weighted sum of the basis functions will be contributing to the system. So we took the desired trajectory of positions, y_d, and calculated the trajectory of desired forces that we need to implement y_d, and that's what I'm referring to there.

Thanks for the questions, please let me know if that's not clear!

Reply

Burooj Ghani says:

May 24, 2015 at 8:22 am

Hi, I am writing my master thesis right now. I found this very beneficial. Thank you.

P.S. I am still wondering if there is a possibility to cite you (In latex). I mean if I use some ideas from this article, should I cite S. Schaal's paper mentioned in the beginning or is there any way to cite this article.

Reply

travisdewolf says:

May 27, 2015 at 6:43 am

Hi Burooj,
glad you found it useful! Thanks!
The ideas from this post are all from the papers that I mention at the beginning of the post, but if you'd still like to cite the blog post in APA form it would be "DeWolf, Travis. 'Dynamic Movement Primitives Part 1: The Basics'. Studywolf 2015. Web. 27 May 2015." Where "27 May 2015" would get replaced with the day that you accessed the article.
Cheers!

Reply

Michi says:

June 4, 2015 at 9:08 pm

Hi, Thank you for good article!! It's quite clear.
I am a ms candidate in Japan and a beginner of DMP.
Actually I cannot find so helpful Japanese article about DMP as yours and I'd like to translate &summarize it on my blog to introduce this novel technique.
Can you allow me to do that? If so, surely I cite this URL and note something more as you like.
Thank you anyway.

Reply

travisdewolf says:

June 4, 2015 at 9:29 pm

Hi Michi,
oh neat! Glad you enjoyed it.
If you'd like to translate it and reference this URL please go ahead, that would be great!

Reply

Michi says:

June 8, 2015 at 2:39 am

Thanks!!

yongqiang says:

June 10, 2015 at 9:07 am

Hi. I am learning DMP and I found I don't have background in dynamics. I read something here and there and now I know how to solve system that looks like \dot{x} = Ax, i.e., system without constant term. The system used in DMP, however, has constant term (the attractor) in it, and I don't know how to solve it.

Also when f is involved, the system becomes nonautonomous, I mean time becomes a phase variable.

Would you happen to be able to recommend some reference that I can read to solve the system with constant term?

I am still looking for solutions myself.

Thank you in advance for your time!

Reply

travisdewolf says:

August 13, 2015 at 10:08 am

Hello! Sorry about the delay in replying, this got lost in my inbox. Hmm, perhaps this would be appropriate? http://www.cds.caltech.edu/~murray/courses/cds101/fa02/caltech/astrom-ch4.pdf it looks like they get in to how to capture the dynamics of a system with transfer functions in 4.4, with examples that include gravity (which should be analogous).
Hopefully that helps!

Reply

Rasmus says:

October 9, 2015 at 5:08 am

Hi,

I'm a PhD student in robotics and I just want to thank you for this great introduction to DMP's. Much better for getting a bit of insight into the concept than what else I have come across.

Cheers!

Reply

travisdewolf says:

October 9, 2015 at 6:58 am

Glad you found it useful, thanks for the note! 😀

Reply

Sanket Meenakshi says:

January 25, 2016 at 4:09 pm

solving an assignment. I am a master student for AI and machine learning. This is an amazing composition to understand DMPs. Cannot thank you enough. 🙂

Reply

travisdewolf says:

January 25, 2016 at 4:16 pm

you're very welcome! glad that you found it useful! 😀

Reply

huiwen says:

February 20, 2017 at 11:07 pm

Hi,

Thank you for your job. I have some doubts about your implementation. Specifically, it's about how to calculate the weight of each base functions. Generally, through some calculations and transforms, the weights can be solved analytically. In you code, it seems that you adopt some algorithms which can solve weight one by one. Intuitively, since each point of fd w.r.t. phase variable x is a weighted summation of base functions, I don't think they can be solved individually. Can you give me some explanations?

Thanks

Reply

huiwen says:

February 20, 2017 at 11:19 pm

Sorry, I noticed that you adopt Schaal's algorithm. I read his paper roughly before, maybe I should devote more time into it. Of course, if you have any visions, let me know.

Reply

travisdewolf says:

March 18, 2017 at 1:19 pm

Hi! Good question! So, you're right, in normal circumstances for doing a least-squares optimization or the such we would also take into account the activity of the other basis functions. As I understand it, they kind of skirt around this here, though, by setting up the Gaussians with a variance such that there's mostly only one active at a given time. Schaal's paper will have more details about it, though!

Reply

huiwen says:

March 26, 2017 at 9:26 pm

Hi,
Thank you for your reply. Your interpretation sounds reasonable. While, when I tried to find an original derivatives in Schaal's paper as you attached in your post, I didn't find any explicit formulas about this result, all about LWPR, RFPR, etc. Would you mind giving me a copy of the original derivatives if you have ever seen them before. My email is huiwen3304@gmail.com.
Thx

travisdewolf says:

April 7, 2017 at 11:00 am

hmmm that is odd, that is definitely not the correct link. Not sure how that happened. I'm having trouble finding the paper that I was originally referencing again, though. I'll keep looking when I have time, if you find the derivation please let me know!

iiturrate says:

April 13, 2017 at 10:41 am

Hi. First of all, thank you very much for these posts. They give a very intuitive understanding of DMPs, as opposed to the papers, which can sometimes be hard to get into.

I also just ran into the same question as huiwen, when trying to find where the solution for the individual weights is derived from. I've tried looking through Schaal's papers to find this derivation, but so far, I've been unsuccessful. If you manage to find the paper again, that would be great! If I run into it, I'll post it here.

travisdewolf says:

April 21, 2017 at 1:07 pm

Hello! OK I was just browsing through the papers again and they reference this giant book: http://bit.ly/2dfDYWs

So, apparently the derivation is somewhere in there. Sorry I can't be more specific! I gave it a quick browse but couldn't find it. Definitely looks like it's the right place though!

travisdewolf says:

April 21, 2017 at 2:10 pm

Oh actually, the derivation also appears to be on page 17 here: http://www-clmc.usc.edu/publications/I/ijspeert-NC2013.pdf

iiturrate says:

May 10, 2017 at 9:11 am

Thanks for the reference!

Reply

Navid says:

October 16, 2017 at 2:11 am

Hello, thank you for your clear and valuable tutorial on DMP. I have a question about how you compute BF centers. In method DMPs_discrete::gen_centers, it seems that you generate evenly spaced points (des_c) on x axis as follows:

```
first = np.exp(-self.cs.ax*self.cs.run_time)
last = 1.05 – first
des_c = np.linspace(first, last, self.bfs)
```

Then the time points corresponding to values in des_c are computed and stored in self.c as follows:

```
for n in range(len(des_c)):
# x = exp(-c), solving for c
self.c[n] = -np.log(des_c[n])
```

So if I'm right, the type of values in self.c is time. But as long as I understand self.c should contain values of type x (i.e. canonical system values). Shouldn't we compute the centers as follows:

```
t = np.linspace(0, self.cs.run_time, bfs)
self.c = np.exp( -self.cs.ax * tau * t)
```

Please correct me if I'm wrong. Thank you for your help.
Best regards.

Reply

travisdewolf says:

October 26, 2017 at 10:53 am

Hello! So let's seeeee you are absolutely correct! It seems that I had gotten into a weird head space coding this and convinced myself what I was doing was sensible … it worked out … but it was very convoluted. The method your using makes more sense, and also handles different values of `cs.ax`. So I've updated the code, thank you for this catch!

Reply

Xibao says:

October 26, 2017 at 3:16 am

Hi,
I am learning DMP and the question about get the force term in the transformation system makes me confused. In your poster, you get the weight respectively by solving a summation expression according to time t. I know x is a function of time and Psi is a function of x. However, I cannot understand the summation expression because I think the expression should be double sum according the time t and the number of basis functions. Can you explain me why the expression in your poster can work?

Reply

travisdewolf says:

October 26, 2017 at 8:45 am

Hi Xibao, by separating the basis functions activation from time directly you get a modularization of the problem, where it's very straightforward to affect the temporal dynamics of the system. I'm not sure what to say about why the expressions work, is there some part in particular that is confusing? Can you rephrase your question?

Reply

Xibao says:

October 29, 2017 at 7:22 pm

Thank you, the paper "Locally Weighted Regression for Control" helps understand the method you adopt to get the weights.

travisdewolf says:

October 30, 2017 at 9:21 am

great! thanks for the follow up, i'll point anyone else with the same questions there!

Mac says:

October 31, 2017 at 1:22 am

Hi, thank for the blog, that are extremely helpful. But I have a question about the canonical dynamical system. I mean, what is the reason to introduce an extra system to represent time? Can't we just use the absolute time? For example, if the duration of a movement is 60 seconds, can't we just use phi(t) with t = 0~60 ? This question had bother me for a very long time.Could you please to help me about that? Thank you!

Reply

travisdewolf says:

January 8, 2020 at 5:09 pm

Hello! The idea here is just to abstract one step away from time into a system that's we can explicitly control, it also lets us allow the canonical system rollout to be a function of variables aside from time. I have an example of having the canonical system slow down when the system being controlled falls behind in a blog post later in this series. Does that help?

I just noticed that this comment was from 3 years ago … sorry about the delay!

Reply

Heki says:

October 31, 2018 at 11:38 pm

Hi, I try to make another example and modify your example "moving target and draw number 2". I want to make number 3 or 4, but i dont have any file like 2.npz for input source. Do you have any data for generating number. Many thanks

Reply

travisdewolf says:

January 12, 2020 at 3:06 pm

Hi Heki, to generate these files I used a phone app that let you draw out a trajectory and returned the (x,y) coordinates of the line. You could also use something like https://automeris.io/WebPlotDigitizer/ to draw out the path you want and get the coordinates. Hope this helps!

Reply

Rich says:

January 6, 2020 at 12:46 am

The \tau^2, this is not critical, right? I can't see any derivation linking the \tau in the x_dot equation to \tau^2 in Ijspeert

Reply

travisdewolf says:

January 8, 2020 at 5:05 pm

Hello rich! Thanks for bringing this up. I'm not sure exactly what was going on when I set this up this way, and while it worked, it was a bit crazy. I've reworked both the section and the code, hopefully to make much more sense! I've also added an example to the pydmps repo that shows the temporal scaling in effect.
Cheers!

Reply

Mr.xia says:

February 9, 2020 at 8:51 am

Hi, thank for the blog.I would like to inquire how I can apply dmp to a 3D environment or a 7 DOF robot arm ?

Reply

travisdewolf says:

February 9, 2020 at 9:27 am

Hello! To do multiple degrees of freedom, you can just make multiple instances of the DMP. If you check out https://github.com/studywolf/pydmps there are examples you can follow for 2D environments.

Reply

Yan says:

March 11, 2020 at 4:04 am

Hi, I'm really new in Dynamic Movement Primitive. When you speak about the equation of point attractor system, you said that it's actually a PD control signal. Could you please explain it a little more?

Reply

travisdewolf says:

March 11, 2020 at 10:24 am

Hi Yan, sure! Are you familiar with proportional derivative controllers? That would be the first step, I think! There are a bunch of resources that explain it well online, a quick search pulled up some videos that look decent: https://www.youtube.com/watch?v=0vqWyramGy8&
The trick of it is that a point attractor and a PD controller are both working to drive the system to a target state using the current position and velocity information! Does that help?

Reply

Yan says:

March 15, 2020 at 1:22 am

Ahh, yes, it does help me to understand the equation. Thanks for your response and blog.

Baum says:

May 30, 2020 at 10:54 am

Hello, thank you for this great post. I have one question: In your example, you use the DMP only for the EE-trajectory. But if you want to, for example, teach the robot not only how to move EE but also other joints to reproduce a certain trajectory, how would you do that…?

Reply

travisdewolf says:

May 30, 2020 at 11:06 am

Glad you enjoyed! It's fortunately very easy, you just need to set up a joint controller, and feed the DMP output into that instead. You can see an example of this in part 3: https://studywolf.wordpress.com/2014/03/07/dynamic-movement-primitives-part-3-rhythmic-movements/
Cheers!

Reply

Baum says:

May 30, 2020 at 11:48 am

Thank you for the quick response. So the idea is that you create multiple DMP instances for each joint. But imagine, i) the operator shows the robot how to move each joint to create a straight line with EE. To create a straight line, it would be important when and how to move each joint. So, I try to model this policy with DMPs (there will be multiple solutions but let's assume that the demonstrated option is the only correct one) ii) For each joint we create DMP to reproduce the demonstrated movement iii) Now, if you want to change the goal or start position of your goal-trajectory, there will be no guarantee that the policy generated in i) is kept for the new goal-trajectory, right? Please correct me if I am wrong… Many thanks

travisdewolf says:

May 30, 2020 at 12:16 pm

I'm afraid I don't understand the situation you're describing…are you talking about the trajectory taken in hand space using a joint space DMP?

Baum says:

May 30, 2020 at 12:59 pm

Sorry, its probably because of my English xD. Pls, let me try again: In your example, there are 3 DMPs according to 3 joints. The movements are separately learned. But, actually, to create a human-like leg movement, it's important how these Hip-Knee-Ankle movements are correlated (What I mean with this correlation is this behavior that makes the movements at the end human-like). However, since every DMP is just providing its output regardless of other DMPs, I am wondering whether this human-like movement can be guaranteed if the task environment changes (i.e. if the learned movements have to be changed due to the collision avoidance, each joint movement will be slightly changed, but will the resulted movement still human-like?). Please enlighten me 😉

Dagothar says:

December 20, 2022 at 5:40 am

Hi,
You are first showing how the BFs should be spread if the x is a linearly decaying function and then introduce the canonical system dotx = -alpha_x x. My question is: why is the canonical system neccesary? Why x isn’t, in fact, simply a linearly decaying value? What is the advantage?

Reply

travisdewolf says:

January 5, 2023 at 9:14 am

Hello! A good question! I believe it was made as an exponentially decaying value initially just because the dynamics were easy to implement. In later versions, I’ve been informed they do just use a linearly decaying value for overall simplicity. Related to the value of the canonical system in the first place, the idea is just to have a layer of abstraction away from depending on time explicitly, so that you can more easily do things like control the speed and run in reverse. Does that help?

Reply