A Step Towards Biologically-Plausible Dynamical Movement Primitives

Ananya Kapoor

Fall Quarter 2022

Abstract

This document contains a short summary of my rotation work with James Murray in the Fall 2022 quarter. A brief description of Dynamical Movement Primitives is given as well as a clear outline of our biological extension. Model performance on different test cases is also shown and a brief list of future directions is also given.

1 Problem Motivation

How does the brain control complex nonlinear trajectories that are robust to small changes? For example, how does the brain learn to control a tennis racquet moving to point A during a serve and reuse that learned information to swing the racquet to point B? Models of motor control that explain this behavior are prevalent in robotics and are able to create systems that efficiently (require little training time) produce complex generalizable movements. However, questions of generalizability in neural motor control are not as well understood. My goal during this rotation was to draw inspiration from the robotics literature to build a biologically-inspired model that could explain neural motor control.

I will first start by giving a brief introduction to dynamical movement primitives, which is the robotics literature that I took inspiration from. Both the features/benefits of the model will be described as well as the aspects of this model that are not conducive to biological plausibility. I will then introduce/describe my biologically-inspired model and show its performance on a couple of test cases. I will then discuss some limitations of my model as well as future directions this project could take.

2 Dynamical Movement Primitives

2.1 Model Formulation

Dynamical movement primitives (DMPs) is a concept originally from the robotics literature (Stefan Schaal and Auke Ijspeert 2007, 2013) to describe motor control that is both (1. complex and (2. can generalize to any goal state we set. This model is a response to optimal feedback control (OFC), the conventional model of motor control. While OFC is capable of producing complex motor trajectories, there is no concept of generalization to new goal states. That is, an OFC model that is trained to produce motor trajectory A must be retrained to produce a trajectory B that is identical to trajectory A except for the goal state. Such OFC models are insufficient in describing how the brain

can produce the same trajectory but flexibly reach new goal states. DMPs address this problem by using attractor dynamics, a mathematical model that ensures convergence to a particular state. The functional form of their model is a critically damped harmonic oscillator called the transformation system:

$$\tau y'' = \alpha[\beta(g - y) - y'] + f \tag{1}$$

Where $\alpha = 25$, $\beta = 6.25$, and f is a learnable nonlinear function called the forcing function:

$$f(x) = \frac{\sum_{i=1}^{n} \psi_i(x) w_i}{\sum_{i=1}^{n} \psi_i(x)} x(g - y_0) \tag{2}$$

Here the i indices correspond to each basis function, which in the DMP framework is a Gaussian distribution centered at c_i and has variance of h_i. The w_i weights are learned through locally weighted regression and are specific to each basis function. Here the weights are optimized over all timepoints (refer to Ijspeet et al 2013 Eq 2.14 to see how the weights are derived).

The general steps of the DMP model are as follows:

1. Fit the forcing function to get forcing values at each point in time (technically speaking: each point in $x(t)$).
2. Provide the forcing values as input to the transformation system.
3. Integrate the transformation system (using a method like Euler's Method) to get velocity and then position. These position values will be our training model predictions.
4. Refit the forcing function using the testing goal state. (In hindsight this is also a limitation of the DMP system. We would want our trained model to not relearn the weights when generalizing to a new goal state).
5. Provide the testing forcing values as input to the transformation system, which uses the testing goal state

The result will be a trajectory that generalizes to the testing goal state.

2.2 Benefits of the DMP Framework

The DMP framework is able to have both spatial and temporal invariance. Spatial invariance happens because the transformation system encodes the attractor state (our goal location). Temporal invariance happens because the transformation system adjusts our acceleration values accordingly This is still a bit unclear to me and would need to revisit the temporal invariance.

There are many other benefits to using DMPs, such as obstacle avoidance and coupling terms that allow for online modification of our system. These would be interesting aspects to explore further, especially when applying this and our biologically-inspired DMP model to understand real data.

2.3 Drawbacks of the DMP System

The DMP system is a nice compromise between the optimization perspective (fitting the desired trajectory as closely as possible) and the attractor dynamics perspective (having a system that converges to any arbitrary goal state we desire). It does this by having a nonlinear forcing function

that is learned and then given as input to the transformation system. If we want to think of this in state space modeling terms it would look as follows:

$$x' = Ax + Bu \tag{3}$$

where $u = f$, $B = I$, and A represents the system dynamics.

DMPs work well for robotics because the transformation system is clearly defined. Here we chose a damped harmonic oscillator, a very specific attractor system. When trying to understand neural motor control, we do not have the luxury of picking a very specific transformation system. Our "transformation system" can instead be thought of as a population of primary motor neurons that generate the necessary signals for movement that are sent downstream. Population activity of primary motor neurons can be abstracted as a recurrent neural network, not as a critically damped harmonic oscillator. Moreover, we can think of the forcing function being the result of the population activity of premotor neurons, which prepare and execute movement. This population activity would best be modeled as an RNN, not as a locally weighted regression.

Another drawback is that we are learning the forcing function *explicitly*. This is because of Eq 1, where we can isolate for f and fit a function that best approximates f. We are able to do this because we have a very specific functional form for the transformation system. If we are abstracting our transformation system to be an RNN or a state space model then we do not have access to an explicit forcing function curve to learn.

3 Biologically-Plausible DMP Model Formulation

A biologically-plausible DMP framework (or at least one step towards it) would:

1. Generalize the forcing function and transformation system to be RNNs / state space models
2. Train the forcing function implicitly rather than explicitly

However making these changes does not necessarily guarantee spatial or temporal invariance. We need to take measures to make sure these benefits in DMPs are guaranteed in this biologically-plausible framework. There are three main modifications we make to the original DMP framework:

3.1 Modification # 1: The Forcing Function

The DMP framework used basis functions in a nonlinear regression to fit the forcing function. Here we will use an RNN trained as follows[1]:

1. 5000 epochs
2. A total of int(1/(dt/tau)) timesteps, ranging from 0 to 1
3. Sequence length of 5 timepoints
4. Batch size of int(1/(dt/tau))/sequence length
5. Learning rate of 0.01

[1]These hyperparameters were not optimized. Further work would involve playing around with hyperparameter optimization (grid search, Bayesian search, random search, etc.)

6. 1 hidden layer of 100 neurons

The forcing function accepts $x(t)$ as input, where

$$\tau x'(t) = -\alpha x \tag{4}$$

The functional form of this RNN is given as follows:

$$\begin{cases} u' = \phi(W^u u + W^x x) \\ f(x) \quad = W^f u \end{cases} \tag{5}$$

The forcing function RNN will produce a forcing value as output, which will then be fed into the transformation system.

3.2 Modification # 2: The Transformation System

Instead of using a critically damped harmonic oscillator we will generalize the transformation system to be a (linear) state space model:

$$\begin{cases} r' \quad = W^r r + W^{f*} u \\ y = W^y r \end{cases} \tag{6}$$

Here W^r represents the dynamics of the system, u will be external input to our system, and $W^{f*} = 1$. Moreover we will choose $W^r \sim N(0, 1)$ and $W^y \sim N(0, 1)$.

Unfortunately, this system is not necessarily stable like how the critically damped harmonic oscillator was in the DMP framework. Since stability is not guaranteed, stability to any goal point is also not guaranteed. We need to find a way to design our state space model above so that it converges to any goal state we provide it. To do this, we will use the Linear Quadratic Regulator from Control Theory. The LQR solution will find us the optimal control policy K such that we converge stably to our specified goal location. Incorporating the LQR solution in our state space model, we obtain the following system:

$$\begin{cases} r' \quad = (A - BK)r + BgK_g + xf(x) \\ y = W^y r \end{cases} \tag{7}$$

If $xf(x) = 0$ then we will be left with $r' = (A - BK)r + BgK_g$, which will always converge to the goal location.
The transformation system will be low dimensional [2], with a total of 2 states completely defining the system. Our final model will now be trained as follows:

1. Do a forward pass on the forcing function and create forcing predictions.
2. Feed the forcing predictions into the transformation system to generate position predictions
3. Calculate the Mean Squared Error loss between the position model predictions and the actual position values

[2]But it doesn't have to be, see the limitation section for more info on how the number of states impact model predictions

4. Backpropagate the gradients to the forcing function

Here the forcing function is learned from training and evaluated on the testing trajectory. The BgK_g term in the transformation system, however, is recalculated for the testing trajectory. This is to ensure that our transformation system will converge to our **testing goal state** rather than our training goal state. Therefore the resulting model will create nonlinear trajectories but will also generalizes to any goal state we provide it.

4 Biological DMP Results

We will now demonstrate this DMP model to generate a sine curve. The model will be trained on a sine curve that has -0.97 as the training goal state. The testing goal state will be +5. First let's visualize the trajectory we want to emulate:

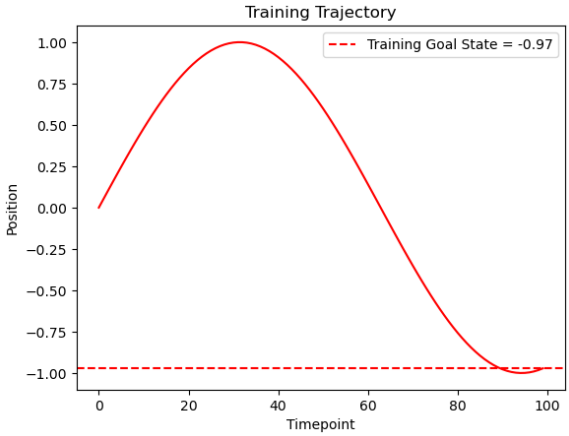

Figure 1: The trajectory that our model will be trained on. This trajectory has a goal state of -0.97, which is reached by the 100th time point.

Our predicted trajectory if the forcing function is 0 for all timepoints is shown in Figure 2.

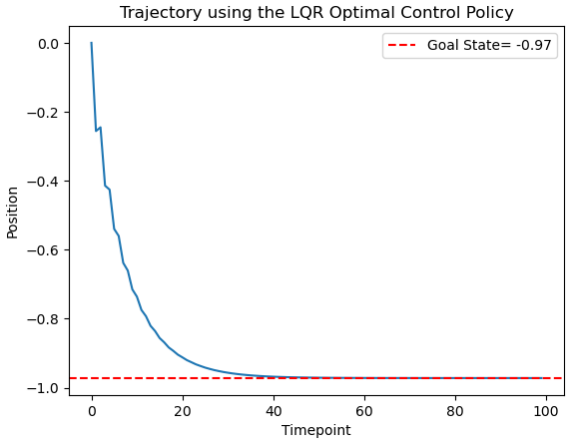

Figure 2: The LQR Solution. Here we see that we simply converge to the goal state.

The LQR Solution in Figure 2 has two hidden states. Therefore the Q matrix of the LQR solution will be an identity matrix of dimension 2×2 and the R matrix is just a single value because we have only one input. Changing the values of Q and R will change how well the LQR solution will converge to the goal state. Increasing the penalty on Q means that we will converge faster to the goal state. Increasing the penalty on R means that we will take less energy to converge to the goal state and will therefore converge slower to the goal state.[3]. Here the R value is set as 20 because we want our trajectory to converge smoothly to the goal state rather than sharply.

We want to see model predictions that look like the trajectory in Fig 1 but also converge to the goal state like in Fig 2. Evaluating our model on the training trajectory, we see the following predictions:

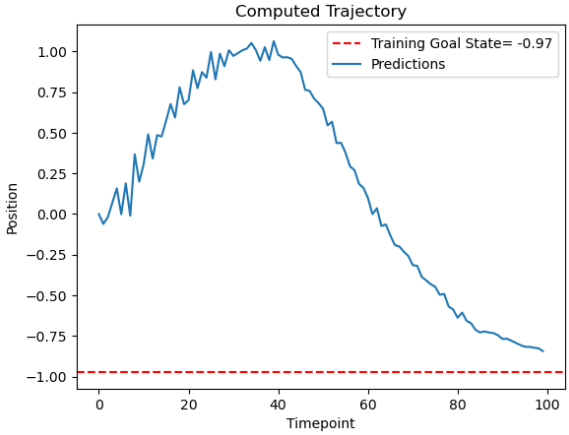

Figure 3: Model predictions on the training trajectory

[3]If we were to generate a new W^r and W^y matrix while holding the values of Q and R constant, we will not always get the same convergence pattern to the goal state.

While the model performance is not perfect, we see that the predictions are closely approximating the training trajectory while also stably converging to the goal state.

Now let's evaluate this model on the testing trajectory. The testing trajectory is the same exact sine curve as the training trajectory, just that the goal state moves from -0.97 to +5. To evaluate this model on the testing trajectory, we need to recalculate BgK_g in Eq 7 with respect to the testing goal state.
The testing predictions are as follows:

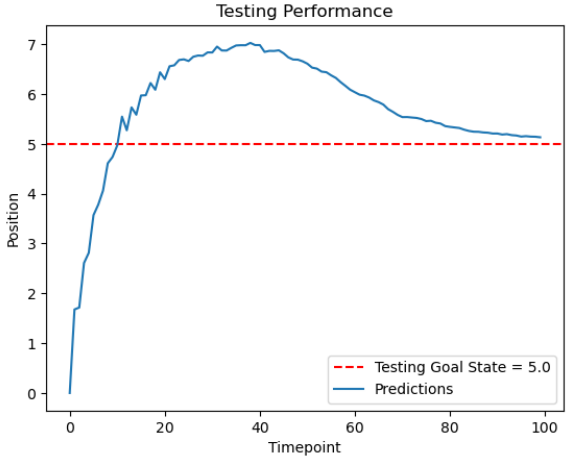

Figure 4: Testing Predictions showing nonlinear behavior but still stably converging to the testing goal state.

We see that our model:

1. Still retains the same nonlinear trajectory shape as in training
2. Converges flexibly to any new goal state we provide it.

Now let's see what happens if we kick the testing predictions off it's course by manually changing a position value. Will the model still converge flexibly back to the goal state? Let's manually set the predicted position value at time 50 to be 10. The predicted value at time 50 for our original testing predictions (Figure 5) was 6.61.

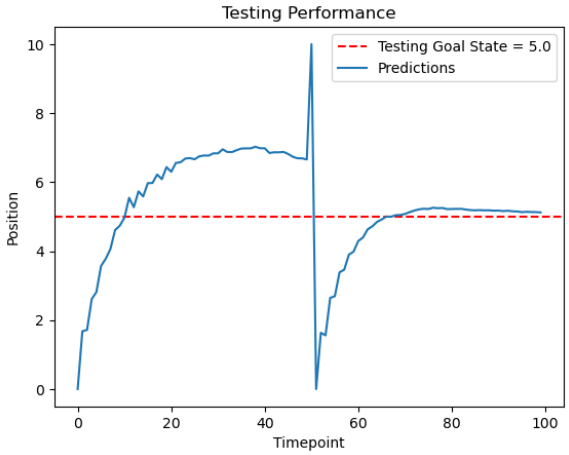

Figure 5: Despite the strong adjustment of the system to being kicked off course at timepoint 50, the system converges stably back to the goal state.

5 Discussion

This model is an extension of Stefan Schaal and Auke Ijspeert's Dynamical Movement Primitives model that is more biologically plausible. While our framework is very closely related to the DMP framework (training the forcing function and utilizing attractor dynamics), we make a couple of key modifications. The first is the generalization of the transformation system to be any arbitrary controllable state space model. This avoids the over reliance on an *a priori* choice for the transformation system such as a damped harmonic oscillator. The second modification we make is to model the forcing function as an RNN rather than a locally weighted regression. These modifications were made to more closely reflect populations of premotor and primary motor neurons, which don't necessarily follow locally weighted regression or a damped harmonic oscillator, respectively.

This biological extension of the DMP model has the same key attributes of the original DMP model, namely the ability to create complex nonlinear trajectories while also stably converging to any goal state we provide the system.

5.1 Limitations and Future Directions

One big limitation of this model is the biologically-implausibility of its transformation system. If the transformation system were to reflect populations of primary motor neurons we would need to have a system with many more states. However, the introduction of an arbitrarily large number of states comes with its own costs. Firstly, the LQR procedure starts to run into numerical issues as the number of hidden states increases. Moreover, even increasing the number of hidden states from 2 to 5 or 10 will cause the convergence pattern to the goal state (as seen in Fig 2) to become much more variable. For example, with a large number of hidden states the convergence pattern could oscillate rapidly before reaching the goal state. For higly complex training/testing trajectories this may be desirable, but then this means that our choice of the number of hidden states would depend on our *a priori* knowledge of how complex the system is (which is not a desirable trait).

Another limitation is the biological implausibility of the Linear Quadratic Regulator. Indeed we would like to find a control policy that is neural network based. This will allow us to abstract correctly the population activity of primary motor neurons while also ensuring convergence to any goal state we desire. A couple of new papers have come out that use neural networks to create control policies (https://doi.org/10.1038/s41467-021-27590-0) that could be useful for our purposes. Additionally, investigating learning rules that primary motor cortex could be using to reach the attractor state would also be a natural next step.

One aspect of the DMP model that would need to be investigated in our biological extension is temporal invariance. We want a model to produce the same dynamics to produce a trajectory that is performed twice as fast or twice as slow. Specifically, we do not want our model to have to retrain in order to change the temporal scale of the movement. We could potentially incorporate this in our model by modifying Eq. 6 by multiplying r' by τ. However, the implications on the LQR stability would need to be investigated more closely because this multiplicative constant would require the LQR solution needing to be recomputed for each temporal factor.

One of the most exciting next steps would be to investigate how our model makes use of motor primitives. For example, our model trained on one training trajectory would be one motor primitive. We could thus make a library of motor primitives by having the model learn various different trajectories. Could it be possible that our model could make use of these motor primitives, combining, modifying, etc., to create even more complex motor trajectories? How would the system know which primitive it is in while doing that complex trajectory? And how does it switch between primitives? This question has roots in continual learning and could potentially form hypotheses that could be tested in neural data.