

CS395T: Robot Learning from Demonstration and Interaction

Assignment 1

In this assignment, you will be implementing Dynamic Movement Primitives (DMPs) to learn, generalize, and improve performance on a task. You should implement the modified version introduced in *Learning and generalization of motor skills by learning from demonstration* by Pastor et al., as described by equations 6-8 (and equation 4, which stays the same in this formulation). Instead of performing a task with a 6-DOF robot hand, you will be implementing a simpler version to control a point moving in 2D space.

Some things to keep in mind when implementing DMPs:

- Phase calculation: When calculating the phase variable $s(t)$, it may be helpful to recall that the differential equation $\dot{y}(t) = ky(t)$ can be solved as $y(t) = Ae^{kt}$, where $A = y(0)$, since $e^{k0} = 1$.
- Linked DMPs: To generate trajectories, you will use 2 different “linked” DMPs (one for each coordinate of the 2D pose) that all share the same phase variable, s (i.e. they are synchronized in time).
- Setting τ : When training the DMP, set τ to the length of the demonstration in seconds. When using the DMP to generate a new trajectory, set τ to the length of the desired trajectory in seconds.
- Setting α : Figure out how to set α such that the phase variable is 99% converged (i.e. $s = 0.01$) at time $t = \tau$ (the length of the trajectory in seconds).
- Setting K and D: You can experiment with different settings of K and D, but make sure that they are set relatively with respect to each other to be critically damped (i.e. $D = 2\sqrt{K}$).

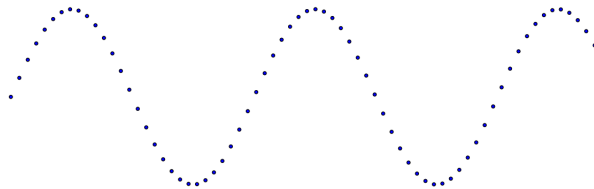
Your main tasks are as follows:

1. Implement two functions that perform the two main tasks of DMPs:
 - (a) DMP learning: This function takes the following as input: a cartesian demonstration trajectory, and the spring / damping constants K and D. It then uses the demonstration to learn the parameters of a DMP, which are stored for later use.
 - (b) DMP planning: This function takes the following as input: stored DMP parameters, a cartesian starting state, a starting velocity, a cartesian goal state, τ (the length of the desired plan in seconds), and dt (the time resolution of the plan). This function should return to the client a planned cartesian trajectory from the start to the goal that is τ seconds long and with waypoints spaced out every dt seconds. You only have to return a trajectory of time-stamped poses—you can ignore the corresponding velocities that the DMP calculates.

2. Function approximation: You will implement two different function approximators:
 - (a) When learning from a single demonstration, there is no need to use a regression-based function approximator for $f(s)$. Instead of using a weighted combination of basis functions, simply store values of $f_{target}(s)$. When looking up values of $f(s)$ during execution, simply perform linear interpolation between the 2 closest stored points (or when s corresponds to a stored value, simply return the stored value of $f_{target}(s)$ at that point.
 - (b) When learning from multiple demonstrations, it can be useful to use linear regression over a set of basis functions to learn the function $f(s)$. Implement Gaussian basis functions (also sometimes known as radial basis functions) as described in the Pastor et al. paper right below equation 3. You will have to choose an appropriate number of basis functions to use, as well as their spacing (in phase space) and width to optimize how well the function is approximated. Note that to turn equation 3 into a standard regression problem, you'll need to multiply both sides of the equation by $\sum_i \psi_i(s)/s$.

Now perform the following data collection and experiments:

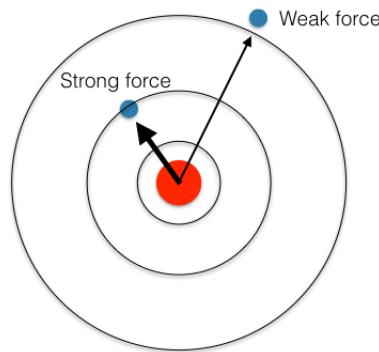
1. Create a “demonstration” by generating (x,y) samples from a sine curve, starting at any point that you want, and ending at any “goal” point that you choose. Sample somewhat densely—collect at least 20 samples per cycle of the sinusoid (i.e. crest to crest) and collect data for at least 2 full cycles of the sinusoid. Remember that each data point has to have a time stamp associated with it as well. Separate each data point by 0.1 seconds, starting a time $t = 0$. Plot your data and include it in your report. The data should look something like the following, but with proper axes, etc.:



2. Use your DMP learning code with the linear interpolation approximator to learn from this data. First, use the same start and goal as you did during the demonstration and generate a new trajectory with your DMP planning code, using the same τ , K , D , and α as you did for learning. Plot this new trajectory on a graph that contains the demonstration trajectory as well, showing that they are similar.
3. Do the same with a significantly different start and goal.
4. Repeat the same experiment, but with a trajectory that is half as fast and twice as fast, respectively. Describe how you did this and show a plot for each, similar to the ones above.
5. Create a second demonstration trajectory that is identical to the first, but with a small amount of zero-mean Gaussian noise added to each point (add noise to both the x and y coordinates). Plot this along with the original demonstration
6. Use your DMP learning code with the Gaussian basis functions to learn from both demonstrations simultaneously by just combining both trajectories into a single list of data points to

learn $f(s)$. Test out different numbers of basis functions, different strategies for spacing them out (i.e. should they be spaced out evenly in phase space for the best performance? Why or why not?), and different widths. List the final parameters you chose and qualitatively describe how you went about choosing them and justify your spacing strategy. Use DMP planning to reproduce a trajectory from the same start and goal as the demo. How accurate is the reproduction? Change the start and goal. Does generalization still work as expected?

7. Add a coupling term to avoid obstacles, similar to the one described by Pastor et al. in Equation 10. However, you will do a simplified 2D version of this—implement a coupling term that exerts an acceleration on the current point's location based on its distance from the center of the obstacle (this acceleration should weaken with distance as a Gaussian that is strongest at the obstacle center) and that exerts this acceleration in a direction along the vector from the obstacle to the current point location. The following image illustrates this:



After adding the coupling term, use DMP planning with the linear interpolation approximator to avoid an obstacle that you place near the path of the nominal trajectory. Plot 2 graphs to understand how this coupling term will change behavior of the DMP. Plot the new trajectory as well as the location of the obstacle and describe how the trajectory changes and why.

Finally, answer the following additional questions:

1. Under what conditions did your DMP generalize properly to new starts and goals? Did it execute the motion in the same "style" that it was demonstrated? Why or why not?
2. Here, we simply have generated a static trajectory with the DMP. How would you go about implementing a DMP that calculates commands on-the-fly, so that the robot could respond to perturbations during execution? In this setting, how would the magnitude of constants K and D affect the behavior of the system under a perturbation?
3. If a learned DMP was used to solve a problem with a well-defined reward or cost function, policy search might be one way to iteratively improve performance. For example, you might randomly perturb DMP parameters, accepting the change if performance is better, and rejecting the change if it is worse. Under this approach, which parameters of the DMP would you randomly perturb, and why?

Be sure to include all relevant code as an appendix to your writeup.

Frequently asked questions:

Q: What does it mean to integrate the equations?

The only equation that you have to analytically integrate is the canonical (phase) system so that for any time t , you can evaluate $s(t)$. I give a hint in the assignment about how to solve the differential equation. When you are planning, all you have to do is divide both sides of the main equations by τ and plug everything in, which solves for what you want \dot{v} and \dot{x} . However, to get a new position x from this you have to integrate over the time-step length dt . There are several ways you can do this, but one way is to do a rough approximation of the integral by updating your new v by $dt \cdot \dot{v}$ and x by $dt \cdot \dot{x}$, but you are welcome to implement more accurate ways. For most applications, this approximation is fine.

Q: How do I plan?

When doing planning x starts at x_0 (the starting position of the trajectory) and v starts at 0. Then you calculate a new x and v from \dot{x} and \dot{v} , as I described. Now you have a new x and v that you plug in and repeat the process to get each new point in the trajectory.

Q: Why isn't the last point of the trajectory exactly at the goal?

Remember that DMPs are only guaranteed to converge in the limit as the phase s approaches zero. Even though for demonstrations, we can interpret τ as the length of the demo in seconds, it takes on a slightly different meaning during planning. It is the number of seconds until the phase variable is a certain percentage converged. However, if you choose, you can still continue to generate samples of the plans for times after $t = \tau$ until the DMP reaches a location that is within some ϵ distance of the goal.

Q: Why isn't it following the path that I expect?

Once you've made sure there are no bugs in your code, try cranking up K and D (while keeping them critically damped). K might need to be large (≥ 1000) before you start to get results that track well.

Comments:

Finally, something to be careful of during learning. The demos you collect provide the positions x . From those you can calculate \dot{x} and \ddot{x} . These are NOT the same as v and \dot{v} in equation 8. Why? Because of time scaling. If you look at equation 7, you'll see that $v = \tau \cdot \dot{x}$, and therefore $\dot{v} = \tau \cdot \ddot{x}$. Essentially any time you take a derivative or integrate with respect to time, the time constants are going to affect your answers.

During planning, you can think of v as an internal variable that represents what the velocity would be if it wasn't being corrected by time scaling, whereas \dot{x} is the scaled version. During learning, you can think of the measured x values (and the computed \dot{x} and \ddot{x}) as being scaled in time based on how long the trajectory is, so you essentially have to unscale them to get v and \dot{v} . Another way of thinking about it is that v and \dot{v} are velocity and acceleration in *phase space*, not in normal time, so they have to be unscaled / scaled when learning / planning.