Robot Learning DMP Assignment

Junhyeok Ahn

September 20, 2017

1 Trajectory generation

I generated the trajectory using below equation:

$$
\begin{aligned}
x &= \frac{0.6}{\tau}t \\
y &= Asin(\frac{2\pi}{0.3}x),
\end{aligned} \tag{1}
$$

where $A = 0.15,\ \tau = 20$. I generated time stamps with 0.1s interval and got Cartesian $\mathbf{x} = (x,\ y)$ trajectory (Fig. 1). Corresponding part in the code is 'gen_demo' function in the attached code. The demonstrated trajectory dataset is composed of

$$
\begin{aligned}
\tau &= [(t_1, \mathbf{x}_1),\ \cdots,\ (t_n, \mathbf{x}_2)], \\
\mathbf{x}_i &= [x_i,\ y_i]^\top.
\end{aligned} \tag{2}
$$

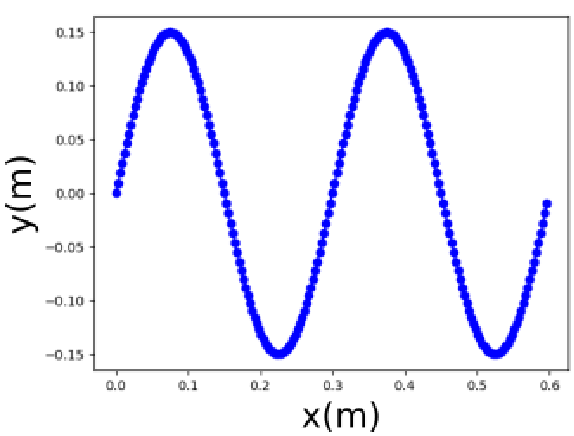

Figure 1: **Sine wave trajectory with identical time interval:** The trajectory starts from $[0,\ 0]^\top$ and ends at $[0.6,\ 0]^\top$ for 20s. Data are sampled with 0.1s interval.

2 DMP planning with linear interpolation

I calculated $\dot{\mathbf{x}}$, $\ddot{\mathbf{x}}$, $\mathbf{v}$, $\dot{\mathbf{v}}$ based on the consecutive $\mathbf{x}$'s in the trajectory τ and phase variable s. Then I calculated $\mathbf{f}_{\text{target}}(s)$ and appended it with corresponding phase variable as below.

$$\begin{aligned} \mathcal{D} &= [(s_1, \mathbf{f}_{\text{target,1}}), \ \cdots, \ (s_n, \mathbf{f}_{\text{target,n}})] \\ \mathbf{f}_{\text{target,i}} &= [f_{\text{target,i,x}}, \ f_{\text{target,i,y}}]^\top. \end{aligned} \tag{3}$$

In this calculation, I used $\alpha = log(1-0.99)$ to make $s = 0.01$ at the end. Also, $K_p = 1000$, $K_d = 600$ are used for both x and y. In this problem, I just copied all $\mathbf{f}_{target}(s)$ to $\mathbf{f}(s) = [f_x(s), \ f_y(s)]$. Once I got $\mathbf{f}(s)$, I planned the trajectory with the same start and goal position with time interval $dt = 0.001$. I queried the value $\mathbf{f}(s)$ at every corresponding phase variables to forward integrate from starting position. As the time stamps for new trajectory are not identical to the time stamps in the demonstration trajectory τ, I linearly interpolate to compute $\mathbf{f}(s)$. The trajectory planned by DMP is shows in Fig 2. In attached code, in DMP class, 'getPVA' function is calculating velocity and acceleration and 'learn' function is learning DMP parameters, such as $\mathbf{f}_{\text{target}}$, weights for Gausian basis functions and so on. In 'pred' function is used whenever I queried $\mathbf{f}$ at specific s. Function 'plan' is the part actually planning using forward integration.

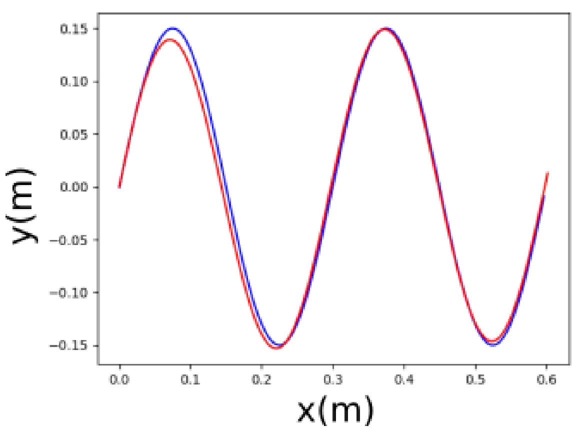

Figure 2: **DMP learning and planning with linear interpolation approximator:** Blue trajectory is demonstrated trajectory in Problem 1 and red trajectory is planned from DMP using linear interpolation

3 Experiments with significantly diffrent goals

Based on the $\mathbf{f}(s)$ learned from $\mathbf{f}_{\text{target}}(s)$ (in this case I just copy this and linearly interpolated), I applied this motion primitive to the different goals. In the attached code, different goals could be easily specified by changing arguments. Figure 3 shows the planned trajectory from DMP with different goals. (a), (b) and (c) show quite convergence compared to (d) because nonlinear function behaves in the reverse direction in the case of the goal located opposite direction.

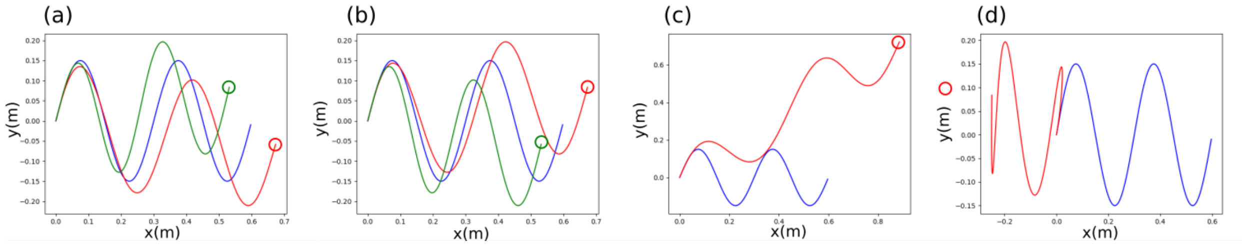

Figure 3: **Experiments with different goals:** (a) red goal $= [0.7, \ -0.1]^\top$, green goal $= [0.5, \ 0.1]^\top$ (b) red goal $= [0.7, \ 0.1]^\top$, green goal $= [0.5, \ -0.1]^\top$ (c) red goal $= [1.0, \ 0.9]^\top$ (d) red goal $= [-0.3, \ 0.1]^\top$

4 Experiments with various speed

In this problem, I experimented with diffrent temporal durations. In Problem 1, I used $\tau = 20$ for both demonstration and DMP planner. To plan for half as fast and twice as fast trajectory from the same demonstration, I maintained $\tau = 20$ when I computed and approximated $\mathbf{f}_{\text{target}}$ and $\mathbf{f}_s$ but use $\tau = 10$ and 20 for the different time durations in planning with DMP. Figure 4 shows time versus x and y and red trajectory represents planned with different time duration.

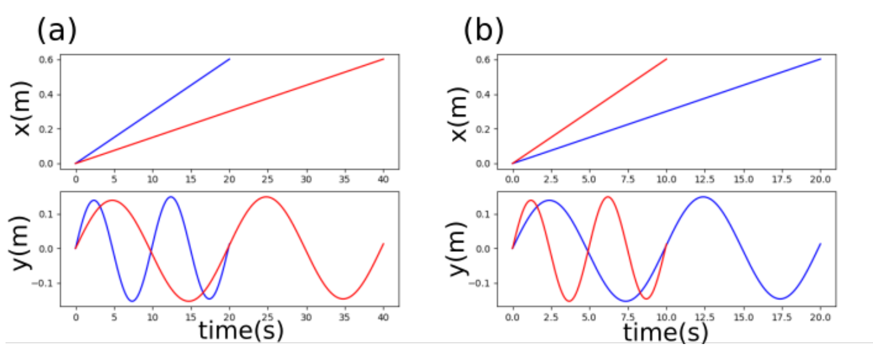

Figure 4: **Trajectory planned by DMP with different time duration:** (a) shows the red trajectory planned for $\tau = 40$, which is twice slower than blue. (b) shows the red trajectory planned for $\tau = 20$, which is twice faster than blue.

5 Generate trajectory with Gaussian noise

Similar to Problem 1, I generated one trajectory τ_1, and add Gaussian noise $\mathcal{N}(0, \ 1)/c$ to x and y, where c is a scaling factor. Figure 5 shows the original trajectory and differentiated trajectory with gaussian noise. I used $c = 250$ as the scaling factor.

6 DMP planning with Radial Basis Function

Based on the trajectories in Figure 5, I calculated $\mathbf{f}_{\text{target}}$ in similar way to Problem 2 for both trajectory and generated dataset $\mathcal{D} = \{(s_1, \ \mathbf{f}_{\text{target,1}}), \ \cdots, \ (s_n, \ \mathbf{f}_{\text{target,n}})\}$. I multiplied $\Sigma_i \frac{\psi_i(s)}{s}$ to

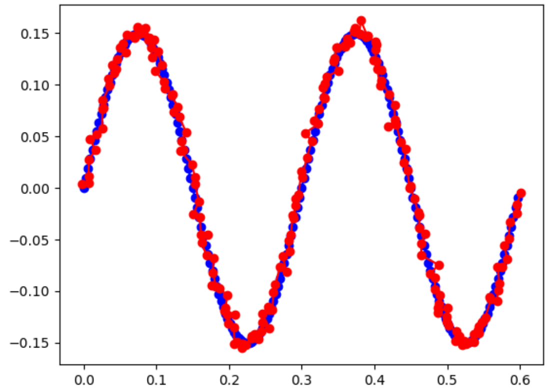

Figure 5: **Trajectory with Gaussian noise:** Blue and red trajectories are both demonstrated trajectory. However, red trajectory includes Gaussian noises

transform problem into the standard regression problem as

$$\begin{bmatrix} \mathbf{f}_{\text{target},1}(s)\Sigma_i \frac{\psi_i(s_1)}{s_1} \\ \vdots \\ \mathbf{f}_{\text{target,n}}(s)\Sigma_i \frac{\psi_i(s_n)}{s_n} \end{bmatrix} = \begin{bmatrix} \psi_{1,1}(s_1) & \cdots & \psi_{1,m}(s_1) \\ \vdots & \ddots & \vdots \\ \psi_{n,1}(s_n) & \cdots & \psi_{n,m}(s_n) \end{bmatrix} \begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix}, \tag{4}$$

where n and m is the number of datapoint and basis functions respectively. Note that $\mathbf{f}_{\text{target}} \in \mathbb{R}^2$ as defined in Eq. (3), so I solved Eq. (4) twice for x and y. Then, using left pseudo inverse, weights could be computed as,

$$\begin{bmatrix} w_1 \\ \vdots \\ w_n \end{bmatrix} = \left(\Psi^\top \Psi\right)^{-1} \Psi^\top \begin{bmatrix} \mathbf{f}_{\text{target},1}(s)\Sigma_i \frac{\psi_i(s_1)}{s_1} \\ \vdots \\ \mathbf{f}_{\text{target,n}}(s)\Sigma_i \frac{\psi_i(s_n)}{s_n} \end{bmatrix}. \tag{5}$$

In here I used 40 Gaussian functions for each x and y to approximate $\mathbf{f}(s)$. Once the weights are calculated, I could forward integrate to the trajectory. The trajectory from the DMP is illustrated in Figure 6.

Now, I will show several trials I played with number of basis functions and placing the center of radial basis functions. Figure 7 shows unevenly distributed center case and evenly distributed center case respectively. It is able to see putting phase variable s more around $0 \sim 0.5$ shows better result. This is because the demonstrated data is more densly distributed where s is close to 0. Therefore, the final parameters I've use is 40 basis functions (put 20 between $0 \sim 0.5$ and put the others 20 between $0 \sim 1$) with 1500 width.

I also described how DMP with radial basis functions generalize the different goals. It generalized trajectories with different goals in Fig. 8.

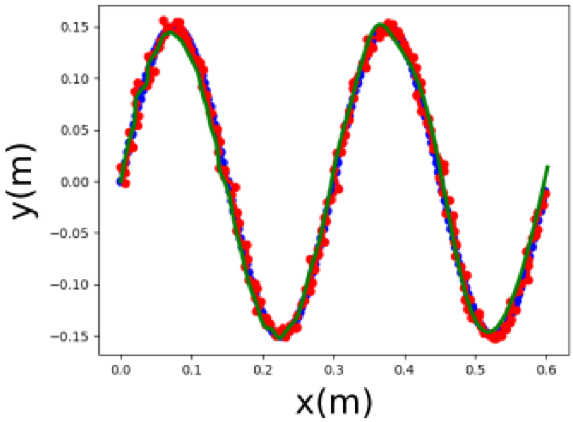

Figure 6: **DMP learning and planning with radial basis function approximator:** Green trajectory is planned from DMP using radial basis functions. 40 basis functions are used to approximate $\mathbf{f}(s)$ for each x and y.

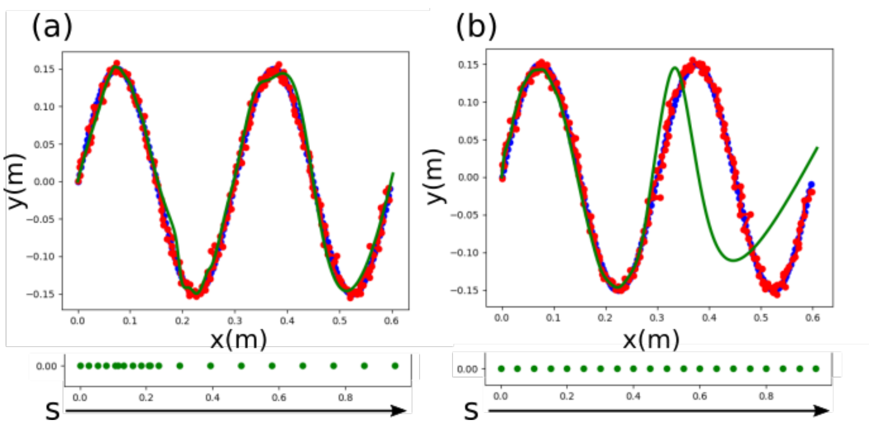

Figure 7: **Comparing two different planned trajectory with different radial basis function center:** (a) shows 20 unevenly distributed centers. (b) shows 20 evenly distributed centers. green dots represents the center of radial basis functions. Note that to show the different clearly I only used 20 basis functions even though I used 40 basis functions for the others.

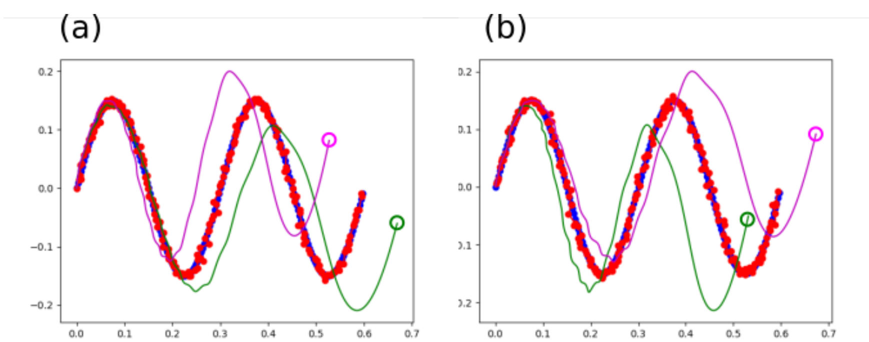

Figure 8: **Experiments with different goals:** (a) pink goal $= [0.5,\ 0.1]^\top$, green goal $= [0.7,\ -0.1]^\top$(b) pink goal $= [0.7,\ 0.1]^\top$, green goal $= [0.5,\ -0.1]^\top$

7 Obstacle Avoidance

I placed an obstacle at $\mathbf{p}_{obs} = [0.21,\ 0.0]^\top$ which is exerting an acceleration, which is weakened with distance as a Gaussian that is strongest at the obstacle center

$$r = \sqrt{(x - p_{obs,x})^2 + (y - p_{obs,y})^2}, \tag{6}$$

$$\theta = arctan2(y - p_{obs,y},\ x - p_{obs,x}), \tag{7}$$

$$\mathrm{acc}_{\mathrm{obs}} = 50e^{-100(r-0)^2}[cos\theta,\ sin\theta]^\top, \tag{8}$$

where $\mathbf{x}$ and $\mathbf{p}_{\mathrm{obs}}$ are position of robot (end effector in Cartesian) and obstacle respectively and $\mathrm{acc}_{\mathrm{obs}}$ is added on the trajectory. I also used scaler factor 50. Figure 9 shows the red trajectory trying to avoid the obstacle.

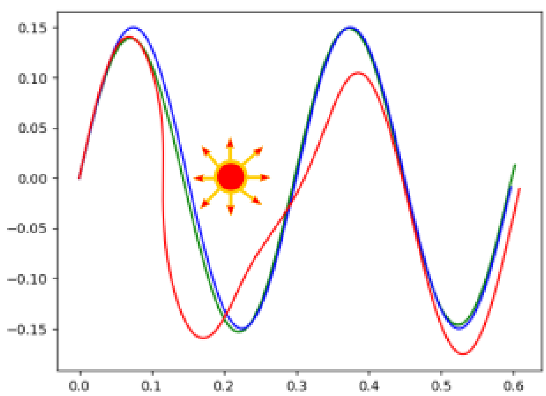

Figure 9: **Obstacle avoidance:** blue trajectory is demonstrated and green trajectory is planned trajectory without obstacle. Obstacle is placed at $[0.21,\ -0.13]^\top$ and the resulting trajectory is illustrated as red.

8 Additional Questions

8.1 Under what condition DMP works well or not

As demonstrated in Figure 3, DMP could generalize trajectory quite well in the case that the new goal is located in the same side of the goal demonstrated. For example, if I look at the (a) and (b), the trajectory almost converges to the goal with maintaining the primitive of the demonstrated trajectory. Even though there is some slight error at the goal, (c) shows the similar trends in the trajectory. However, if I look at (d), x axis goal lays in the opposite side but, y axis goal is not located so far. The trajectory planned from DMP shows a big error in x compared to y. Therefore, I could conclude the goal location affects the performance of convergence as well as the similarity on the shape (or primitive).

Also, DMP is not converged if the obstacle is located near the goal. It is quite obvious because it is generating an acceleration. Figure 10 shows the case I put obstacle at the goal position $[0.6,\ 0]^\top$.

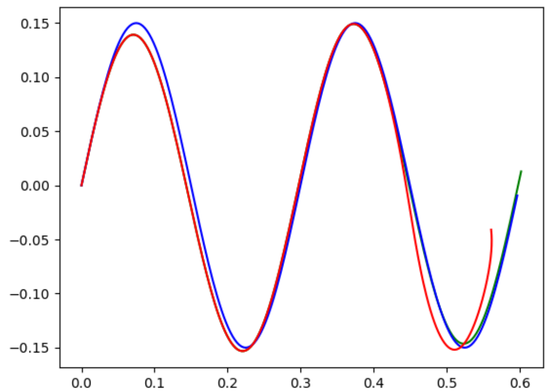

Figure 10: **Obstacle at goal position:** The obstacle is located at goal position where $[0.6,\ 0]^\top$. The green trajectory shows the planned one without obstacle and the red shows the trajectory considering obstacle

8.2 How to respond to perturbation

The easiest way to implement DMP robust to the perturbation is to use large K_p and K_d gain and makes phase variable s converge to zero quickly. This will reduce a tendency to follow up the trajectory demonstrated and improve convergence to the goal. Also, I could also update $f(s)$ whenever the trajectory is executed.

8.3 Which parameters could be perturbed to improve performance

If I assume the cost/reward function is well described, now the problem is more than minimizing the error between $f(s)$ and f_{target}. Because demonstration should not be illustrating the trajectory in terms of well described cost. So the weights as well as the center and width of radial basis function are all subject to be perturbed.