Advanced Robotics: Probabilistic Movement Primitives (github Repo)

March 21, 2022

This notebook includes a class called `ProMP` which includes all the information required to define a ProMP as well as important methods which allow the manipulation of the MPs to show the potential of the probabilistic approach.

For stroke-based movements, the mean of a ProMP is given by a weighted sum of N Gaussians. These Gaussian basis functions have the following form:

$$b_i(t) := \exp\left(-\frac{(t-c_i)^2}{2h}\right),\ i = 1, \ldots, N$$

These functions are commonly normalized to improve regression

$$\phi_i(t) := \frac{b_i(t)}{\sum_{j=1}^{N} b_j(t)} \tag{1}$$

The function $z(t)$ allows temporal modulation, so that our basis functions are given by:

$$\phi_i(z) = \phi_i(z(t))$$

We work in discrete time, so we define a time step $\mathrm{d}t$. By default, we work with a time lapse of 1s, therefore, if T steps are needed, $\mathrm{d}t = 1/T$. We then introduce $\mathbf{\Phi}_t := (\phi_1(t), \ldots, \phi_N(t)) \in \mathbb{R}^{N \times 1}$, which includes the evaluation of all basis functions in a time step.

```
def BasisFuncGauss(N, h, f, dt):
    tf = 1/f;
    T = int(round(tf/dt+1))
    Phi = np.zeros((T,N))
    t = z*dt
        q = np.zeros((1, N))
        for k in range(1,N+1):
            c = (k-1)/(N-1)
            q[0,k-1] = np.exp(-(f*t - c)*(f*t - c)/(2*h))
        Phi[z,:N] = q[0, :N]
# -- INSERT YOUR CODE HERE --
    return Phi
```

The `BasisFuncGauss` function creates and evaluates N basis Gaussian functions with a bandwidth h. Input `f` refers to the linear modulation factor, so that the modulation is given by $z(t) := ft$. Then, the Gaussian functions $\phi_i(t)$ are evaluated as $\phi_i(ft)$. `dt` represents the time step. The functions have modes at c_i, evenly spread along $[0, 1/f]$. The output of this function is a $T \times N$array for which every row corresponds to the vector $\mathbf{\Phi}_t$ for every time step.

In the code list above, write instructions that will normalize the basis functions in F (see Eq. (1)).

In a ProMP, at time step t, a joint variable q is modeled as:

$$q_t = \mathbf{\Phi}_t^{\mathrm{T}} \mathbf{w} + \epsilon_q$$

where ϵ_y adds zero-mean Gaussian observation noise with variance Σ_y. It follows that the probability of observing q_t is represented by:

$$p(q_t|\mathbf{w}) = \mathcal{N}\left(q_t \,\middle|\, \mathbf{\Phi}_t^{\mathrm{T}} \mathbf{w}, \mathbf{\Sigma}_q\right)$$

Since Σ_y is the same for every time step, the values q_t are taken from independent and identical distributions, i.i.d. Hence, the probability of observing a trajectory $\tau := \{q_1, \ldots, q_T\}$ is given by:

$$p(q_t|\mathbf{w}) := \prod_{t=1}^{T} p(q_t|\mathbf{w})$$

However, since parameters $\mathbf{w}$ are to be learnt from data, we also assume such parameters are taken from a distribution $\mathbf{w} \sim p(\mathbf{w}|\theta) = \mathcal{N}(\mathbf{w}|\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}})$. We therefore would like to have a predictive distribution of q_t which does not depend on $\mathbf{w}$, but on $\theta := (\mu_{\mathbf{w}}, \mathbf{\Sigma}_{\mathbf{w}})$. This is done buy marginalizing $\mathbf{w}$ out in the distribution as follows:

$$\begin{aligned} p(q_t|\theta) &= \int \mathcal{N}\left(q_t \,\middle|\, \mathbf{\Phi}_t^{\mathrm{T}} \mathbf{w},\, \Sigma_q\right) \mathcal{N}\left(\mathbf{w} \,\middle|\, \mu_{\mathbf{w}},\, \Sigma_{\mathbf{w}}\right) \mathrm{d}\mathbf{w} \\ &= \mathcal{N}\left(q_t \,\middle|\, \mathbf{\Phi}_t^{\mathrm{T}} \mathbf{w},\, \Sigma_q + \mathbf{\Phi}_t^{\mathrm{T}} \mathbf{\Sigma}_{\mathbf{w}} \mathbf{\Phi}_t\right) \end{aligned} \tag{2}$$

```
class ProMP:
    def __init__(self, N, h, dt, covQ, Wm, covW):
        self.N = N
        self.h = h
        self.dt = dt
        self.covQ = covQ
        self.Wm = Wm
        self.covW = covW
        self.Phi = BasisFuncGauss(N,h,1,dt)
        self.T,_ = self.Phi.shape
        self.Qm = np.matmul(self.Phi,self.Wm)
        self.cov = np.zeros((self.T,1))
        for i in range(0,self.T):
            self.cov[i,0] = ## write your code here ##
```

In the `ProMP` class, a `ProMP` object is initialized by the number of basis functions, `N` (N), the bandwidth in the basis functions, `h` (h), the time step, `dt` ($\mathrm{d}t$), the covariance of the original distributions on q, `covQ` (Σ_q), the mean of the weights, `Wm` ($\mu_{\mathbf{w}}$), and the covariance of the weights, `covW` ($\mathbf{\Sigma_w}$).

When initialized, the class also defines other convenience variables. `Phi` is the $T \times N$ matrix containing the basis functions evaluated for every time step and with a default modulation factor equal to 1; `Qm` ($\mu_{q,t}$) is the mean values of q for every time step; and `cov` is the covariance of the marginal distribution on q for every time step according to Eq. (2).

One of the big advantages of the ProMPs approach is the inclusion of new via-points. Say we wish to add a new via-point $q^*_{t^*}$ at time t^* which is to be observed with an uncertainty of Σ^*_q. We simply have to obtain a posterior distribution over $\mathbf{w}$ by affecting our prior distribution with the new information given by $q^*_{t^*}$ and Σ^*_q. We thus apply Bayes theorem as follows:

$$p(\mathbf{w}|q^*_{t^*}, \Sigma^*_q) \propto \mathcal{N}\left(q^*_{t^*} \,\middle|\, \mathbf{\Phi}^{\mathrm{T}}_{t^*}\mathbf{w},\, \Sigma^*_q\right) p(\mathbf{w})$$

The result of applying the Bayes theorem leads to the following mean and variance of the posterior distribution:

$$\begin{aligned} \mu^{[\mathrm{new}]}_{\mathbf{w}} &= \mu_{\mathbf{w}} + \mathbf{\Sigma_w}\mathbf{\Phi}_{t^*}\left(\Sigma^*_q + \mathbf{\Phi}^{\mathrm{T}}_{t^*}\mathbf{\Sigma_w}\mathbf{\Phi}_{t^*}\right)^{-1}\left(q^*_{t^*} - \mathbf{\Phi}^{\mathrm{T}}_{t^*}\mu_{\mathbf{w}}\right) \\ \mathbf{\Sigma}^{[\mathrm{new}]}_{\mathbf{w}} &= \mathbf{\Sigma_w} - \mathbf{\Sigma_w}\mathbf{\Phi}_{t^*}\left(\Sigma^*_q + \mathbf{\Phi}^{\mathrm{T}}_{t^*}\mathbf{\Sigma_w}\mathbf{\Phi}_{t^*}\right)^{-1}\mathbf{\Phi}^{\mathrm{T}}_{t^*}\mathbf{\Sigma_w} \end{aligned} \tag{3}$$

```
def condition(self, tstar, Qstar, covQstar):
    Phit = np.transpose(self.Phi[tstar-1:tstar,:])
    self.Wm = # -- INSERT YOUR CODE HERE --
    self.covW = # -- INSERT YOUR CODE HERE --
    for i in range(0,self.T):
        self.cov[i,0] = self.covQ \
+ np.matmul(np.array([self.Phi[i,:]]), \
                    np.matmul(covW,np.transpose(np.array([self.Phi[i,:]]))))
```

The `condition` method conditions a ProMP to the new observation `Qstar` ($q^*_{t^*}$) at time `tstar` (t^*) with precision of observation `covQstar` (Σ^*_q).

Complete the above code list to compute `Wm` and `covW` follwing Eq. (3). Use variable `Phit` (Φ_{t^*}) which is defined inside the function.

Temporal modulation is carried out by means of the previously defined phase function $z(t)$.

```
def modulate(self, factor):
        self.Phi = BasisFuncGauss(self.N,self.h,factor,self.dt)
        self.Phi = self.Phi/np.transpose(mat.repmat(np.sum(self.Phi, axis=1), \
                                            self.N,1));
        self.T,_ = self.Phi.shape
        self.Qm = np.matmul(self.Phi,self.Wm)
        self.cov = np.zeros((self.T,1))
        for i in range(0,self.T):
            self.cov[i,0] = self.covQ + np.matmul(np.array([self.Phi[i,:]]), \
np.matmul(self.covW, np.transpose(np.array([self.Phi[i,:]]))))
```

Member function `modulate` allows temporal modulation of a ProMP. Only linear time modulation is supported by this class. Modulation is given by the `factor` (f) input variable which is originally set to 1 when a ProMP is initialized. The phase function is then given by $z(t) = ft$. Note that `dt` is constant throughout the code, hence, temporal modulation modifies T.

```
    def printMP(self, name):
        t = np.arange(0, self.T*self.dt, self.dt)
        plt.plot(t,self.Qm)
        upper = # -- INSERT YOUR CODE HERE --
        lower = # -- INSERT YOUR CODE HERE --
        plt.fill_between(t, upper[:,0], lower[:,0], color = 'k', alpha = 0.1)
        plt.title(name)
        plt.show()
```

The `printMP` method inside the `ProMP` class plots the mean `Qm` (μ_q) against time. It also plots two standard deviations above and below μ_q in order to show the marginal distribution at every time step. The `name` value refers to the title of the plot.

To complete the code above, compute the $T \times 1$ arrays `upper` and `lower` which contain two standard deviations above and below the mean μ_q, respectively, for every time step. Remember that we store the covariance at every time step in the member variable `cov`.

We would like to be able to blend different MPs into a single movement. For example, given two ProMPs, each having an important via-point, we wish to blend them so that our new ProMP crosses both via-points. We can do this by activation and deactivation of the ProMPs. We define the activation functions $\alpha^{[1]}(t)$ and $\alpha^{[2]}(t)$, with values $\alpha_t^{[1]}, \alpha_t^{[2]} \in [0, 1]$ at each time step $t = 1, \ldots, T$. When the value of an activation function reaches 0, the corresponding ProMP is fully deactivated. Oppositely, when the value reaches 1, the ProMP is fully activated. With these tools, the result of blending two ProMPs with predictive distributions $p^{[i]}(q_i) = \mathcal{N}(q_t \,|\, \mu_{q,t}^{[i]}, \Sigma_t^{[i]})$, $i = 1, 2$ yields to a new

ProMP described by $p^*(q_t) = \mathcal{N}(q_t \,|\, \mu^*_{q,t}, \Sigma^*_t)$, where:

$$\begin{aligned} \Sigma^*_t &= \left(\frac{\alpha_t^{[1]}}{\Sigma_t^{[1]}} + \frac{\alpha_t^{[2]}}{\Sigma_t^{[2]}} \right)^{-1} \\ \mu^*_{q,t} &= \Sigma^*_t \left(\frac{\alpha_t^{[1]}}{\Sigma_t^{[1]}} \mu^{[1]}_{q,t} + \frac{\alpha_t^{[2]}}{\Sigma_t^{[2]}} \mu^{[2]}_{q,t} \right) \end{aligned} \tag{4}$$

```
def blend(MP1, MP2, alpha1, alpha2):
    cov12 = # -- INSERT YOUR CODE HERE --
    Qm12 = # -- INSERT YOUR CODE HERE --
    M12 = ProMP(MP1.N, MP1.h, MP1.dt, MP1.covQ, np.zeros((MP1.N,1)), \
            np.zeros((MP1.N,MP1.N)))
    M12.cov = cov12
    M12.Qm = Qm12
    return M12
```

The `blend` function blends two `ProMPs` objects, `MP1` and `MP2`, according to the activation functions `alpha1` ($\alpha_t^{[1]}$) and `alpha2` ($\alpha_t^{[2]}$). Vectors `alpha1` and `alpha2` contain the values of the activation functions for every time step. It is assumed that both ProMPs have the same value for `T`, hence, `alpha1` and `alpha2` must be arrays of `T` elements.

Using Eq. (4), complete the code above by writing the expressions for `cov12` and `Qm12`, which correspond to the covariance Σ^*_t and mean $\mu^*_{q,t}$ of the resulting ProMP at every time step. In order to be compatible with the `ProMP` class, `cov12` and `Qm12` must be arrays with shapes `(T,1)`.

Now that the functions and the `ProMP` class have been defined, we start with the main loop of the `ProMP.py` code. This program uses the samples of weights $\mathbf{w}$ obtained in 5 observations. So we assume that the learning process has been completed. With this data, a ProMP for a single joint variable q is created. We will carry out the following experiments:

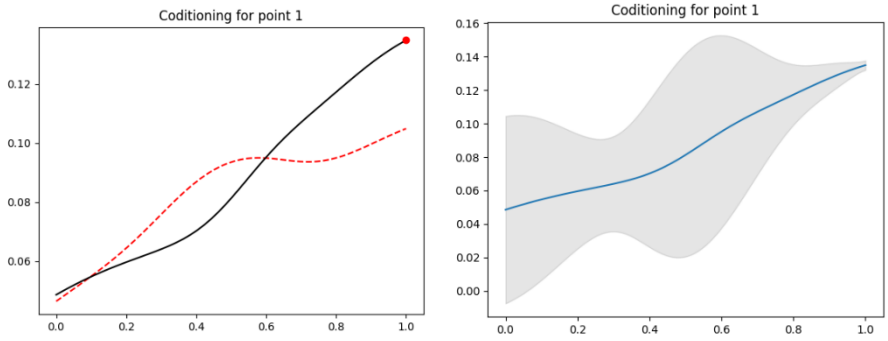
- Condition the ProMP to a first via-point
- Condition the resulting ProMP from the previous experiment to a second via-point
- Obtain two ProMPs after conditioning the original MP to each via-point, separately, and blend the two results
- Time-modulate the original ProMP to different phases

```
Wsamples = np.array([[0.0141,0.0130,0.0038,0.0029,0.0143],
                         [0.0044,0.2025,0.0178,0.0703,0.0143],
                         [0.0388,0.1042,0.0531,0.0854,0.1479],
                         [0.0025,0.0321,0.0235,0.0495,0.0086],
                         [0.0810,0.0178,0.1500,0.0310,0.0843],
                         [0.0658,0.1258,0.0488,0.1650,0.1398],
                         [0.1059,0.0821,0.0116,0.2260,0.0531],
                         [0.0032,0.0952,0.0305,0.2220,0.0025],
                         [0.2031,0.1665,0.1430,0.0842,0.0656],
                         [0.0491,0.1543,0.1232,0.1505,0.0049],
                         [0.1914,0.0525,0.0783,0.0009,0.0292],
                         [0.0584,0.1035,0.0830,0.0305,0.1452],
                         [0.0157,0.1713,0.2550,0.0695,0.0051],
                         [0.2106,0.0630,0.0942,0.0086,0.1512],
                         [0.0959,0.2093,0.1388,0.0566,0.0819]])
    Wmean = np.transpose([np.mean(Wsamples, axis=1)])
    Wcov = np.cov(Wsamples)
    N,_ = Wsamples.shape
    T = 100
    dt = 1/(T-1)
    MP1 = ProMP(N,0.02, dt, 1e-6, Wmean, Wcov)
    MP2 = ProMP(N,0.02, dt, 1e-6, Wmean, Wcov)
    MP3 = ProMP(N,0.02, dt, 1e-6, Wmean, Wcov)
```

Array `Wsamples` includes the values of the `N = 15` weights after 5 observations. $\mu_{\mathbf{w}}$ is computed from these observations and stored in `Wmean`. Similarly, $\mathbf{\Sigma}_{\mathbf{w}}$ is saved in `Wcov`. Using these values, three identical objects of the class `ProMP` are defined.

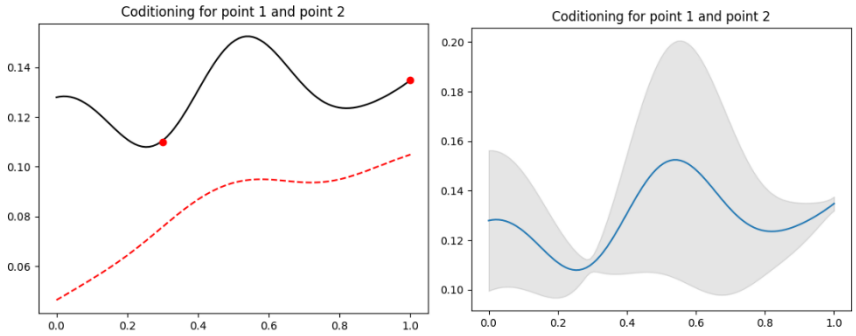
```
    tstar1 = 100
    Qstar1 = MP2.Qm[100-1]+ 0.03
    covQstar1 = 1e-6
    MP2.condition(tstar1,Qstar1,covQstar1)
    t = np.arange(0, 1+dt, dt)

    plt.figure()
    plt.plot(t,MP1.Qm, 'r--', t, MP2.Qm, 'k')
    plt.plot(tstar1/(MP1.T), Qstar1,'ro')
    plt.title('Coditioning for point 1')
    plt.show()

    MP2.printMP('Coditioning for point 1')
```

`MP2` is conditioned to point `Qstar1`, at time step `T`, with an uncertainty of the observation equal to `covQstar1`. The desired via-point, the original mean and the new conditioned mean are then plotted.




```
tstar2 = 30
Qstar2 = 0.11
covQstar2 = 1e-6
MP2.condition(tstar2,Qstar2,covQstar2)

plt.figure()
plt.plot(t,MP1.Qm, 'r--', t, MP2.Qm, 'k')
plt.plot(tstar1/(MP1.T), Qstar1,'ro')
plt.plot(tstar2/(MP1.T), Qstar2,'ro')
plt.title('Coditioning for point 1 and point 2')
plt.show()
MP2.printMP('Coditioning for point 1 and point 2')
```

The resulting MP is now conditioned to a second via-point, `Qstar2`, at time step `tstar2`. The resulting ProMP must cross both points.



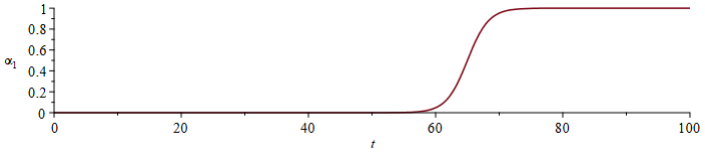
```
MP1 = ProMP(Wrows,0.02,dt,1e-6,Wmean,Wcov)
MP2 = ProMP(Wrows,0.02,dt,1e-6,Wmean,Wcov)

MP1.condition(tstar1,Qstar1,covQstar1)
MP2.condition(tstar2,Qstar2,covQstar2)

MP1.printMP('MP1: Coditioning for point 1')
MP2.printMP('MP2: Coditioning for point 2')

alpha1 = # -- INSERT YOUR CODE HERE --
alpha2 = -alpha1+1

MP12 = blend(MP1,MP2,alpha1,alpha2)
MP12.printMP('blending of MP1 and MP2')
```

MPs `MP1` and `MP2` are conditioned to points 1 and 2, respectively. Then both are blended. The result of the blending must have both points as via-points.

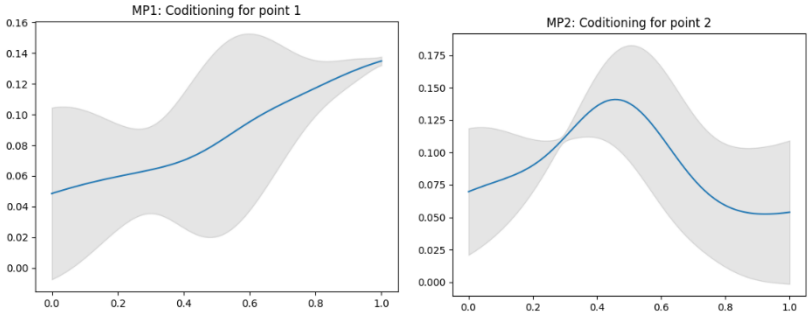
To complete the code, come up with an expression for the $T-$element array `alpha1` which contains the values $\alpha_t^{[1]}$ of the activation function for MP `MP1`. Since `tstar1 = 100` and `tstar2 = 30`, $\alpha_t^{[1]}$ must be equal (or close) to 1 at $t = 100$ but must be equal (or close) to 0 at $t = 30$, and there must be a smooth transition between these values. $\alpha_t^{[2]}$ is simply computed by `alpha2 = -alpha1+1` in the following line of the code. For this, find suitable values for A, B and C in the following function:

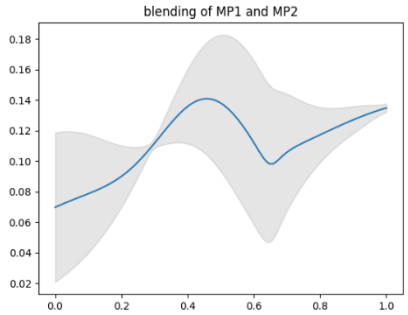
$$\alpha^{[1]}(t) := A\tanh\left(B\left(\frac{t-C}{T}\right)\right) + A$$

The figure below shows an example of such a function using $B = 30$, but you can play with this value and see how it affects the result.



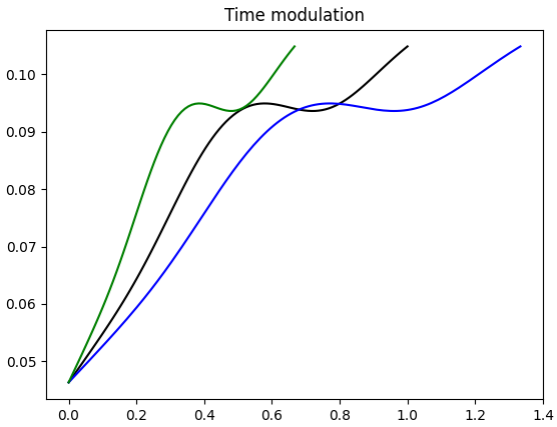
The program then plots the ProMPs with the following results:





```
MP1 = ProMP(Wrows,0.02,dt,1e-6,Wmean,Wcov)
MP2 = ProMP(Wrows,0.02,dt,1e-6,Wmean,Wcov)
MP3 = ProMP(Wrows,0.02,dt,1e-6,Wmean,Wcov)
MP2.modulate(0.75)
MP3.modulate(1.5)

t1 = np.arange(0, MP1.T*dt, dt)
t2 = np.arange(0, MP2.T*dt, dt)
t3 = np.arange(0, MP3.T*dt, dt)

plt.figure()
plt.plot(t1,MP1.Qm, 'k')
plt.plot(t2,MP2.Qm, 'b')
plt.plot(t3,MP3.Qm, 'g')
plt.title('Time modulation')
plt.show()

MP1.printMP('modulation factor = 1')
MP2.printMP('modulation factor = 0.75')
MP3.printMP('modulation factor = 1.5')
```

Given three initially identical MPs, `MP1`, `MP2` and `MP3`, `MP2` is modulated with a factor of 0.75, i.e. $z(t) = 0.75t$, and `MP3` with a factor 1.5, i.e. $z(t) = 1.5t$. `MP1` is left unmodified.


Time modulation
0.10
0.09
0.08
0.07
0.06
0.05
0.0
0.2
0.4
0.6
0.8
1.0
1.2
1.4