Obstacle Avoidance using Dynamic Motion Primitives

by

Varad Vaidya

An internship report submitted for
the completion of the internship

at
Indian Institute of Technology, Delhi

Under the guidance of
Prof. S.K. Saha and
PhD. Deepak Raina

Contents

Chapter 1

Dynamic Motion Primitives

The term *Motion Primitives* has its roots in neurobiology and motor control, where researchers explain the execution of complex motion of biological systems and their ability to adapt to different types of motion effortlessly. The Dynamic Motion Primitives formulation is one among the many that use the popular approach in *Robot Learning* called Learning from Demonstrations, which uses the demonstrations given by a human in some form (teleoperation, kinesthetic control etc. . .) and learn from it to scale to different tasks for robot to perform. In this respect, DMPs attempt to answer the question:

How artificial systems can execute complex movements in a versatile and creative manner?[1]

Thus, Dynamic Motion Primitives (DMPs) can be seen as rigorous mathematical formulation of motion primitives as stable nonlinear dyanmical systems.

1.1 Dynamics of DMPs

At the heart of DMP lies a spring mass damper system.

$$\tau \ddot{y} = \alpha \left(-\beta \left(y - y_g \right) - \dot{y} \right) \tag{1.1}$$

where, α and β are the constants of the spring mass damper system, y_g is the goal state of the system. To avoid overshooting or slow convergence of the system to the goal state, the system is set to be critically damped. Thus in the DMP notation, $\beta = \alpha/4$. This determine the value of β for a given value of α.

The infulence of α on the DMP system can be seen in Fig.

This representation has properties such as convergence to goal state and robutnesss to external perburtation but can only represent very simple movements. To solve this we add a time dependaant forcing term to the spring mass damper system. The spring damper system along with the forcing term is called as *transformation system*.

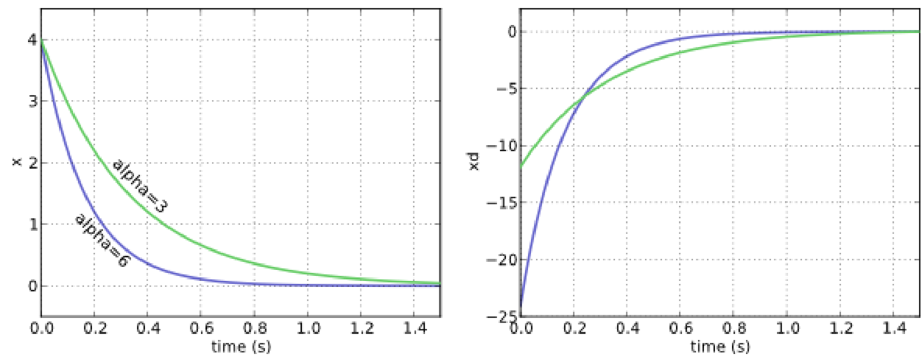


Figure 1.1: DMP system with different values of α

After adding forcing term to our sping damp system, the we can no longer gurantee the convergence property and the tranformation system is no longer time independant. To solve the later we let f be a function of a phase variable x representing the phase of the movement. Ijspeert2002 used a first order dynamical system to model the phase variable, coing the term *canonical system*.

Thus the complete Dynamic Motion Primitives system is given by:

$$\tau \dot{z} = \alpha_z(\beta_z (z - z_g) - \dot{z}) + f(x) \tag{1.2a}$$
$$\tau \dot{y} = z \tag{1.2b}$$
$$\tau \dot{x} = \alpha_x x \tag{1.2c}$$

where, z is the state of the system, x is the canonical variable, $f(x)$ is the non linear forcing term. And as described above, parameters α_z define the charateristics of the DMP system, and with $\tau > 0$, $\beta_z = \alpha_z/4$ and $\alpha_x > 0$ the convergence to goal state is guaranteed.

The term $f(x)$ is defined as the linear combination of N nonlinear Radial Basis Functions (RBF) which are also known as Gaussian basis functions. This enables the DMP to follow any arbitrary smooth trajectory from initial point y_0 to final position y_g.

$$f(x) = \frac{\sum_{i=1}^{N} w_i \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} x \tag{1.3}$$
$$\Psi_i(x) = e^{-h_i(x-c_i)^2} \tag{1.4}$$

where h_i are the centers of the RBF, and c_i are the centers of the RBF for $i = 1, 2, \ldots, N$. The weights w_i are found oout from the measured data so that the

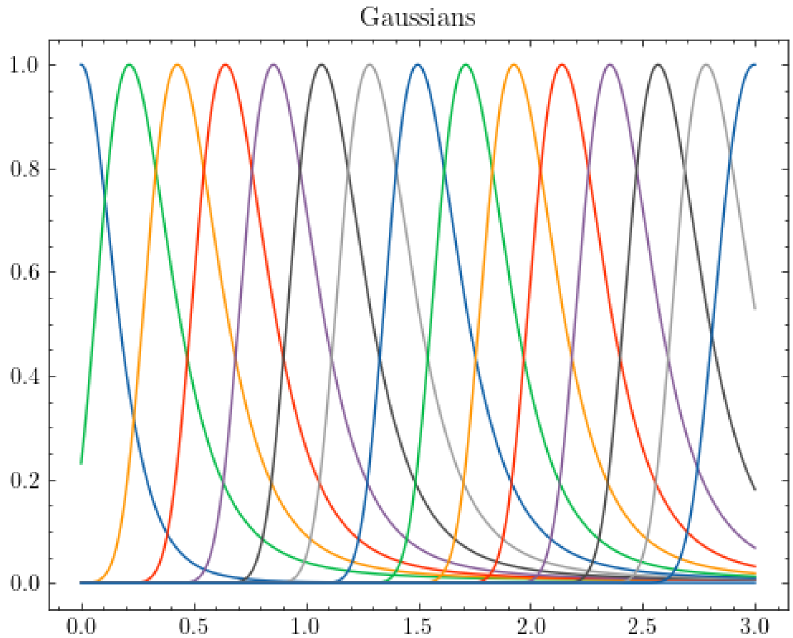


Figure 1.2: Gaussian Activations in Time

desired trajectory is achieved. Fig. 1.2 shows the Gaussian activations in time. We can define the venters and widths as

$$c_i = e^{-\alpha_x \dfrac{i-1}{N-1}} \tag{1.5}$$

$$h_i = \frac{1}{(c_{i+1} - c_i)^2} \tag{1.6}$$

where $h_N = h_{N-1}$. The selection of the number of weights and hence the RBF's is decided based on the accuracy required. But it is observed that a certain minimum number of RBF's is required to encode the trajectory any meaningful manner.

1.2 Learning from demonstration

For a discrete motion, given a demostrated trajectory $y_d(t_j)$, where $t_j = 1, \ldots, \mathcal{T}$ is the number of time steps, and its time derivatives $\dot{y}(t_j)$ and $\ddot{y}(t_j)$, we can invert eqn (1.2a) to approximate the desired forcing term.

$$f_d(t_j) = \tau^2 \ddot{y}(t_j) - \alpha_z(\beta_z(g - y_d(t_j)) - \tau \dot{(y)}_d(t_j)) \tag{1.7}$$

Arranging the desired forcing term $f_d(t)$ and the unknoen weights w_i into column vector such that,

$$\boldsymbol{\mathcal{F}} = [f_d(t_1), f_d(t_2), \ldots, f_d(t_T)]^T \text{ and } \boldsymbol{w} = [w_1, w_2, \ldots, w_N]^T$$

we obtain a linear system,

$$\boldsymbol{\Phi w} = \boldsymbol{\mathcal{F}} \tag{1.8}$$

where,

$$\boldsymbol{\Phi} = \begin{bmatrix} \frac{\psi_1(x_1)}{\sum_{i=1}^{N} \psi_i(x_1)} x_1 & \cdots & \frac{\psi_N(x_1)}{\sum_{i=1}^{N} \psi_i(x_1)} x_1 \\ \vdots & \ddots & \vdots \\ \frac{\psi_1(x_{\mathcal{T}})}{\sum_{i=1}^{N} \psi_i(x_{\mathcal{T}})} x_{\mathcal{T}} & \cdots & \frac{\psi_N(x_{\mathcal{T}})}{\sum_{i=1}^{N} \psi_i(x_{\mathcal{T}})} x_{\mathcal{T}} \end{bmatrix} \tag{1.9}$$

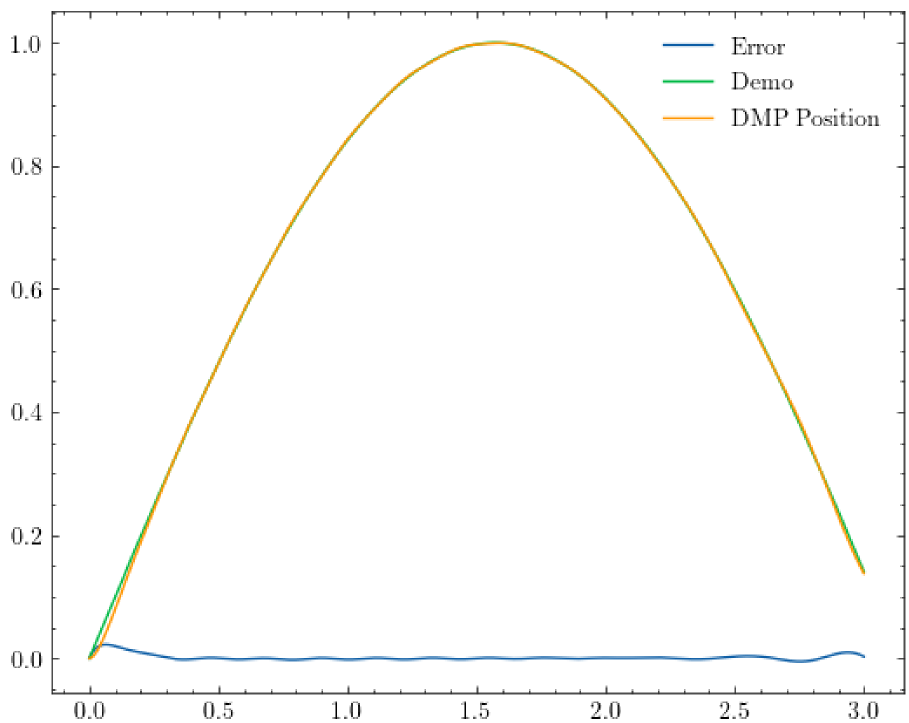


Figure 1.3: A classical DMP is used to reproduce the trajectory of $x(t) = sin(t)$ for a total time of 3 seconds.

This system can be solved by using various techniques such as the Least squares method, Locally Weighted Regression(LWR) etc. LWR is a popular approach to update the weights w_i learned from the previous demonstration, using the error between the learned trajectory a and desired trajectory by using a forgetting factor λ. In this project work, Least squares methods was used as it approximated test results accuratly.

1.3 Temporal and Spatial Scalability of DMP

The most important feature of DMP is that once trained on a certain training trajectory, it can replicate the trajectory even when the goal and initial conditions are changed and follow it without requiring any additional training to train the weights. This is called as *spatial scalability* of DMP.

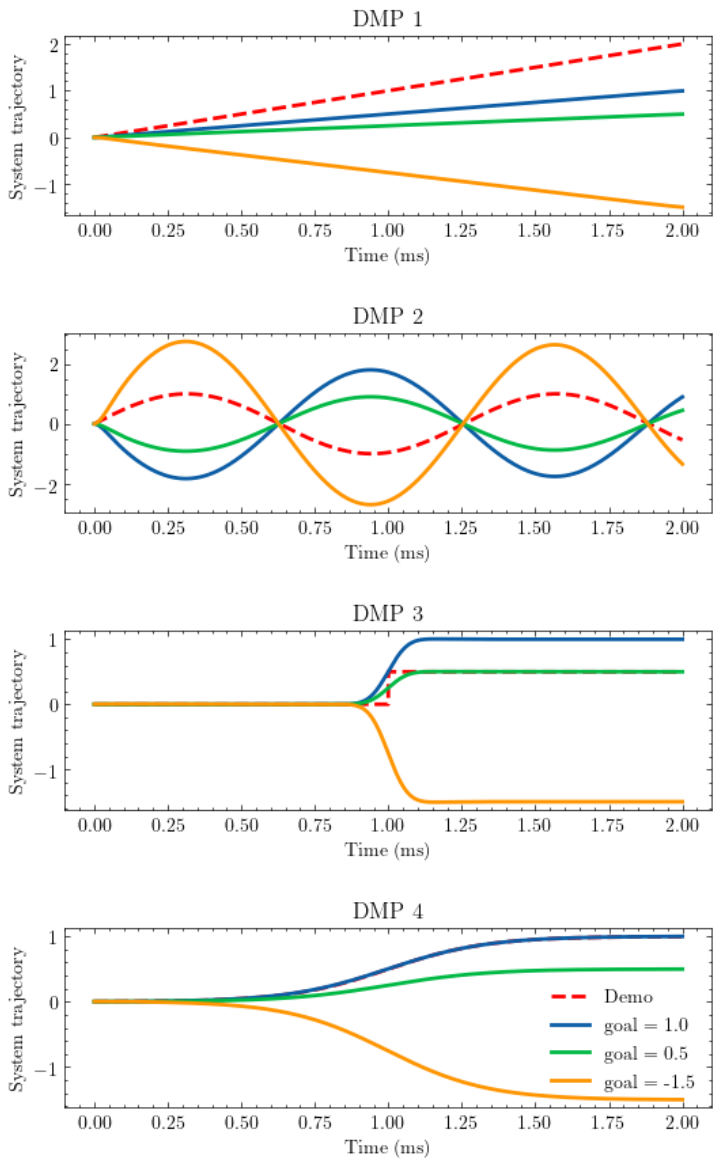


Figure 1.4: The time constant τ is the parameter that controls the speed of the movement.

DMPs can also scale down or up a trajectory in the temporal direction by changing the time constant τ, resulting in a shorter or longer trajectory time. This is called *temporal scalability* of DMP. The time constant τ is the parameter that controls the speed of the movement.

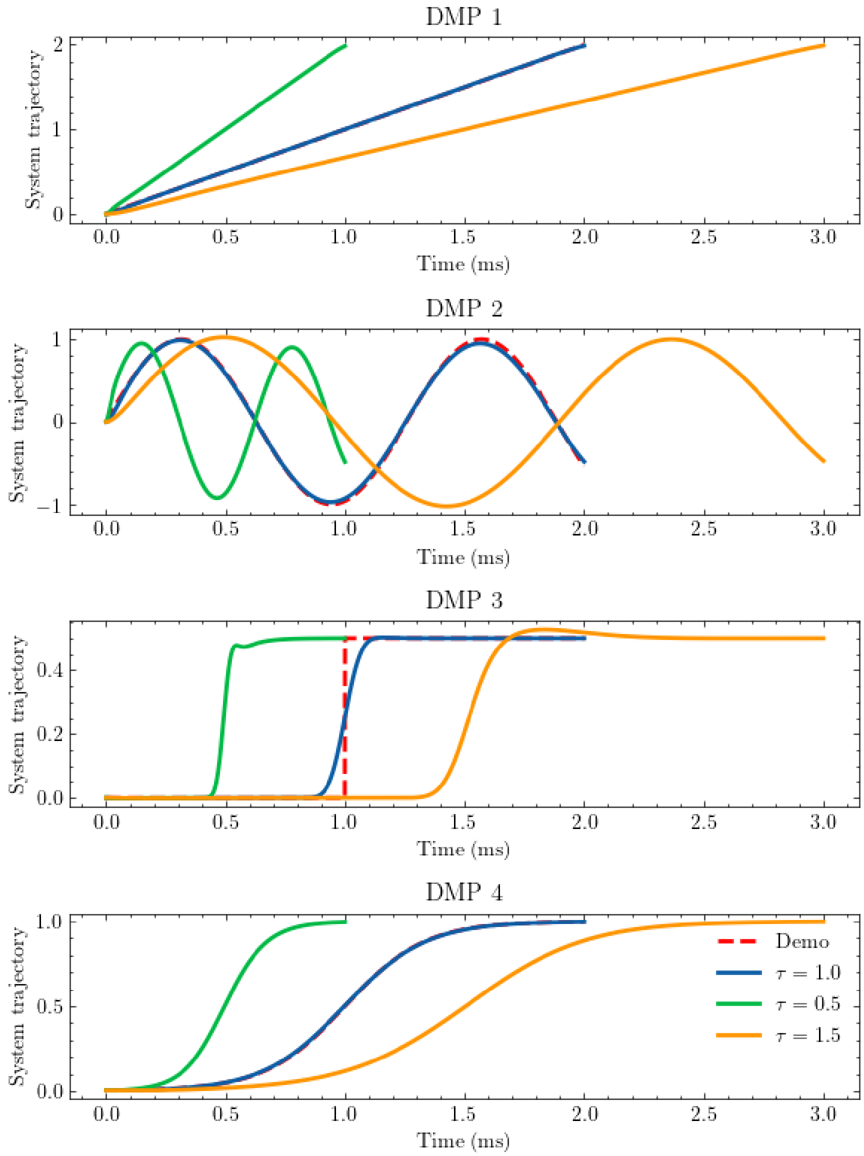


Figure 1.5: The time constant τ is the parameter that controls the speed of the movement.

This property when mixed with joint DMP can provide a method for online adaptation of desired trajectory in robotic manipulators and other robotic systems.

This framework of DMP is called Discrete DMP. One of its main disadvantages is that it cannot enocde the orientation of the trajectory into the system. As such a new set of equations are transofrmaiton systems are created to handle this issues.

Chapter 2

Cartesian Dynamic Motion Primitives

To modify the formulation to handle multi-dimensional system or D dimensional trajectories, for instance the 7 joints of an arm or the position of its end effector we simply use D transformation systems. A key principle in DMPs is to use the same phase system for all the transformation system, to ensure that the transformation systems are synchronized in time. Thus the states in the tranformation sytem will be vectors and can be written as:

$$\tau \dot{\boldsymbol{z}} = \alpha_z(\beta_z (\boldsymbol{z} - \boldsymbol{z_g}) - \dot{\boldsymbol{z}}) + \boldsymbol{f(x)} \tag{2.1a}$$

$$\tau \dot{\boldsymbol{y}} = \boldsymbol{z} \tag{2.1b}$$

$$\tau \dot{x} = \alpha_x x \tag{2.1c}$$

where the states are vectors of lengths D which are all synchronized in time by the phase equation(2.1c)

2.1 Position Only DMP

2.1.1 2D DMP

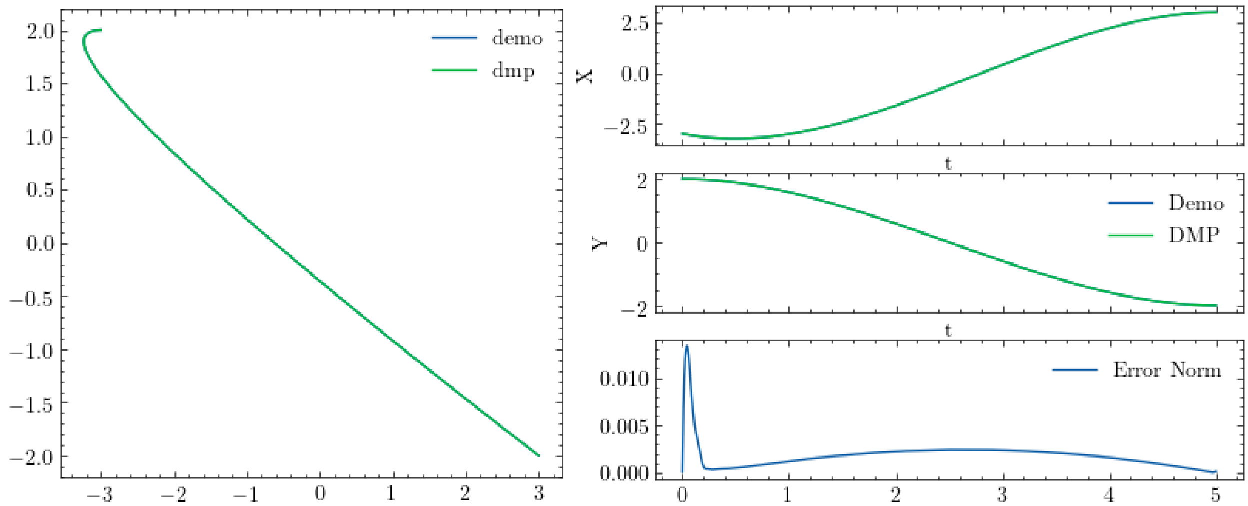
To produce a 2 dimensional DMP in x-y plane, we crated a minimum acceleration trajectory between points [-3,2] and [3,-2] with initial velocity as [-1,0] which has to be travelled in 5 secs. The DMP parameters are: $\alpha_z = 50, N_{bfs} = 100$ with the canonical system parameters as: $\alpha_x = 1, \tau = 1$

Minimun acceleration trajectory $x(t)$ is a cubic polynomial in time

$$x(t) = a_1 t^3 + a_2 t^2 + a_3 t + a_4 \tag{2.2}$$

where $t \leq T$ and the equation is constrained by the boundary conditions $x(0) = x_0, x(T) = x_g, x(\dot{0}) = \dot{x_0}$ and $x(\dot{T}) = \dot{x_g}$ where x_0 and x_g is the initial and final position of the trajectory.

The resultant DMP trajectory is plotted as shown in Fig. 2.1



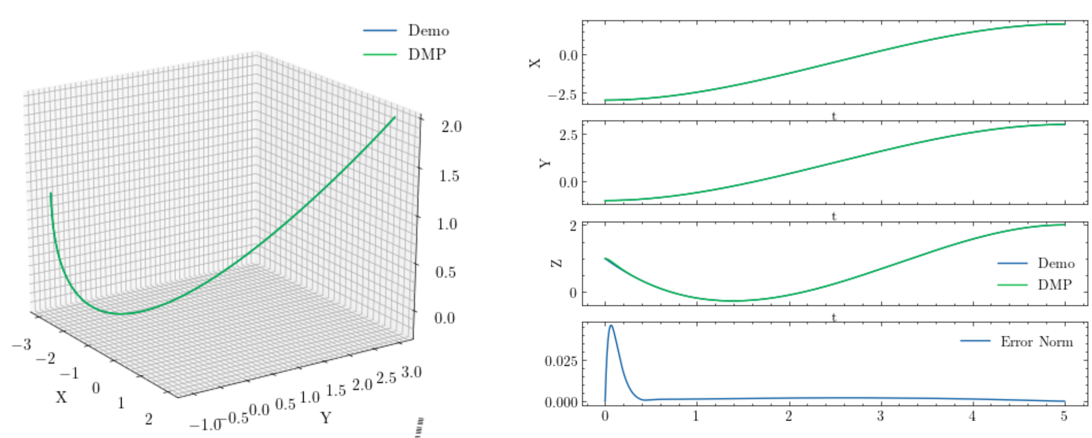
(a) 2D DMP trajectory (b) Deviation from the desired trajectory

Figure 2.1: 2D DMP trajectory

2.1.2 3D DMP

Similar to 3D DMP, we can create a 3D DMP in cartesian space by creating a minimum acceleration trajectory for a total time of 5 secs between points [-3,-1,1] and [2,3,2] with initial velocity as [0,0,-2].The DMP parameters are: $\alpha_z = 30, N = 100$ with the canonical system parameters as: $\alpha_x = 3, \tau = 1$

The resultant DMP trajectory is plotted as shown in Fig. 2.2



(a) 3D DMP trajectory (b) Deviation from the desired trajectory

Figure 2.2: 3D DMP trajectory

This 3D cartesian DMP can be simulated on a robotic manipulator which follows a certain trajectory. In this simulation experiment, and all the simulated experiments following, we use Kuka LBR iiwa 7 axis industrial robot, simulated in PyBullet. A minimum acceleration trajectory is created between points [0.4,0.4,0.9] and [-0.2,-0.2,1.2] with initial velocity as [0,0.2,-0.1] which has to be travelled in 10 secs. The DMP parameters are: $\alpha_z = 20, N = 100$ with the canonical system parameters as: $\alpha_x = 0.5, \tau = 1$

The resultant DMP trajectory simulated is shown in Fig. 2.6

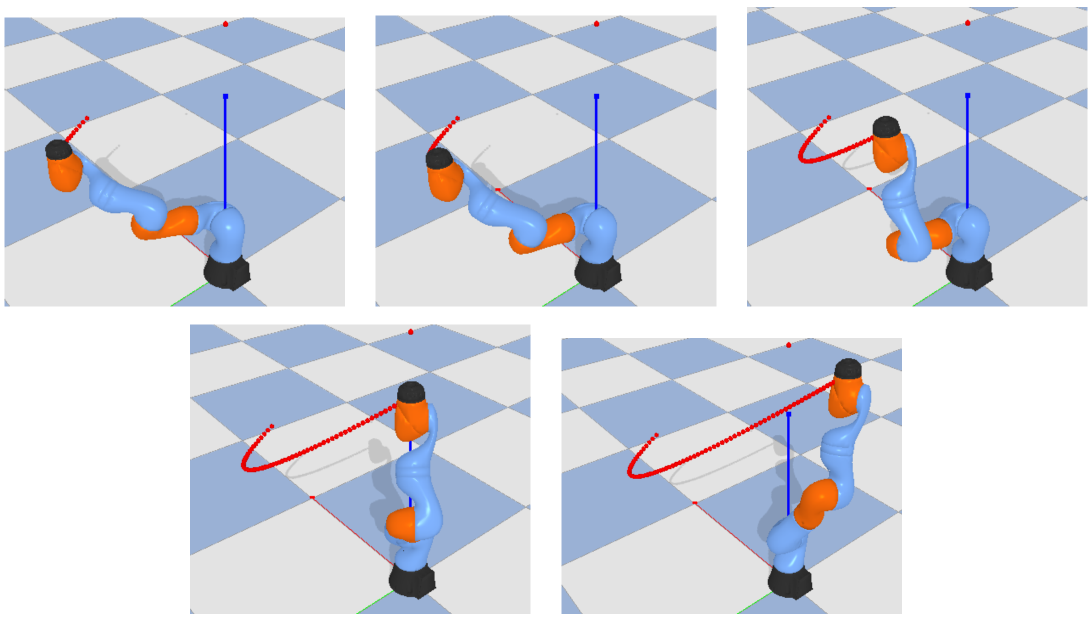


Figure 2.3: 3D DMP trajectory simulated on Kuka LBR iiwa 7 axis industrial robot

2.1.3 Joint Space DMP

The joint space DMP is a special case where each of the transformation system represents a joint of the robot. Like previoulsy mentioned the simulation is done on Kuka LBR iiwa. thus the transformation system will be of size 7. The each joint of the robot follows a sine wave trajectory in time.

Thus, the desired trajecotry in this case is:

$$\boldsymbol{\Theta(t) = \sin(t)} \tag{2.3}$$

where Θ is the vector of joint angles and $t \leq T$ where T is the total time of the trajectory. The parameters of the DMP are: $\alpha_z = 10, N_{bfs} = 1000$ with the canonical system parameters as: $\alpha_x = 0.5, \tau = 1$.

The resultant DMP trajectory simulated in DMP is shown in Fig. 2.4

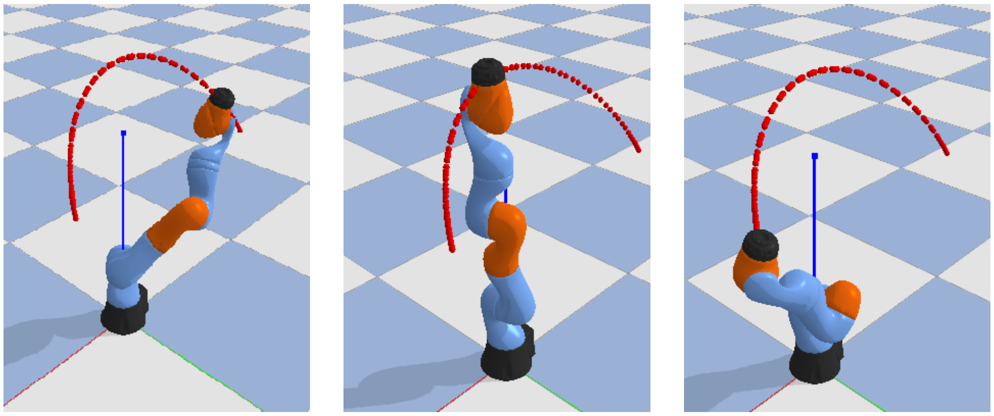


Figure 2.4: Joint space DMP trajectory

2.2 Orientation DMP

As mentioned previously, unlike Cartesian positions, the elements of orientation representations are constrained. And these constraints need to be encoded in the transformation system of DMPs.

There are many ways in which the orientation is represented. In this work, we will use the quaternion representation.

2.2.1 Quaternion Representation of DMP

A quaternion or unit quaternion $\boldsymbol{q} = v + \boldsymbol{u}$ provides a representation of the orinetation of robots end effector, where The DMP equations for unit quaternion can be represented as:

$$\tau\dot{\boldsymbol{\eta}} = \alpha_z(\beta_z * 2 * log(\boldsymbol{g_q} * \bar{\boldsymbol{q}}) - \boldsymbol{\eta}) + \boldsymbol{f_q}(\boldsymbol{x}) \tag{2.4}$$

$$\tau\dot{\boldsymbol{q}} = \frac{1}{2}\boldsymbol{\eta} * \boldsymbol{q} \tag{2.5}$$

where, $\boldsymbol{g_q} \in \S^3$ is the goal orientation, the quaternion conjugate is defined as $\bar{\boldsymbol{q}} = v - \boldsymbol{u}and*$ denotes quaternion multiplication. η is the scaled angular velocity ω and is treated as unit quaternion with xero scalar component. The logaritminc mapping is defined as:

$$\log \boldsymbol{q} = \begin{cases} \arccos{(v)}\dfrac{\boldsymbol{u}}{||u||} & \boldsymbol{u} \neq 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.6}$$

The above equations are based on the fact that the unit quaternion that takes $\boldsymbol{q_1}$ to $\boldsymbol{q_2}$ is given by $\Delta\boldsymbol{q} = \boldsymbol{q_2} * \bar{\boldsymbol{q_1}}$.Unlike with Rotation Matrices the logaritminc mapping defined on $\S^3$ has no discontinuity boundary, just a single singularity at a sinlg e quaternion $\boldsymbol{q} = -1 + [0, 0, 0]^T$.

The nonlinear forcing term is defined similar to the standard discrete DMP, and is as follows:

$$\boldsymbol{f_q(x)} = \frac{\sum_{i=1}^{N} w_i^o \Psi_i(x)}{\sum_{i=1}^{N} \Psi_i(x)} = \tau\dot{\boldsymbol{\eta}} - \alpha_z \left(\beta_z * 2 * \log\left(\boldsymbol{g_q} * \bar{\boldsymbol{q}}\right) - \boldsymbol{\eta}\right) \tag{2.7}$$

where, w_i^0 is the weights related to the orientation DMP, and like the discrete DMP can be solved using similar weight calculation methods.

To create a DMP system to reach a desired orientation, we created a orientation trajectory of quaternions using Spherical Liner Interpolation (SLERP). In SLERP, the quaternions are interpolated using the following formula:

$$\text{SLERP} = \boldsymbol{q_0}\left(\boldsymbol{q_0^{-1}}\boldsymbol{q_1}\right)^t \tag{2.8}$$

where $\boldsymbol{q_0}$ is the initial quaternion, $\boldsymbol{q_1}$ is the goal quaternion, and$\boldsymbol{q_0^{-}1q_1}$ is the quaternion that takes the initial quaternion to the final quaternion. The results of the quaternion DMP where $\boldsymbol{q_0} = [0.70710678, 0, -0.70710678, 0]$ is interpolated to $\boldsymbol{q_g} = [0.37796447, 0.75592895, 0.37796447, -0.37796447]$ is shown below.

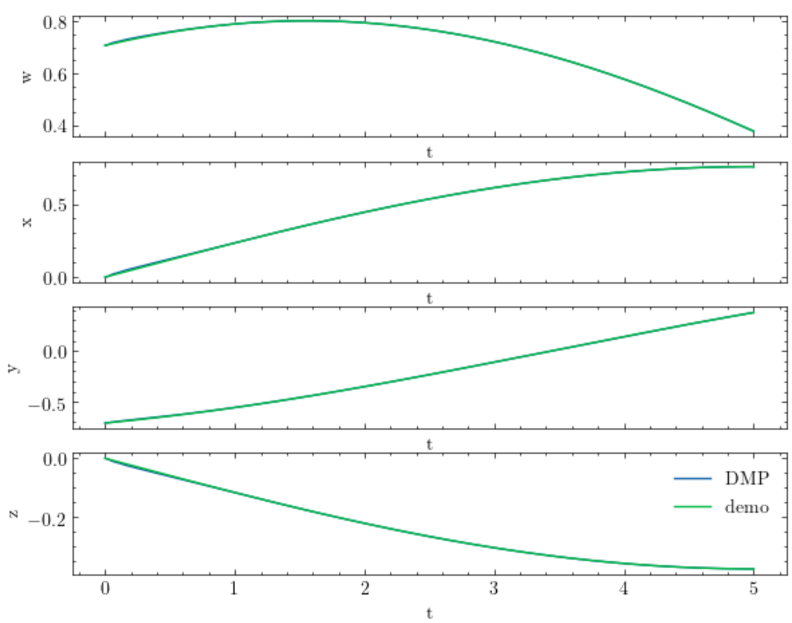


Figure 2.5: Quaternion DMP

2.3 6DOF DMP

A robot can truly make the most use of its workspace by moving in both cartesian and orientation space. Thus it becomes important to implement DMP in 6 DOF. We combined DMPs in Cartesian and Orientation space mentioned in the previous sections to create a 6DOF DMP. To make the implementation simpler, the demonstrated trajectory, in position and orientation space is recorded and rolled out separately, then the trajectory followed by the DMP system is then achieved via a cartesian velocity controller. In this section, the robot was moved between initial position [0.4,-0.2,0.8] with orientation [-0.17494102, 0.68512454 , -0.17494102 , 0.68512454] to final position [-0.2,0.4,1] with orientation [0.2474,-0.1,0.1,0.9689]. The results of this are shown below.

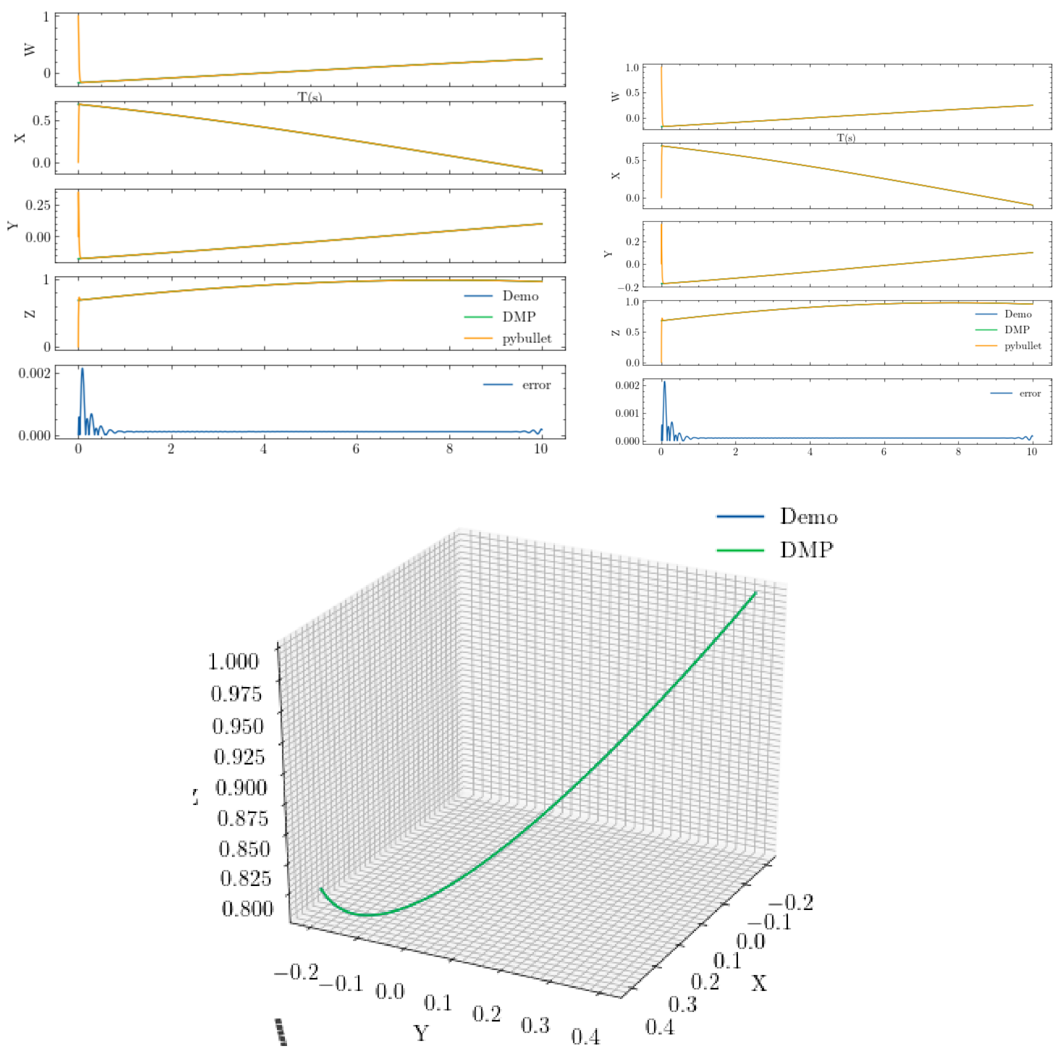


Figure 2.6: 3D DMP trajectory simulated on Kuka LBR iiwa 7 axis industrial robot

This trajectory was then simulated via Pybullet. A cartesian velocity controller was used in achieving the desired trajectory. A description of the controller used is given below.

$$\dot{\boldsymbol{q}} = J^{-1}\left(\boldsymbol{K_p}(\boldsymbol{x_d} - \boldsymbol{x})\right) \tag{2.9}$$

where J is the geometric Jacobian of the system, and x_d,x are the desired and current pose of the system. The error between the desired and current quaternion in the pose difference is calculated from Eq. 2.10 , and only the vector term is used

$$\Delta\boldsymbol{q} = \boldsymbol{q_{des}} * \bar{\boldsymbol{q}} \tag{2.10}$$

in the controller. The sign of the real part of the quaternion difference can be multippled to ensure that shortest rotation is followed. The tuning parameters were, K_p = [55, 55, 55, 44, 44, 44]. The results of the pybullet simulation are shown below.

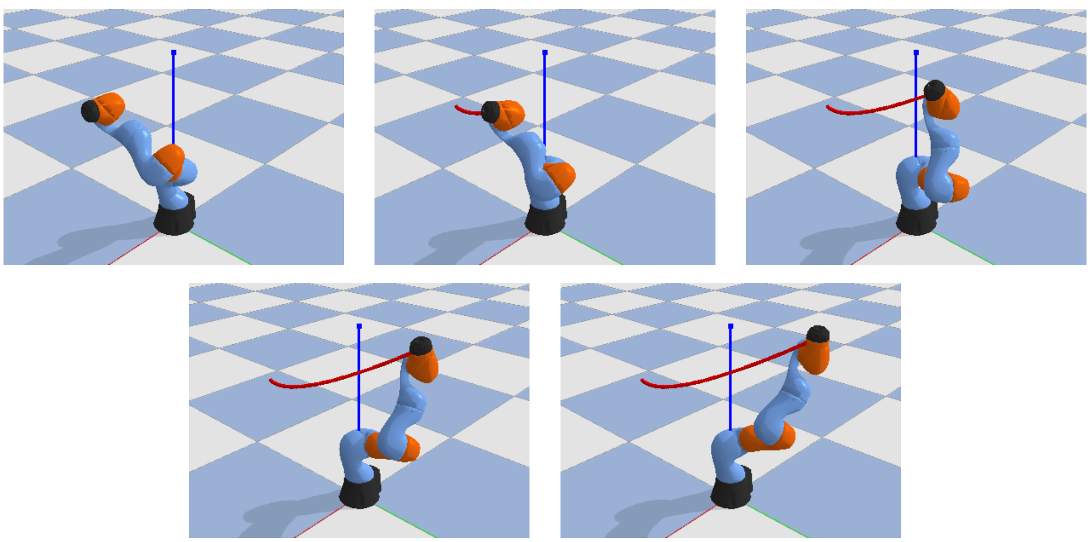


Figure 2.7: 6DOF DMP trajectory simulated on Kuka LBR iiwa

Next we considered the extension of Dynamic Movement Primitives in the case of Obstacle Avoidance.

Chapter 3

Obstacle Avoidance using Dynamic Motion Primitives

Avoiding obstacles is one of the most important tasks in robotics. To sucessfully complete a given task, a robot must avoid obstacles, stationery and moving.

Especially in Dynamic Movement Primitives where there exists an online change in trajectories, it seems fitting to have an obstacle avoidance approach that evolves in time with the movement primitives. There exists many approaches to obstacle avoidance in movement primitives, and the one that we have implemented uses dynamic potential field as an attraction-repulsion system, to avoid obstacles.

3.1 Potential Fields for Obstacle Avoidance

3.1.1 Static Potential Fields

The concept of the potential field was first introduced by Khataib. Here each obstacle acts as a repulsive potential field defined by Eq (3.1). Hence the obstacle "exerts" a force on the robot, given by the gradient of the potential field, i.e. $\boldsymbol{\varphi}(\boldsymbol{x}) = -\nabla U(x)$.

$$U_{static}(\boldsymbol{x}) = \begin{cases} \frac{\eta}{2}\left(\frac{1}{p(\boldsymbol{x})} - \frac{1}{p_0}\right)^2 & p(\boldsymbol{x}) \leq p_0 \\ 0 & p(\boldsymbol{x}) > p_0 \end{cases} \tag{3.1}$$

where, p_0 is the radius of influence of the obstacle, and η a constant gain paramerter.

This field is called the static potential field because it is not influenced by the obstacles and the robots velocity. This robot external force is then added to the trandoformation system of DMP which can be written as:

$$\tau\dot{z} = \alpha_z(\beta_z\,(z - z_g) - \dot{z}) + f(x) + \boldsymbol{\varphi}(\boldsymbol{x}, \boldsymbol{v}) \tag{3.2}$$

Peter Pastor showed that this method did not allow for smooth obstacle avoidance, and proposed a new method which we implemented in this work.

3.1.2 Dynamic Potential Fields

The dynamic potential field is a function of the robots (or its end effectors) position $\boldsymbol{x}$ and velocity $\boldsymbol{v}$, hence the term *dynamic*.

The dynamic potential field is defined to achieve the following properties:

- The magnitude of the potential decreases with the distance from $\boldsymbol{x}$ to the obstacle.
- The magnitude of the increases with the increase in the robots velocity.
- The magnitude of the potential should increase with the angle between the robot's velocity vector and the obstacle, and be zero if the angle is more than 90°.

Thus the formation can be formulated as:

$$U_{dynamic}(\boldsymbol{x}, \boldsymbol{v}) = \begin{cases} \lambda(-\cos\theta)^{\beta} \dfrac{||\boldsymbol{v}||}{p(\boldsymbol{x})} & \dfrac{\pi}{2} < \theta \leq \pi \\ 0 & 0 \leq \theta < \dfrac{\pi}{2} \end{cases} \tag{3.3}$$

where λ is a constant for the strength of the field, β is a hyperparameter. The angle θ is the angle

$$\cos\theta = \frac{\boldsymbol{v}^T \boldsymbol{x}}{||\boldsymbol{v}||\boldsymbol{p(x)}} \tag{3.4}$$

between the current velocity $\boldsymbol{v}$ and the robot's position $\boldsymbol{x}$ relative to the obstacle, and $p(x)$ denotes the distance between the $\boldsymbol{x}$ and the obstacle. The obstacle force as stated before is derived from a negative gradient of the potential field as

$$\boldsymbol{\varphi}(\boldsymbol{x}, \boldsymbol{v}) = -\nabla U_{dynamic}(\boldsymbol{x}, \boldsymbol{v}) \tag{3.5}$$

$$= \lambda(-\cos\theta)^{\beta-1} \frac{||\boldsymbol{v}||}{p(\boldsymbol{x})} \left(\beta \nabla_x \cos\theta - \frac{\cos\theta}{p(\boldsymbol{x})} \nabla_x p(\boldsymbol{x}) \right) \tag{3.6}$$

where,

$$\nabla_x p(\boldsymbol{x}) = \frac{\boldsymbol{x} - \boldsymbol{x_{obstacle}}}{p(\boldsymbol{x})} \tag{3.7}$$

$$\nabla_x \cos\theta = \frac{p(\boldsymbol{x})\boldsymbol{v}^T - \boldsymbol{v}^T x . \nabla_x p(\boldsymbol{x})}{||\boldsymbol{v}||p^2(\boldsymbol{x})} \tag{3.8}$$

3.2 Multiple Obstacle Avoidance

The formulation described above is for a single point like obstacle. One advantage of this approach is that it can be easily extended to avoid multiple obstacles, simply by adding the force field of each obstacle. Thus for N obstacles, we can write

$$\varphi_{total} = \sum_{i=1}^{N} \varphi_i(\boldsymbol{x}) \tag{3.9}$$

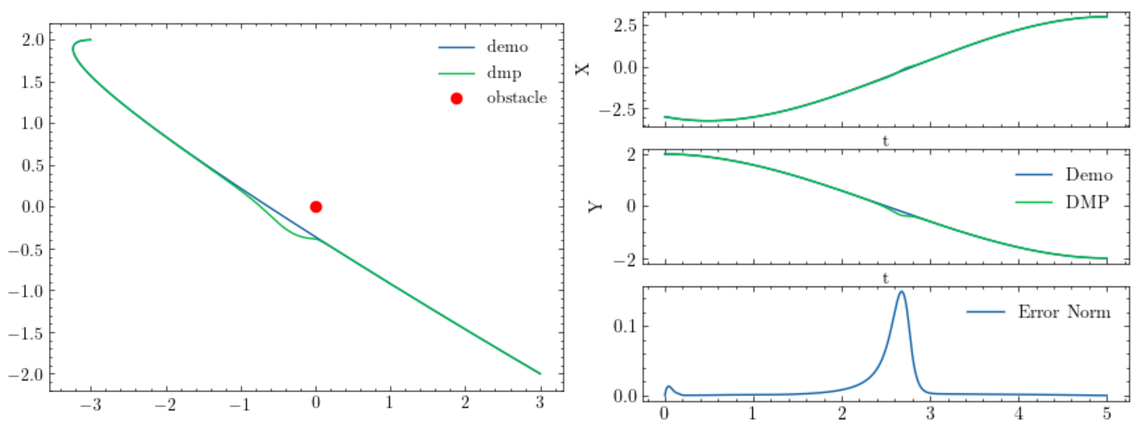
where φ_i is the force field of the i^{th} obstacle given by Eq. 3.6. An important point to note is that this method, *does not* gurantee the convergence to goal pose, due to the presence of local minima in the potential field.

3.3 Implementation and Results

The above formulation like the previous results is implemented in Python and simulated in Pybullet.

3.3.1 2D Obstacle Avoidance

A minimum acceleration trajectory between the initial and goal position of [-3,2] to [3,-2] with initial velocity of [-1,0]. The obstacle is placed at the origin. The desired trajectory passes close to the obstacle and the force field from the obstacle makes sure that the robot avoids the obstacle.



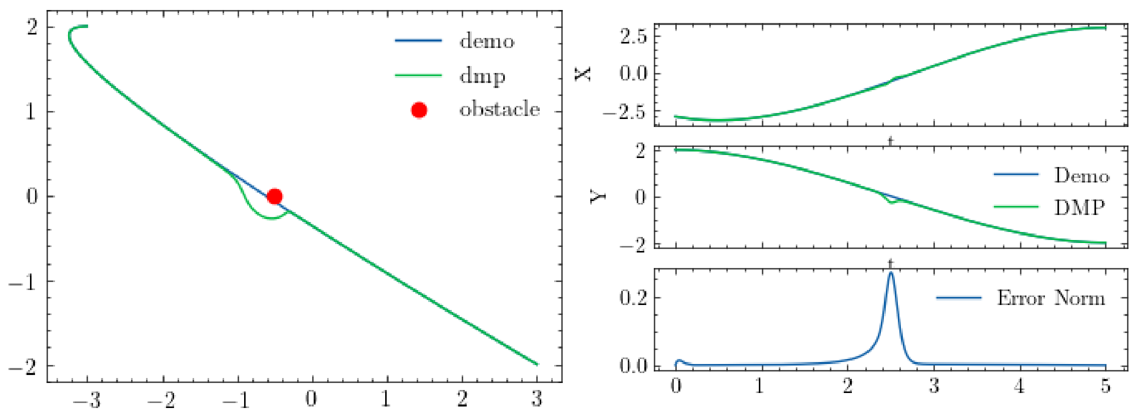
(a) 2D Obstalce Avoidance

(b) Deviation from the desired trajectory

Figure 3.1: Obstacle Avoidance in 2D and the deviation due to obstacle

The graph in Figure 3.1b shows the deviation from the demonstrated trajectory when the trajectory is close to the obstacle.

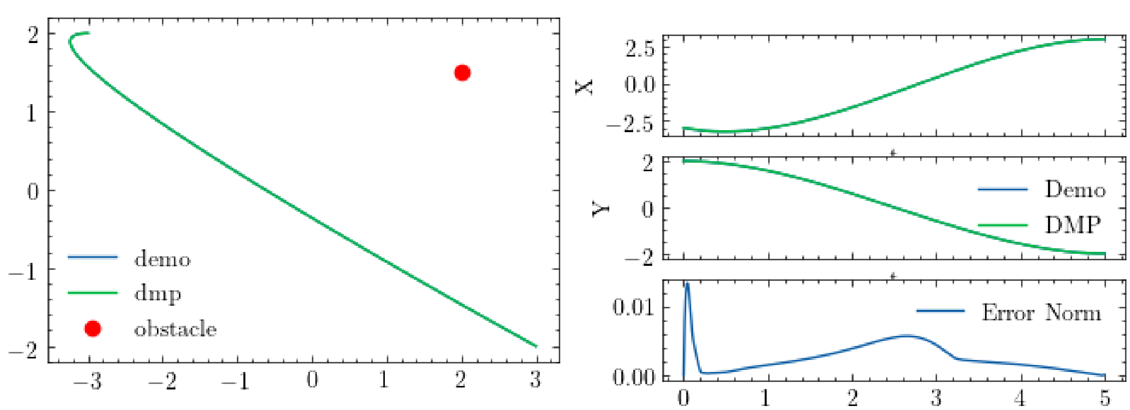
This same effect is magnified when the obstacle is *on* the demostrated trajectory, which is shown in Figure 3.2, where the obstacle was placed at [-0.5,0]. As mentioned in the formulation that we want the force field to decrese with the distance from the obstacle, which is the case when the obstacle is placed at [2,1.5]. In this case their is litte to no deviation from the demostrated trajectory as demonstrated in Figure 3.3. The parameters for the dynamic potential field were $\lambda = 10$ and $\beta = 2$



(a) 2D Obstalce Avoidance

(b) Deviation from the desired trajectory

Figure 3.2: Obstacle Avoidance in 2D and the deviation due to obstacle when the obstacle is very close to the demonstrateed trajcetory



(a) 2D Obstalce Avoidance when the obstacle is far away

(b) Deviation from the desired trajectory

Figure 3.3: Obstacle Avoidance in 2D and the deviation due to obstacle when the obstacle is far away from the demonstrateed trajcetory

Moving Obstacle Avoidance

To demonstrate the ability to avoid moving obstacles, an obstacle is placed at origin and is moved with a constant velocity of [-0.1,-0.1], with the parameter $\lambda = 10$ and $\beta = 2$. The demonstraed trajectory is kept the same. The results of whoch can be seen below.

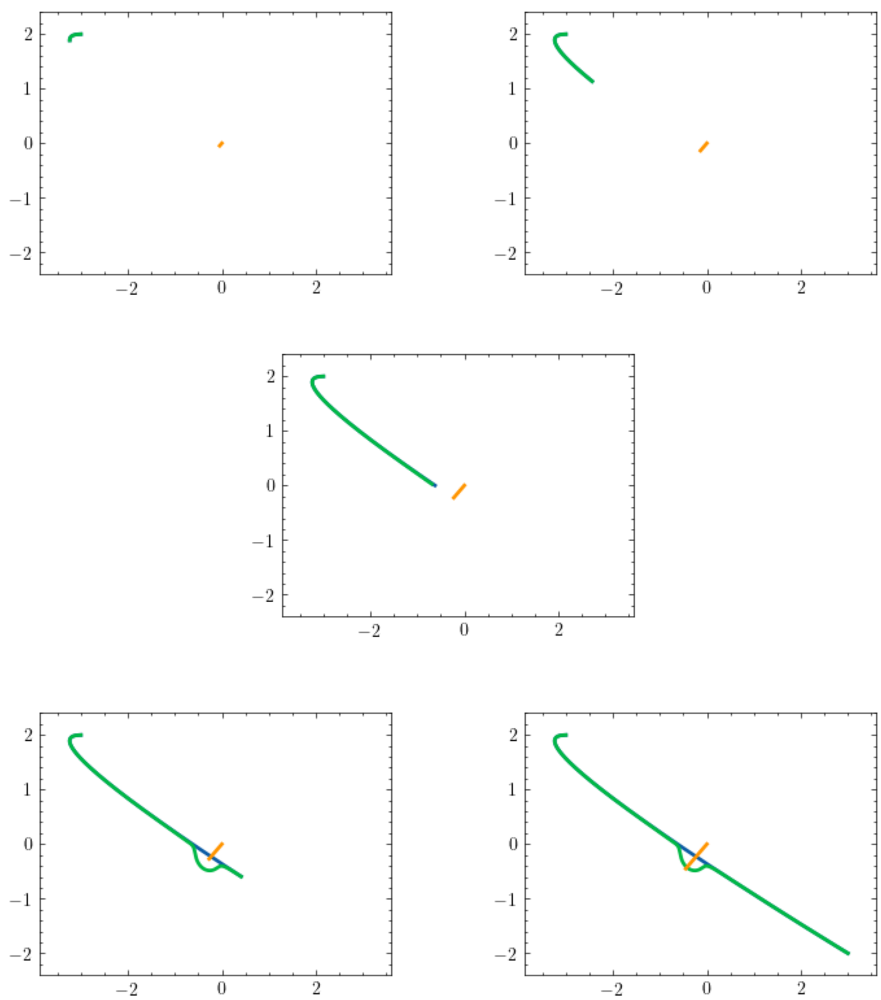


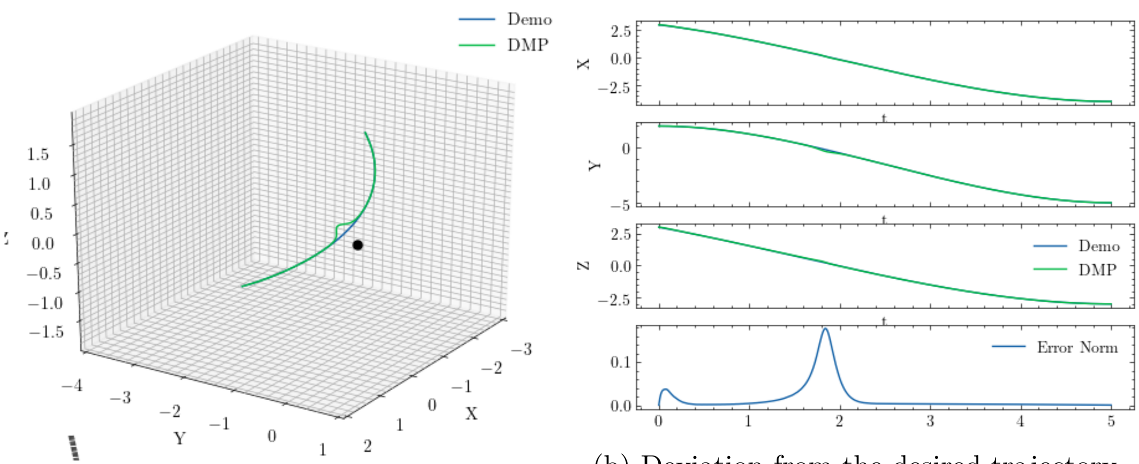
Figure 3.4: 2D obstacle avoidance of moving obstacles

The green path is the trajectory simulated by the DMP to follow the blue demonstrated trajectory, whereas the orange path is the path of the obstacle.

3.3.2 3D Obstacle Avoidance

The similar structure used in the 2D case is used in the 3 dimensional space. A minuimum acceleration trajectory is demonstrated between the points [3,2,3] to [-

4,-5,-3] with initial velocity of [-1,0,-1.3], where the obstacle is placed at the origin. The parameters for the potential field in this case are, $\lambda = 5$ and $\beta = 2$.



(a) 3D Obstalce Avoidance

(b) Deviation from the desired trajectory

Figure 3.5: Obstacle Avoidance in 3D and the deviation due to obstacle from the demonstrateed trajcetory

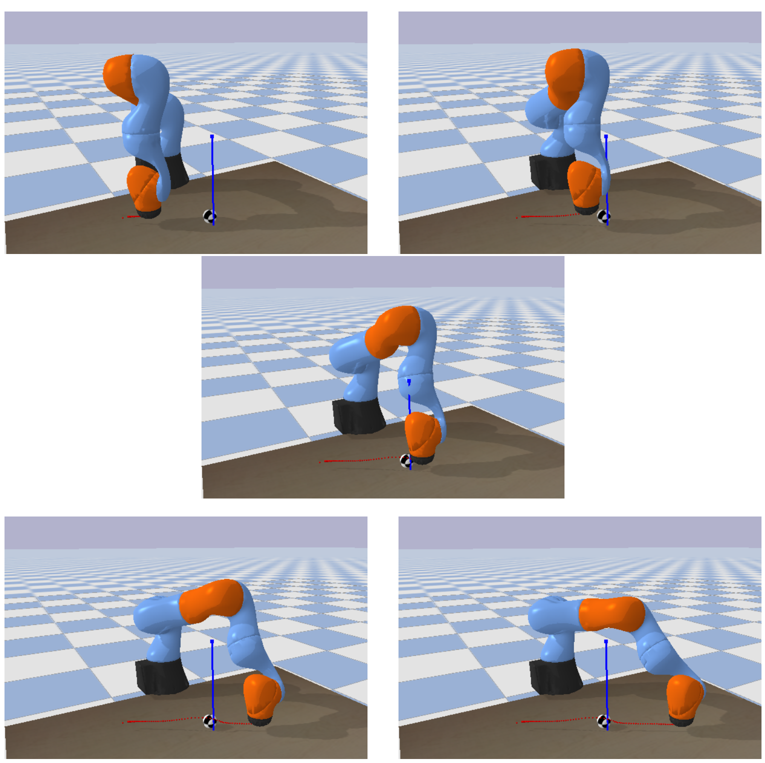


Figure 3.6: 3D obstacle avoidance of obstacles in Pybullet