Hochschule
Bonn-Rhein-Sieg
University of Applied Sciences

b-it Bonn-Aachen International Center for Information Technology

R&D Project

Dynamic Motion Primitives

Abhishek Padalkar

Submitted to Hochschule Bonn-Rhein-Sieg,
Department of Computer Science
in partial fullfilment of the requirements for the degree
of Master of Science in Autonomous Systems

Supervised by

Prof. Dr. Paul Plöger
Alex Mitrevski

August 2018

I, the undersigned below, declare that this work has not previously been submitted to this or any other university and that it is, unless otherwise stated, entirely my own work.

Date	Abhishek Padalkar

Abstract

Your abstract

Acknowledgements

Thanks to ....

Contents

List of Figures

List of Tables

1

Introduction

The ease with which humans and animals accomplish extremely complex motions has influenced the research in robotic motion planning and control at various levels, right from motor control to high level planning. It is widely accepted that humans learn a great variety of movements, and that these movements are stored in some form in our memories [14]. One key observation can be made that the biological motions are consisted of *motion primitives* (basic units of motion) perfected through experience over time[22]. This can be concluded from the example of a tennis player. A tennis player takes months of practice to perfect his *move* and to learn when to use it as well. This example is just a representative of vast number of skills acquired by humans and animals through experience. Various efforts have been made to adopt the concept of such motion primitives to generate robust robot control policy. Many variants of motion primitives are summarized in [12] and [3] which are necessarily model-free motion planning approaches because the primitives are learned independently without considering robot and environment model and validated at the time of execution. A motion primitive framework built around second order differential equation representing mass-spring damped system called *Dynamic Movement Primitives* is particularly famous and this work uses the same.

To effectively combine and use motion primitives to achieve complex tasks, a knowledge representation framework is needed which can be used to store the knowledge about the motion primitive such as pre-conditions on motion primitives, effects of the motion primitive on the environment, feasibility, resource requirements,

etc. This hypothesis can be backed by above example of tennis player. While playing, a tennis player has to choose his move considering various factors like, direction of approaching ball, possible direction of ball after hitting it, possibility of racket hitting other player in case of double's game, etc. A wrong move may have undesirable consequences. Tennis player acquires this knowledge through experience. But in case of robots, this initial knowledge can be hard-coded, stored and made available at the time of planning. Large community has been addressing the problem of efficiently learning and generalizing motion primitives. But knowledge representation and its effective use to generate complex behaviors is still an open issue.

Dynamic motion primitive is essentially an *Learning from Demonstration* approach. A trajectory in joint space or task space is obtained from human demonstration. Then the control policy behind that trajectory is learned in attractor space of a nonlinear dynamic equation.

Possible alternative for above mentioned biological skills is model-based motion planning and model-based control policy search. Both of these need a fairly accurate model of robot as well as the environment. Need of skills and experience required for motion execution is replaced by the accurate model of robot and environment.

During this Research and Development project, Dynamic movement Primitives were implemented with KUKA YouBot mobile manipulator along and the existing inverse kinematic velocity controller to effective use of DMP framework in mobile manipulation. Crucial modifications were made in already implemented inverse kinematic solver. Various experiments were performed to prove the usability of DMPs in ROBOCUP scenario and identify the need for knowledge base for DMPs.

1.1 Motivation

To be done .....

1.1.1 Combining Motion Primitives

Though above framework is endowed to learn complex motions, it is always desirable to learn simple motions and combine them to do complex task. By doing so, we can increase the re-usability of each motion primitive in other tasks. Nemec et

al and Lioutikov et al presented approaches for sequencing simple motion primitives learned for a particular task in smooth manner. But for combining DMPs learned for one particular task to do a different task, a framework for knowledge base representation is needed. Such framework will allow not only the representation of DMP, but also the knowledge about a DMP which allows us to decide if DMP is suitable to be combined or not.

1.2 Challenges and Difficulties

To be done

1.3 Problem Statement

To be done ...

2

State of the Art

2.1 Robot Motion Planning

"...eminently necessary since, by definition, a robot accomplishes tasks by moving in the real world." - J.-C. Latombe (1991). The first requirement of a robot, no-matter whether it is a manipulator, a mobile robotic platform or a drone, is to be able to move in the environment to accomplish its tasks. Robot Motion Panning is a decades old concept and over the time various algorithms have been developed to tackle the problem of planning which are good enough to provide solutions for real world situations. Few of them are summarized below.
Potential field methods for motion planning were proposed in the early period, which follow the gradient of a potential that guides a robot to its goal [10]. It is difficult, though, to come up with a general mechanism to escape local minima of a potential function or design a potential function that has only one minimum.
Another family of planning algorithms is composed of heuristic search techniques (e.g., A*) that operate over a discretization of possible robot configurations. These algorithms provide resolution completeness: A path will be found if the discretization is fine enough. A careful choice of resolution and heuristics is critical for efficient heuristic search. But complexity of these algorithm makes it hard to scale them for higher dimensions and adopt them for constraints [11].

Dynamic programming is one the algorithm which is proven to be good fo low dimensional problems. It is mathematical optimization method where set

of constraints on robot motion is provided and algorithm tries to optimize the trajectory to meet all the constraints.

Recently, approaches that use penalty functions to optimize the trajectory have been proposed. They soft the hard constraints into soft constraints and combine them into one formulation. By optimizing the penalty function, a optimized trajectory against parameters like time, energy, etc. is obtained. But these algorithms suffer from the problem of being trapped in local minima and may require huge amount of computation leaving them useless for real time application in robots with limited computation power[11].

Sampling-based algorithms take a very different approach. They randomly sample valid robot configurations and form a graph of valid motions. Many algorithms provide probabilistic completeness: The probability of finding a solution goes to 1 with the run time of the algorithm, provided a solution exists. Sampling-based motion planning algorithms are effective at solving motion planning problems in a broad range of settings with minimal changes, including very-high-dimensional systems[11].

The Rapidly-exploring Random Tree (RRT) is sampling based exploration algorithm for quickly searching high dimensional spaces that have both global constraints (arising from workspace obstacles and velocity bounds) and differential constraints (arising from kinematics and dynamics) [13]. The key idea is to bias the exploration toward unexplored portions of the space by randomly sampling points in the state space, and incrementally pulling the search tree toward them. RRT can be used for motion planning by using randomly generated tree in configuration space and validating it using the constraints defined. Not every solution found by RRT is feasible and optimal which makes method incomplete and non-optimal. But this method is quiet effective in in high dimensional problems with numerous constraints.

MoveIt!, one of the widely used motion planning and control framework, uses sampling based motion planing algorithms provided by Open Motion Planning Library (OMPL) by default. The solutions generated by sampling based algorithms, which use inverse kinematic solvers to find configuration space samples, are often not continuous. It can be understood by following example.

Moreover, motion plans are often generated without considering robots ability to

execute them in real world, the uncertainties like slip, inaccuracy in executions, and dynamic changes in the environment like real time change in goal or moving obstacle. Often these dynamic changes and uncertainties lead to inaccurate manipulation and re-planning. Theoretically it is possible to accommodate these dynamic environmental parameters into a plan but it hard and complex to do so.

Another way to go is to generate a control policy instead of generating motion plan. Mechanisms can be developed to accommodate uncertainties and dynamic changes in environment and easily adopted into control policy.

2.2 Control Policy Search in Robotics

Control policy search algorithms can be categorized into two categories, *model-based policy search* and *model-free policy search*. Model based policy search algorithms need a fairly accurate model of robot as well as the environment. Using the observed trajectories, forward model of the robot's dynamics and environment is learned. This forward model is used for internal simulations for validation of the generated trajectory. (e.g collision, reachability). These methods heavily suffer from inaccurate model. Control policies generated by using inaccurate model are not robust and can be dangerous to execute in real world scenarios.

In model-free methods, a trajectory generation policy is learned without considering model of the robot. An external framework is used for validation and control of trajectories generated. Learning a policy is often easier than learning accurate forward models. Hence model-free policies are more widely used than model-based methods[3].

Deisenroth et al.[3] presented survey on policy search methods in robotics. Model free policy search methods have obvious advantage over model-based methods as accurate model of robot and environment is not needed which is often very difficult to obtain. Paper also discusses main policy representations with their advantages and disadvantages.

Linear Policies: Linear controllers are the most simple time independent representation. Policies are represented as linear combination of basis function. However, specifying the basis functions by hand is typically a difficult task, and, hence, the application of linear controllers is limited to problems where appropriate basis functions are known[3].

Radial Basis Functions Networks: A typical nonlinear time independent policy representation is a radial basis function (RBF) network. RBF networks are powerful policy representations, they are also difficult to learn due to the high number of nonlinear parameters. RBF networks are local representations, they are hard to scale to high-dimensional state spaces[3].

Dynamic Movement Primitives: Dynamic Movement Primitives (DMPs) are the most widely used time-dependent policy representation in robotics.

2.3 Control Policy Representation

In the field of Learning from Demonstration (*LfD*) and Reinforcement Learning, methods used for representing underlying control policy has significant impact on generalization ability, stability and computational complexity of the approach. In LfD, control policy is learned from state-action examples whereas in reinforcement learning control policy is learned form general experience of the robot.

In [12], Kober et.al. presented comprehensive survey on function approximation methods for control policy representation. Much of the success of reinforcement learning methods has been due to the clever use of such approximate representations. Main approaches for policy representation :

Via Points & Splines : An open-loop policy may often be naturally represented as a trajectory, either in the space of states or targets or directly as a set of controls. Such spline-based policies are very suitable for compressing complex trajectories into few parameters. Typically the desired joint or Cartesian position, velocities, and/or accelerations are used as actions. To minimize the required number of parameters, not every point is stored. Instead, only important via-points are considered and other points are interpolated.

Neural Networks : Neural networks are another general function approximation used to represent policies. Neural oscillators with sensor feedback have been used to learn rhythmic movements where open and closed-loop information were combined, such as gaits for a two legged robot.

Motor Primitives : Motor primitives combine linear models describing dynamics with parsimonious movement parametrizations. While originally biologically-inspired, they have a lot of success for representing basic movements in robotics

such as a reaching movement or basic locomotion. These basic movements can subsequently be sequenced and/or combined to achieve more complex movements. Dynamic Movement Primitives is the most successful example of such control policy.

Gaussian Mixture Models and Radial Basis Function Models : When more general policies with a strong state-dependence are needed, general function approximators based on radial basis functions, also called Gaussian kernels, become reasonable choices. This approach has been used to generalize a open-loop reaching movement and to learn the closed-loop cart-pole swing-up task.

Non-parametric Policies : Polices based on non- parametric regression approaches often allow a more data-driven learning process. This approach is often preferable over the purely parametric policies listed above because the policy structure can evolve during the learning process. Such approaches are especially useful when a policy learned to adjust the existing behaviors of an lower-level controller.

2.4 Control Policy Representation Using Movement Primitives

Movement Primitives (MPs) are commonly used for representing and learning basic movements in robotics. MP formulations are compact parameterizations of the robots control policy. Modulating their parameters permits imitation and reinforcement learning as well as adapting to different scenarios [19].

2.4.1 Motion Primitives Using Principle Component Analysis

In [14], authors have proposed a high-level framework for robot movement coordination and learning that combines elements of movement storage, dynamic models, and optimization, with the ultimate objective of generating natural, human-like motions. Movement primitive is represented and stored as a set of joint trajectory basis functions; these basis functions are extracted via a principal component analysis of human motion capture data. Dynamics based optimization

is performed on the learned movement primitives to optimize them to use minimum torque. They also discuss about the general framework to deploy the movement primitive in real world task. In this, a task parser uses description of task to generate a sequence of the movements. Then the movement compiler chooses appropriate motion primitives to perform sequence of movements.

This method learns movement primitives in terms of joint trajectories, hence it has limited generalization ability when it comes to task space goal execution. This class of movement primitives does not incorporate feedback from the environment.

2.4.2 Probabilistic Movement Primitives

Paraschos et al. in [19] present probabilistic approach to model movement primitives called Probabilistic Movement Primitives (ProMPs). In this framework, a MP describes multiple ways to execute a movement, which naturally leads to a probability distribution over trajectories. These motion primitives are able to generalize the motion by encoding variance of the trajectory from multiple demonstration of same motion. Temporal modulation is possible with additional phase variable. The choice of the basis functions depends on the type of movement, which can be either rhythmic or stroke-based. ProMPs support simultaneous activation, match the quality of the encoded behavior from the demonstrations, are able to adapt to different desired target positions, and efficiently learn by imitation.

2.4.3 Dynamic Movement Primitives

Ijspeert et. al. in [8] used the system of autonomous second order differential equations for encoding control policy. This formulation was motivated by following five goals:

1. The ease of representing and learning a desired trajectory,
2. Compactness of the representation,
3. Robustness against perturbations and changes in a dynamic environment,
4. Ease of re-use for related tasks and easy modification for new tasks, and

5. Ease of categorization for movement recognition.

Trajectory information is encoded in the second order differential equation of damped mass spring system with a non-linear forcing term which is used for modifying the shape of trajectory. The non-linear forcing term is normalized weighted sum of Gaussians timed by a state vector whose evolution is guaranteed by another linear second order differential system. This whole system is stable and converges in limited time. Weights of the Gaussians are learned by using locally weighted regression. This work was the necessary foundation of the theory of dynamic motion primitives.

$$\dot{z} = \alpha_z(\beta_z(g - y) - z) \tag{2.1}$$

$$\dot{y} = z + \frac{\sum_{i=1}^{N} \psi_i w_i}{\sum_{i=1}^{N} \psi_i} \tag{2.2}$$

Above system of equations is essentially a simple second-order system with the exception that its velocity is modified by a nonlinear term (the second term in equation 2.2) which depends on internal states. These two internal states, (v, x) have the following second-order linear dynamics

$$\dot{z} = \alpha_v(\beta_v(g - x) - v) \tag{2.3}$$

$$\dot{x} = v \tag{2.4}$$

and ψ_i is given by,

$$\psi = exp(-\frac{1}{2\sigma_i^2}(\tilde{x} - c_i^2)) \tag{2.5}$$

where $\tilde{x} = (x - x0)/(g - x0)$

Use of this second dynamical system allows modification in the speed of execution, motion can even be halted.

Ijspeert et al. in [6] presented the idea of learning a complex control policy by transforming an existing simple canonical control policy instead of learning it from scratch. They chose stable differential equation system as the canonical policy. By nonlinearly transforming the canonical attractor dynamics using techniques from nonparametric regression, almost arbitrary new nonlinear policies can be generated

without losing the stability properties of the canonical system. Portions of the work presented in this paper were published in [8]. Preliminary work in [8] was extended with an improvement and simplification of the rhythmic system, an integrated view of the interpretation of both the discrete and rhythmic CPs, the fitting of a complete alphabet of Grafitti characters, and an implementation of automatic allocation of centers of kernel functions for locally weighted learning. In this work authors explained the properties of temporal and spatial invariance i.e. shape of the trajectories does not depend on goal separation or speed of execution. Also authors argued about the robustness against perturbations which later become the foundation for obstacle avoidance properties and extra coupling terms such as feedback. Experimental evaluation consisted of successful implementation on of rhythmic and discrete control policies on 30 DOFs hydraulic anthropomorphic robot using learning from demonstration technique. This evaluation also gave the glimpse of possible extension of this approach for movement recognition by observing co-relation between weights of the sample movements (stored as a library) and the one which is to be recognized. This was demonstrated by fitting of a complete alphabet of Grafitti characters with 84% success rate.
Stefan Schaal in [22] formulated DMPs and implemented DMP system on a 30 DOF Sarcos Humanoid robot. Main contribution of this paper was to define terminology for DMP framework and standardizing the equations by separating the canonical system. New DMP framework is represented by following set of equations,

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f \tag{2.6}$$

$$\tau \dot{y} = z \tag{2.7}$$

and the non-linear function f

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(t) w_i}{\sum_{i=1}^{N} \psi_i(t)} x(g - y_0) \tag{2.8}$$

where,

$$\tau \dot{x} = -\alpha_x x \tag{2.9}$$

and,

$$\psi_i = \exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2) \tag{2.10}$$

Eq. 2.9 is called canonical system which eliminates the time dependency. It is also used for co-ordination between different degrees of freedom. In typical robotic application, each DMP corresponds to one controlled variable, which might be, for example, one of the joint coordinates or one of the Cartesian coordinates and all DMPs share the same phase variable. In this work, humanoid robot demonstrated drumming task and tennis swing task. Point attractive behavior of dynamical system enabled the task of tennis swing as it is discrete in nature and limit cycle behavior enabled drumming task which is rhythmic or periodic in nature.
Pastor et al. in [21] provided a general approach for learning robotic motor skills from human demonstration. They identified important features of DMP framework represented by above equations. They are:

- Convergence to the goal g is guaranteed (for bounded weights) since non-linear term f vanishes at the end of a movement.
- The weights w_i can be learned to generate any desired smooth trajectory.
- The equations are spatial and temporal invariant, i.e., movements are self-similar for a change in goal, start point, and temporal scaling without a need to change the weights w_i.
- The formulation generates movements which are robust against perturbation due to the inherent attractor dynamics of the equations.

Based on the DMP representation of the movements library of movements was built by labeling each recorded movement according to task and context (e.g., grasping, placing, and releasing). Semantic information stored can be used to choose particular DMP for doing a certain task. Semantic information was added to movement primitives, such that they can code object oriented action. Differential equation used is formulated such that generalization can be achieved simply by adapting a start and a goal parameter in the equation to the desired position values of a movement. For sequencing DMPs, initial conditions of next DMP were modified and it was started before finishing the current one so that there is

no jump in the acceleration and velocity. The idea of building library of motion primitives and attaching semantics is coined in this work but such motion movement primitives were handpicked at the time of demonstration. No automated algorithm was presented to choose the combination of movement primitives for achieving complex task.

In [15], sequencing of motion primitives was implemented for the task of cutting vegetables. Basic primitives were which are necessary for performing given complex task were learned and then sequenced to perform the task. The experiment was conducted on Darias robot platform which consists of dual-arm setup using two 7 DOF KUKA Light Weight Robot arms. This work shows that sequencing learned motion primitives to do complex task is possible by enforcing goal state of preceding primitive and initial state of next primitive to be same. But this work doesn't specify any method for combining arbitrary motion primitives in order to do entirely new task.

Another approach for sequencing of DMPs is presented in [18]. A method for achieving continuous acceleration at the moment of transition from one DMP to another DMP is to represent DMPs in the form of 3^{rd} order differential equation. This representation will enable us to provide initial velocity along with position and yield differentiable acceleration term. Another way to achieve this is online Gaussian kernel functions modification of the second order dynamic motion primitives. Both methods were tested in simulation using a simple illustrative example, where authors joined two ramp functions, as well as in real-world experiments that included pouring a liquid into a glass, table wiping, and carrying a glass. The third experiment shows that continuous accelerations are essential when performing typical kitchen scenario tasks such as carrying a glass of liquid, even at relatively low velocities. It has been shown that both proposed approaches are appropriate when joining any combination of discrete and rhythmic motions for the cases where smooth accelerations are important.

Work presented in [20] solves problem of reproduction of motion in the presence of obstacles. It is achieved by using potential field centered around the obstacle and adding its gradient to the equation of motion. This work also compares different representations of potential fields for obstacle-link collision avoidance. This work also expand the previous work in order to learn motion primitives in end-effector

space. This work presents a modification in DMP framework to eliminate the restriction on separation between start and goal position. In previous formulation, start and goal position has to have a separation between them otherwise DMP could not leave the initial position. If this separation is very small, learned DMP might show unstable behavior (very large trajectories for very small change in goal position).
New formulation is as follows:

$$\tau\dot{v} = K(g - x) - Dv - K(g - x_0)\theta + Kf(\theta) + \psi(x, v) \tag{2.11}$$

$$\tau\dot{x} = v \tag{2.12}$$

$$\tau\dot{\theta} = \alpha\theta \tag{2.13}$$

Here forcing term $f(\theta)$ is no longer multiplied by $(g - x_0)$ term, hence DMP is no longer dependent on initial position and goal separation to take off. The term $\psi(x, v)$ is extra coupling term or perturbation term added to deal with the obstacle which is essentially the gradient of static or dynamic potential field around the obstacle which will repel the trajectory.
Static potential field is given by:

$$U_{static} = \frac{\eta}{2}(\frac{1}{p(x)} - \frac{1}{p_0})^2 \tag{2.14}$$

where p_0 is the radius of influence of the obstacle, $p(x)$ is distance of end effector from obstacle, and η is a constant gain. This field is zero outside the radius p_0.
Dynamic potential field is given by:

$$U_dynamic = \lambda(-cos(\theta))^\beta \frac{||v||}{p(x)} \qquad : \frac{\pi}{2} < \theta <= \pi \tag{2.15}$$

here, v is relative velocity of obstacle and θ is angle between relative velocity vector and position vector of obstacle with respect to point of interest (end effector or any point on links).

Both the potential fields were experimentally tested in simulation as well as on real robot. The conventional static potential method shows an unstable avoidance

movement, which oscillates. On the other hand, the dynamic potential method results in a smooth obstacle-avoidance movement. A approach of constraining the null space of the robot to avoid the collision with links of the robot was also tested with dynamic field. Two scenarios were evaluated : a) end-effector position (x) as input into the potential field, b) Closest point on manipulator as (x) input. In scenario a), large number of trajectories converged to goal than b). But scenario b) was better in terms of robustly avoiding collisions.
Meier et.al. presented the probabilistic representation of dynamic motion primitives in [16]. In this work, authors showed how DMPs can be reformulated as a probabilistic linear dynamical system with control inputs. Through this probabilistic representation of DMPs, algorithms such as Kalman filtering and smoothing are directly applicable to perform inference on proprioceptive sensor measurements during execution. Inference on this probabilistic form of the DMP allows us to measure the likelihood of DMP being successfully executed.
Karlsson et.al.[9] presented the approach for modification of dynamic motion primitives. A modified DMP is formed, based on the first part of the faulty trajectory and the last part of the corrective one. Demonstrations were autonomously interpreted for their quality when a new corrective demonstration was presented to the robot.
Hoffman et.al. in [5] presented solution for the problem of automatic real-time goal adaptation and obstacle avoidance in dynamic motion primitives.They also modified the DMP framework to enable use of DMP where there is no offset in initial and goal position. This was not possible in previous formulation.

2.4.4 Advantages of Dynamic Movement Primitives

- It is a model free learning approach.
- Any arbitrary trajectory can be learned in end-effector space as well as in joint space.
- Here learning is linear regression, so it does not need large dataset. One trajectory is sufficient ideally.
- Trajectories can be scaled in space as well as in time.

- Dynamic motion primitives can be initialized anywhere in the attractor space.
- Trajectory evolves as robot actually moves along the trajectory. Hence on-line modifications in the trajectory are possible.
- These modifications can be realized by introducing vanishing coupling terms in the differential equations (e.g. potential field around a obstacles [20]). Perturbations can be handled robustly due to this reason.
- Re-planning is not needed unless an event causing major disturbance in the environment occurs.

3

Methodology

3.0.1 Formalization of Dynamic Motion Primitives

Dynamic motion primitives use nonlinear differential equations to model the motion primitives. These differential equations is essentially represent a damped mass spring system. Attractor landscape of differential equations represent desired kinematic state of the robot. Over the time, various versions of DMPs were presented which were slightly different than one another, modified for specific use case or scenario. Following formalization of DMP is taken from [7]. A DMP can be represented by following set of equations,

$$\tau \dot{z} = \alpha_z(\beta_z(g - y) - z) + f(x) \tag{3.1}$$

$$\tau \dot{y} = z \tag{3.2}$$

and the non-linear function $f(x)$

$$f(x) = \frac{\sum_{i=1}^{N} \psi_i(x) w_i}{\sum_{i=1}^{N} \psi_i(x)} x(g - y_0) \tag{3.3}$$

where,

$$\psi_i = \exp(-\frac{1}{2\sigma_i^2}(x - c_i)^2) \tag{3.4}$$

and,

$$\tau \dot{x} = -\alpha_x x \tag{3.5}$$

Equation 3.1 and 3.2 are the first order representation of a autonomous non-linear second-order differential equation where $f(x)$ is the non-linear term. Upon solving these equations, we get state $[\ddot{y}, \dot{y}, y]$ at each time instance. This state represents the acceleration, velocity and position i.e. kinematic state of robotic system.

If the forcing term in eq. 3.1 is made 0 i.e. $f = 0$, these equations represent a globally stable second-order linear system with $(z, y) = (0, g)$ as a unique point attractor. With appropriate value for α and β (with $\beta_z = \alpha z/4$), the system can be made critically damped resulting in y monotonically and asymptotically converging towards g. Such a system implements a stable but trivial pattern generator with g as single point attractor [7].

By introducing term f, the path followed by the system in attractor landscape of differential equation from initial state to the goal state can be modified. This in turn modifies the trajectory followed by mobile robot or robotic manipulator in task-space or joint-space. This non-linear forcing function enables DMP framework to learn almost any arbitrary motion in end-effector space or joint space. It is essentially a weighted sum of equally spaced Gaussian functions denoted by ψ (eq. 3.4) activated at each time step by phase variable x. Forcing term $f(x)$ is learned from the demonstrated trajectories. It should be noted that the forcing term f is modified by separation between goal and initial position $(g - y_0)$, which ensures that the shape of trajectory is also modified by goal separation. Eq. 3.1 and 3.2 are together called *transformation system*, as they transform current state of the system to next state.

Equation 3.5 is called *canonical system*. x acts as the phase variable modifying forcing term $f(x)$ and hence the shape of the trajectory. It removes the the explicit time dependency of of the system, which eliminates the need of maintaining complex timing mechanisms to synchronize multiple DMPs. x initialized at 1 decays to 0 at the end of the motion ensuring convergence to goal state g. It is used to localize the basis functions (i.e., as a phase signal) but also provides an amplitude signal (or a gating term) that ensures that the nonlinearity introduced by the forcing term remains transient due to asymptotical convergence of x to zero at the end of

the discrete movement [7].

Term τ is time scaling factor which can be used to modify the speed of execution of the system by modifying the acceleration and velocity term.

Similar to the above discussed point attractor behavior, DMPs can also learn rhythmic motions exploiting the limit cyclic behavior of non-linear second order differential equation. To achieve rhythmic motion, we need to learn forcing term $f(x)$ which is periodic itself and makes system of equation 3.1 and 3.2 exhibit limit cyclic behavior. This can be achieved by choosing canonical system to be,

$$\tau\dot{\phi} = 1 \tag{3.6}$$

where $\phi \in [0, 2\pi]$ is the phase angle of the oscillator in polar coordinates and the amplitude of the oscillation is assumed to be r. Similar to the discrete system, this rhythmic canonical system serves to provide both an amplitude signal r and a phase signal ϕ to the forcing term f in equation 3.1.

$$f(r, \phi) = \frac{\sum_{i=1}^{N} \psi_i(x) w_i}{\sum_{i=1}^{N} \psi_i(x)} r \tag{3.7}$$

Point attractor and limit cycle behaviors exhibited by non-linear second order differential equation can be used to model discrete (point to point) and rhythmic motions of robotic arm or mobile platform.

During this Research and Development project, point attractor DMPs for discrete motion (point-to-point motion) were implemented. For the ease of implementation, above system was slightly modified as follows,

$$\ddot{y} = \tau^2 \alpha_y(\beta_y(g - y) - \dot{y}) + f \tag{3.8}$$

$$\dot{x} = \tau(-\alpha_x x) \tag{3.9}$$

This modification does'nt change the properties of original DMP formulation.

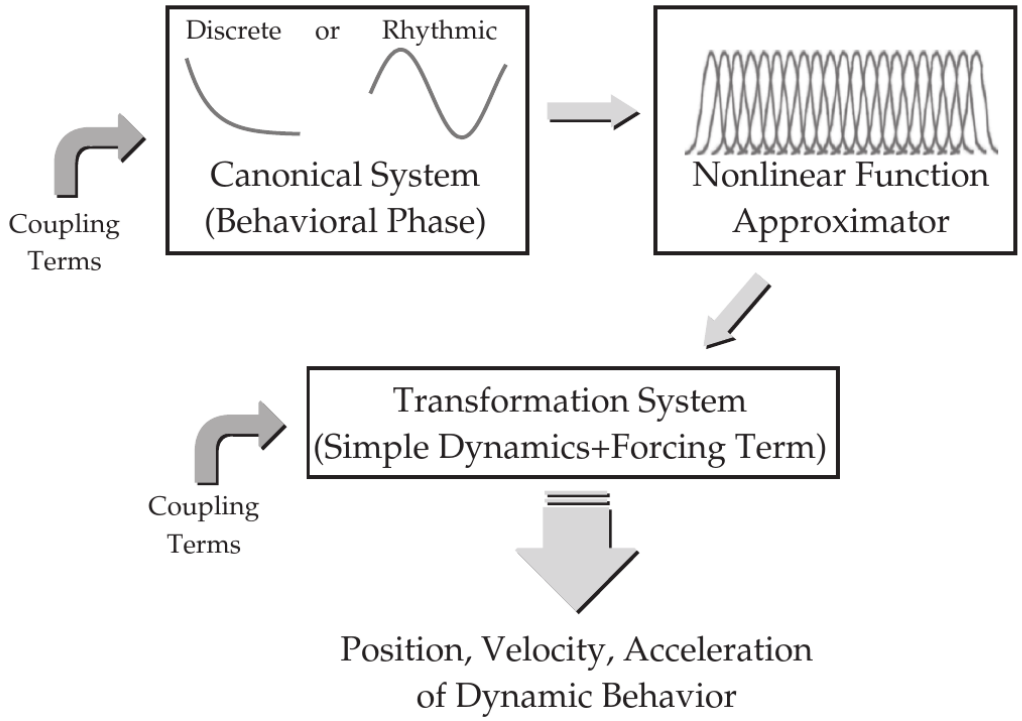

Figure 3.1: Dynamic Movement Primitive framework [7]

Figure 3.1 taken from [7] show composition of the DMP framework. The *Coupling Terms* in the figure denotes consists of feedback form the sensors, obstacle coupling terms, error in execution speed of the DMPs. Canonical system generates the phase variable x at each time step which is used for generating a non-linear forcing term. This forcing term then modifies the behavior of the kinematic motion commands generated by transformation system. Kinematic motion commands are then executed by robot controller (e.g. velocity controllers, torque controllers, etc.)

Figure 3.2 shows the path generated by 2 degrees-of-freedom DMP with no forcing term in Cartesian space. Figure illustrates the superposition of 2 DMPs in different degrees of freedom. DMP systems in X and Y axes follow the kinematic trajectories same as of damped mass-spring system and superposition of these two motion generate a straight line motion in 2D space.

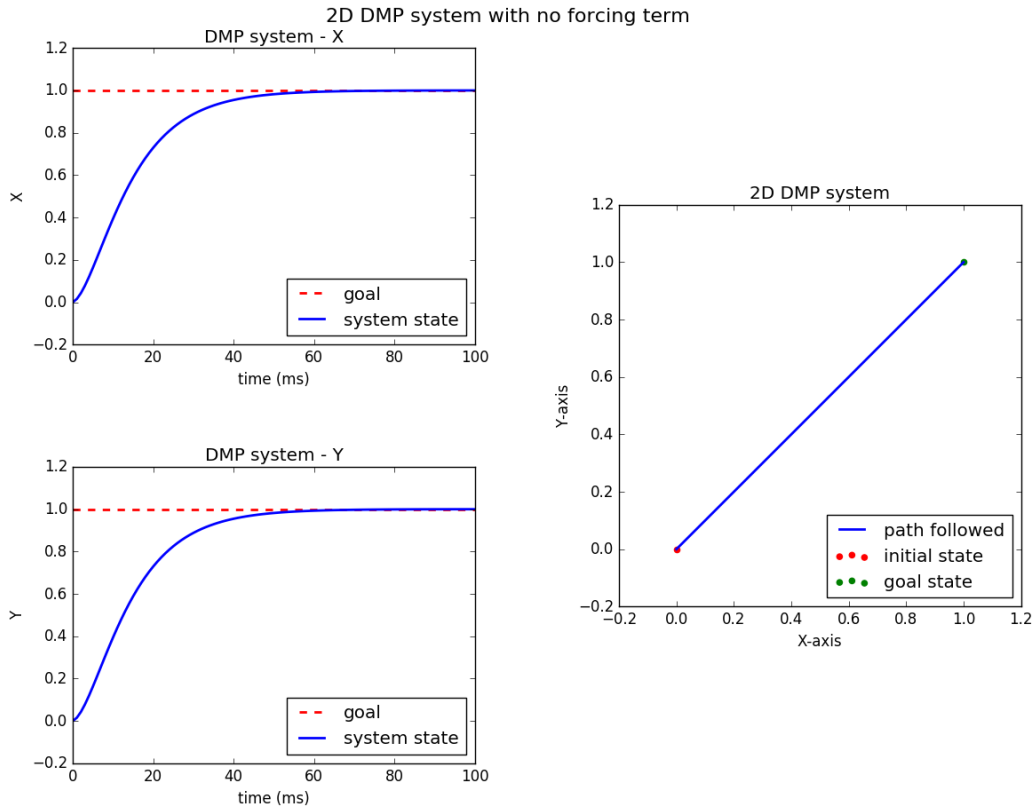

Figure 3.2: 2D DMP with no forcing term

3.0.2 Analysis of the effects of parameters used in DMP

For analyzing the effects of the various parameters used in DMP, a artificially generated step function trajectory was learned. Parameters were changed in order to evaluate their effects on the DMP. The evaluation criteria was the closeness between the desired path and the path generated by DMP after learning. The error between path was calculated as normalized point-to-point distance between discrete positions on two paths. Error was calculated as follows:

$$error = \frac{1}{N}\sum_{i=1}^{N} min\{\|P_i - Pd_1\|, \|P_i - Pd_2\|, \|P_i - Pd_3\|, ..., \|P_i - Pd_M\|\} \tag{3.10}$$

Where,
P is the path generated by DMP,
Pd is the original demonstrated path,
N is the number of poses in P,
M is the number of poses in Pd.

This error function evaluates the closeness of two paths, hence does'nt consider the deviation of the trajectory at each time step. It also does not consider the deviation in velocity and acceleration.

Analysis of other undesired behaviors like jumps was also done.

The artificial trajectory generated contains 100 2D poses sampled at equal time interval of 0.01 *sec*. Values of X and Y co-ordinate of these values lie between 0 and 1. velocity and accelerations associated with it. But this trajectory is very useful for analysis of DMP system. Following diagram shows the trajectory path in 2D plane.

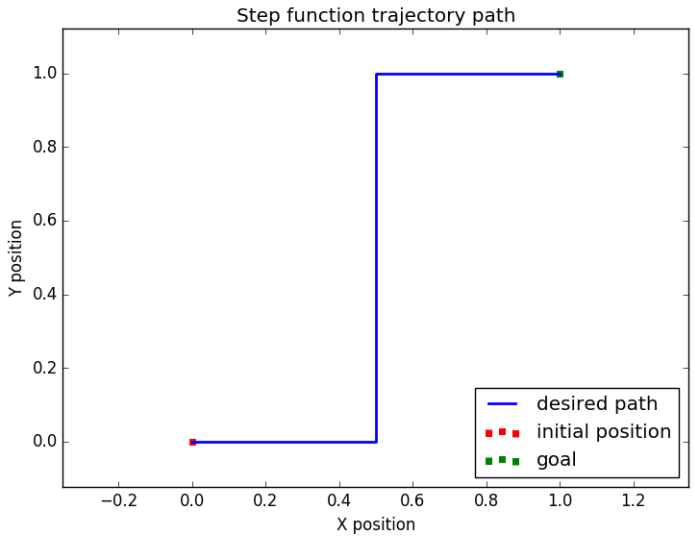

Figure 3.3: Step function trajectory path

Effect of number of basis functions

Number of basis functions used in the DMP affects the approximation of non-linear forcing function which modifies the shape of motion. More the number of basis functions results into better approximation of forcing function. Better approximation of forcing function means the shape of the trajectory generated by DMP will be more similar to the original trajectory which was used for learning.

Figure 3.4 illustrates the effect of different number of basis functions on learning the trajectory. It can be observed that the increasing the number of basis function results into better approximation of shape the original trajectory.

The error in mimicking the trajectory is summarized in table 3.0.2. From data, it can be concluded that the trajectories containing high frequency components can be better approximated with high number of basis functions.

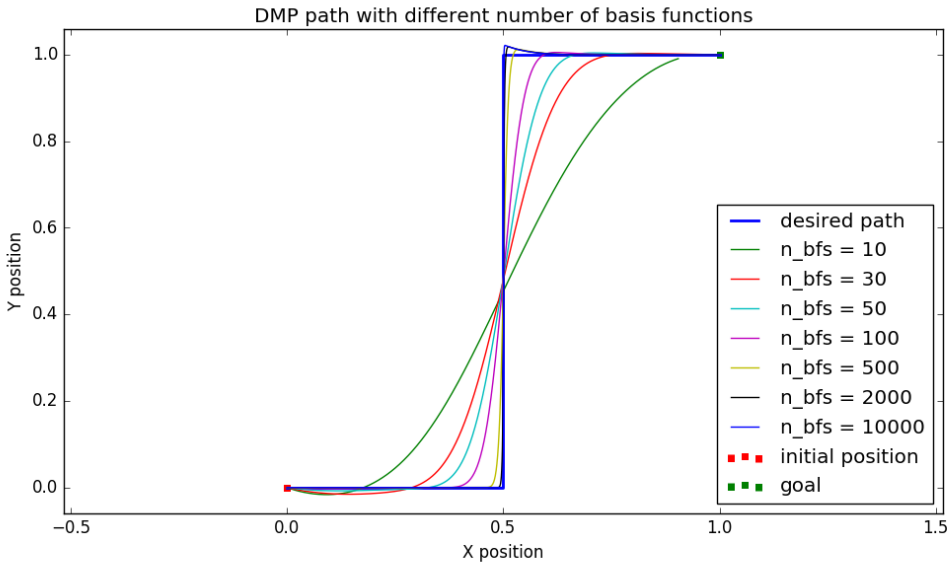

Figure 3.4: Effect of number of basis functions (n_bfs)

Number of basis functions	10	30	50	100	500	2000	10000
Error	0.093	0.038	0.021	0.009	0.002	0.001	0.001

Table 3.1: Error in mimicking the trajectory

Number of basis functions used while learning the DMP also affects the learning of noise. While demonstrating the trajectory, it is possible that high frequency noise is recorded especially because of vibrations and shaking of hands of teacher. If the number of basis function used is very high, high frequency noise is also learned which is not desired. Low number of basis functions result into a smoother trajectory filtering out the noise.

Use of high number of basis functions also increases the number of computations at run-time as well as at the time of learning the DMP. As the learning algorithm used is linear and DMPs are learned off-line, increase in learning time does not affect much. But if it is required to update the motion command at high frequency while executing the DMP, using high number of basis functions is undesirable.

Effect of time step size

Motion commands and trajectory generated by DMP framework are in discrete time. Euler's integration method is used to solve the differential equations to obtain the kinematic state at each time step. Hence the choice of step size affects the error in generated trajectory. If the step size used is not small enough, DMP can generate the trajectories those are potentially undesirable and dangerous to be executed by a robot. Large step sizes result in oscillations in the trajectories. Figure 3.5 illustrates the effect of various time step sizes on the trajectories generated by the DMP framework.

If the step size is not small enough, acceleration at a specific point on the trajectory is not updated for long enough time, so that the generated trajectory overshoots and deviates from the desired trajectory. On next time step, a counter acceleration is applied in order to bring the trajectory close to desired trajectory which again overshoot in opposite direction as previous time step. This intuitive explanation signifies the need of small enough time step for stable behavior of DMP.

Time step size	0.05	0.01	0.005	0.001
Error	0.062	0.017	0.011	0.007

Table 3.2: Error in mimicking the trajectory

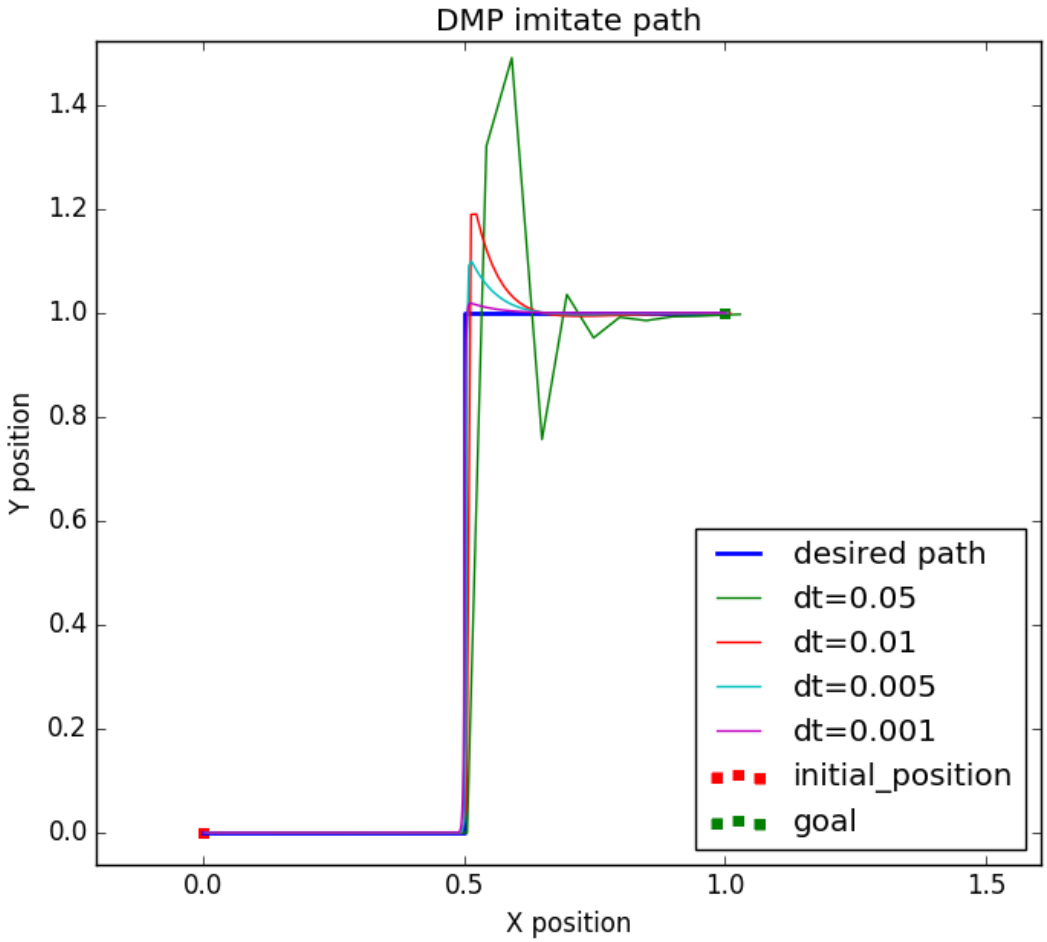

Figure 3.5: Effect of time step size (dt)

If the DMP is used for generating instantaneous motion command and feedback form the robot and environment (e.g. position feedback from robot or obstacle from environment) is coupled with DMP, then it is desirable to keep the time step in the order of 10^{-3} to avoid large overshoots.

Effect of time scaling factor *tau*

Time scaling factor (*tau*) is used for modifying the speed of execution of the DMP. It a term which scales the acceleration term produced by transformation system in DMP, which in turn results in scaling of kinematic state at the next step. If the value of *tau* is set to 1, then the execution speed of DMP is same as the demonstration.

If the large value of *tau* is used, undesirable oscillations can be observed in robot trajectories. The reason behind these oscillations is same as the large time step. High value of *tau* makes acceleration at certain time step very large making trajectory to overshoot from desired path. On next time steps, acceleration in opposite direction is generated which is again scaled by *tau* and hence trajectory overshoots in other direction. The effect of large *tau* can be nullified up-to certain extent by choosing sufficiently small step size.

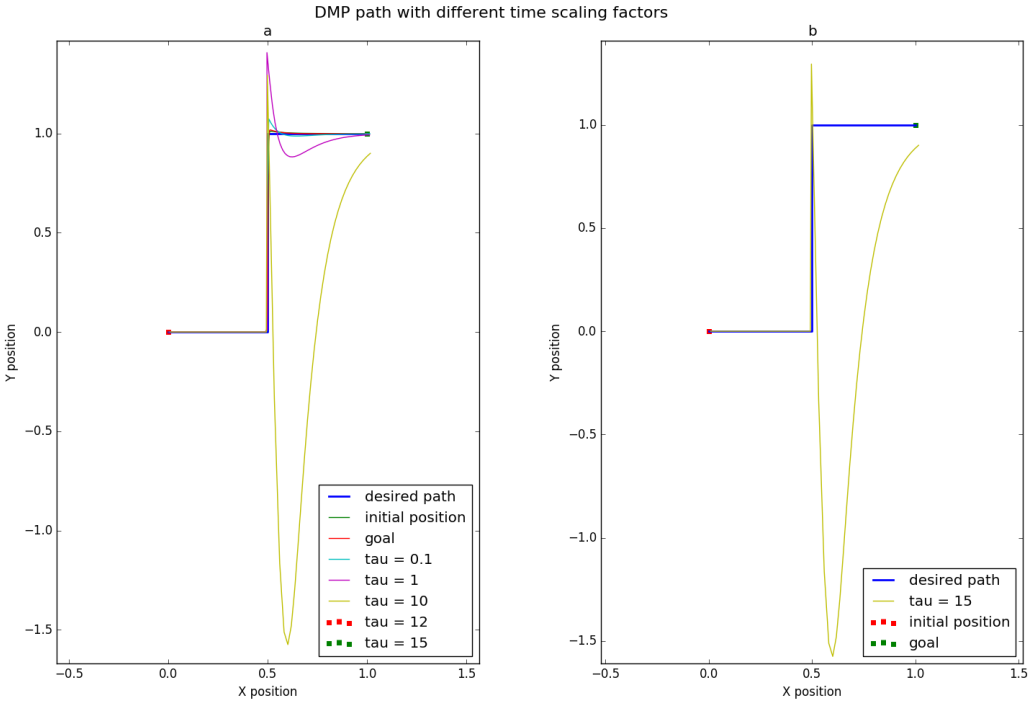

Figure 3.6: Effect of time scaling factor *tau*

Figure 3.6 a illustrates the effect of different values of *tau* on DMP. It should

be noted that value of tau less than 1, does not affect the stability of DMP. Figure 3.6 b shows the instable behavior of DMP due to large value of tau.

Time scaling factor (tau)	0.1	1	10	12	15
Error	0.007	0.007	0.010	0.036	0.292

Table 3.3: Error in mimicking the trajectory

3.0.3 Learning the Motion Primitive from Demonstrated Trajectories

Learning motion primitive implies learning the weights w_i in the eq. 3.3. Above system presented in [7] is linear in the weights w_i. So variety of learning algorithms can be used. [7] uses locally weighted regression to learn the weights.

Desired behavior that should be exhibited by the system is presented as a tuple ($y_{demo}(t)$, $\dot{y}_{demo}(t)$, $\ddot{y}_{demo}(t)$) representing position, velocity and acceleration respectively where $t = [1, 2, 3, 4, ..., p]$. Parameter g is goal hence, $g = y_{demo}(t = p)$ and $y_0 = y_{demo}(t = 0)$. Parameter τ is temporal scaling factor which needs to adjusted for achieving desired time scaling in the motion. In order to use LWR for estimating w_i, eq. (3.8) can be rearranged to generate function approximation problem as,

$$f = \ddot{y} - \alpha_z(\beta_z(g - y) - \dot{y}) \tag{3.11}$$

Substituting the information from the demonstrated trajectory in the left-hand side of this equation, we obtain,

$$f_{target} = \ddot{y}_{demo} - \alpha_z(\beta_z(g - y_{demo}) - \tau\dot{y}_{demo}) \tag{3.12}$$

While learning the motion primitives, time scaling factor τ was set to 1 in all the experiments.

As we have the values f_{target}, we can perform a supervised learning to find a best fit for the function represented by f_{target}.

Locally weighted regression finds for each kernel function ψ_i in f, the corresponding w_i, which minimizes the locally weighted quadratic error criterion,

$$J_i = \sum_{t=1}^{p} \psi_i(t)(f_{target}(t) - w_i\xi(t))^2 \tag{3.13}$$

where $\xi_i = x(t)(g - y_0)$ for the discrete system and $\xi_i = r$ for rhythmic system. Solution to above linear regression problem is,

$$w_i = \frac{s^T \Gamma_i f_{target}}{s^T \Gamma_i s} \tag{3.14}$$

where,

$$s = \begin{pmatrix} \xi(1) \\ \xi(2) \\ \vdots \\ \xi(p) \end{pmatrix} \quad \Gamma = \begin{pmatrix} \psi_i(1) & & & 0 \\ & \psi_i(2) & & \\ & & \ddots & \\ 0 & & & \psi_i(p) \end{pmatrix} \quad f_{target} = \begin{pmatrix} f_{target}(1) \\ f_{target}(2) \\ \vdots \\ f_{target}(p) \end{pmatrix}$$

3.0.4 Inverse Kinematics Solver and Trajectory Controller

In order to execute the task-space trajectory generated by DMP framework, Cartesian velocities are needed to be converted to the joint velocities with the help of inverse kinematic solver. Robot arm used in the experiments (KUKA YouBot arm) has five degrees of freedom. Five degrees of freedom are not sufficient to carry out 6 degrees of freedom cartesian motion. Due to this design deficiency, manipulator is always in the singular configuration. It is well-known that when a manipulator is at-or is in the neighborhood of a singular configuration, severe restrictions may occur on its motion. To overcome this situation, *weighted damped least square pseudo inverse method* for computing joint velocities was chosen. This method allows us to loose the constraints on individual degrees of freedom in task-space. Which allowed us to ignore velocity constraints on 3 rotational degrees of freedom. This is called user-defined accuracy method to control the manipulator. [2]

The relation between joint space and task space velocity can be given by,

$$v = J(\theta)\dot{\theta} \tag{3.15}$$

$$\dot{\theta} = J(\theta)^{-1}v \tag{3.16}$$

Where,
$J(\theta)$ is jacobian of manipulator,
v is task space velocity vector,
$\dot{\theta}$ is joint space velocity vector.

When manipulator is near a singularity, sigma large joint velocities may occur or degenerate directions may exist where end-effector velocity is not feasible[2]. To overcome this situation, damped least square method can be used where a degraded solution is generated near the singularities proposed in [23, 17].

$$J^T(\theta)v = (J^T(\theta)J(\theta) + \lambda^2 I)\dot{\theta} \tag{3.17}$$

Where,
$\lambda > 0$ is the damping factor, and
I is identity matrix.

Solution to above problem can be given by,

$$\dot{\theta} = (J^T(\theta)J(\theta) + \lambda^2 I)^{-1}J^T(\theta)v \tag{3.18}$$

It should be noted that, if the value of λ in eq. 3.18 is set to 0, eq. 3.18 reduces to eq. 3.15.

The effect of value of lambda on joint velocities can be analyzed in detail by singular value decomposition of the Jacobian matrix and that is,

$$J = \sum_{i=1}^{6} \sigma_i u_i \mathbf{v}_i^T \tag{3.19}$$

Using singular value decomposition, equation 3.18 can be re-written as,

$$\dot{\theta} = \sum_{i=1}^{6} \frac{\sigma_i}{\sigma_i^2 + \lambda^2} \sigma_i u_i \mathbf{v}_i^T v \tag{3.20}$$

It can be observed in above equation that if $\sigma \gg \lambda$, then λ has practically no effect on joint velocities. But if σ is close to zero, then the joint velocities are greatly affected by value of λ.

Since the arm has only five degrees of freedom, it is always in singular configuration and hence it is not possible to execute the 6 degrees of motion at all the time instances. This situation can be overcome by ignoring motion in 3 rotational degrees of freedom with the help of the method called *weighted damped least square pseudo inverse*. In this method,

3.1 Setup

3.2 Experimental Design

4

Solution

During this research and development project, a system was implemented which is able to record human demonstrations in terms of kinematic trajectory, learn the motion in terms of dynamic motion primitive, reproduce the motion using dynamic motion primitive and execute this motion on manipulator.

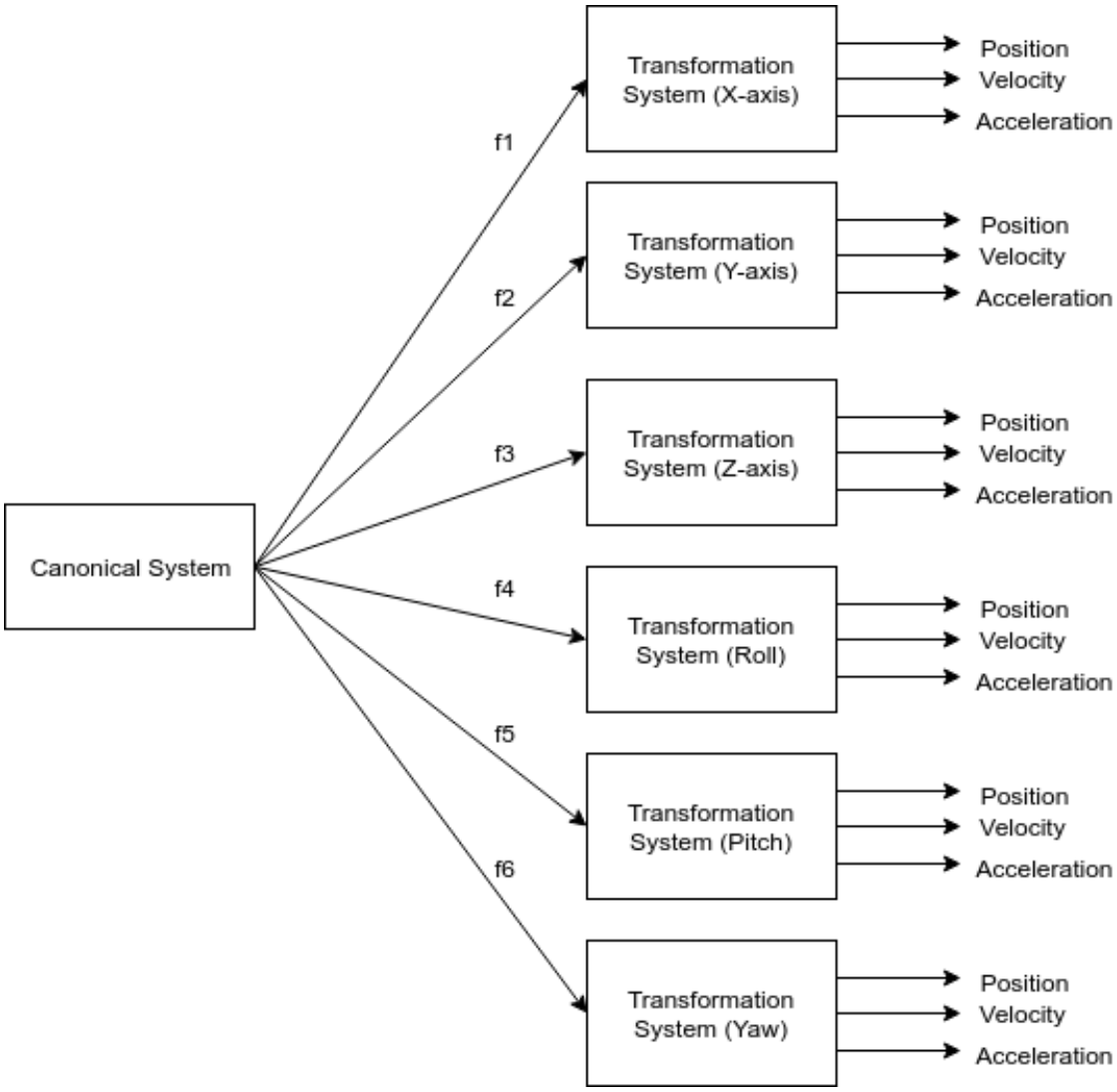

Figure 4.1: 6D DMP framework used during experimentation

4.1 Proposed algorithm

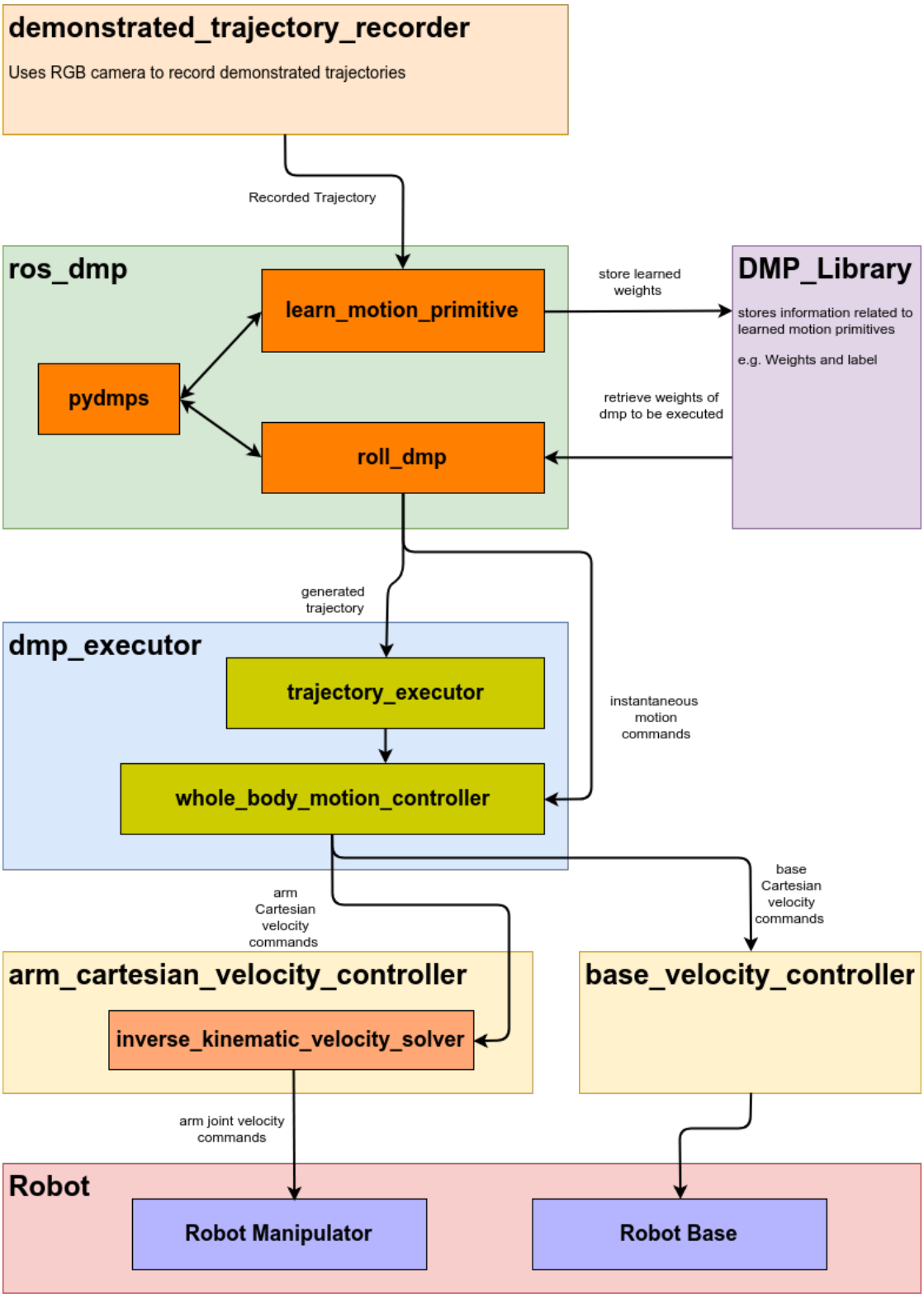

Figure 4.2: Dynamic Movement Primitive framework

4.1.1 Whole Body Motion Control

For the mobile manipulators like KUKA YouBot and Toyota HSR, it is always desirable to have a whole body control functionality i.e. executing motion using mobile base as well as manipulator in order to achieve a objective (e.g. attaining a particular end effector pose in global reference frame or executing a trajectory represented in global reference frame). Commonly used sampling based algorithms can be used to achieve this goal by providing model of environment as well as robot. Dynamic programming based approaches and optimization based motion planning can also solve the problem of whole body motion control, but they heavily suffer from the increased dimensionality of the problem. Prominent drawbacks of such methods are described below.

While conducting the initial experiments on DMP framework with KUKA YouBot, we came across the serious problem of limited manipulation functionality of 5 degrees of freedom manipulator. Numerous trajectories demonstrated were not executable by manipulator alone. This triggered the idea of incorporating base movement to execute the manipulator motion.

4.2 Implementation details

In order to use dynamic motion primitives in robotics, one has to demonstrate the motion to robot, learn the control policy using DMPs, reproduce the motion policy or trajectory and then execute the motion with robot. In this research and development project, motion trajectory is demonstrated using visual demonstrations with the help of arUco marker board. Since the trajectory is learned and reproduced in task space, it is mapped to joint space using kinematic solver and then executed on the robot. Above figure shows the architecture of the software implementation. Necessary components for above mentioned steps are as follows,

4.2.1 Demonstrated Trajectory Recorder

Various methods for demonstrating trajectories are practiced by researchers mainly :

- Teleoperation : A demonstration technique in which the teacher operates the robot learner platform and the robots sensors record the execution. [1]

- Shadowing : A demonstration technique in which the robot learner records the execution using its own sensors while attempting to match or mimic the teacher motion as the teacher executes the task.

- Sensors on teacher : An imitation technique in which sensors located on the executing body are used to record the teacher execution.

- External observation : An imitation technique in which sensors external to the executing body are used to record the execution.

In this project, *external observation* method was used to demonstrate trajectories. Teacher demonstrated trajectories by moving arUco marker board on desired path. A computer vision system ensured the recording of poses of arUco marker board in robot *base_link* frame, at constant rate.

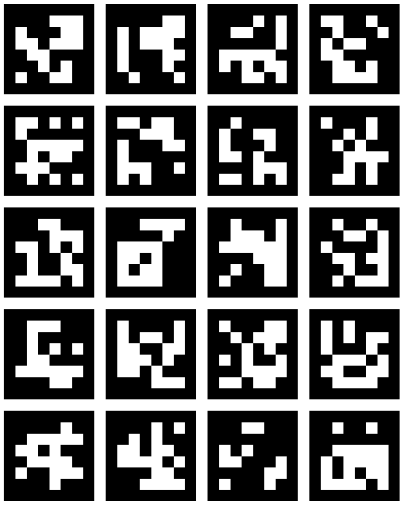

Figure 4.3: arUco marker board used for demonstrations

A RGB camera mounted on the robot (mounted on arm in case of KUKA YouBot and mounted on head in case of Toyota HSR) constantly captures images of the arUco marker board which is being moved by teacher for demonstration. Captured images are processed for estimating boards position in camera optical frame. A already available library in opencv is used to estimate the board's 6D pose. Then the estimated pose is transformed to robot's base_link using transforms published by robot_state_publisher.

4.2.2 pydmps

During this research and development project, the DMP framework described by eq. 3.1, 3.2 and 3.5 implemented in open-source project *pydmps*[4], is used. The original implementation cannot be used for robot control because of following reasons:

-

5

Evaluation

Implementation and measurements.

6

Results

6.1 Use case 1

Describe results and analyse them

6.2 Use case 2

6.3 Use case 3

7

Conclusions

7.1 Contributions

7.2 Lessons learned

7.3 Future work

A

Design Details

Your first appendix

B

Parameters

Your second chapter appendix

References

[1] Brenna D Argall, Sonia Chernova, Manuela Veloso, and Brett Browning. A survey of robot learning from demonstration. *Robotics and autonomous systems*, 57(5):469–483, 2009.

[2] Stefano Chiaverini, Bruno Siciliano, and Olav Egeland. Review of the damped least-squares inverse kinematics with experiments on an industrial robot manipulator. *IEEE Transactions on control systems technology*, 2(2):123–134, 1994.

[3] Marc Peter Deisenroth, Gerhard Neumann, Jan Peters, et al. A survey on policy search for robotics. *Foundations and Trends® in Robotics*, 2(1–2):1–142, 2013.

[4] Travis DeWolf. pydmps. `https://github.com/studywolf/pydmps`, 2017.

[5] Heiko Hoffmann, Peter Pastor, Dae-Hyung Park, and Stefan Schaal. Biologically-inspired dynamical systems for movement generation: automatic real-time goal adaptation and obstacle avoidance. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 2587–2592. IEEE, 2009.

[6] Auke J Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning attractor landscapes for learning motor primitives. In *Advances in neural information processing systems*, pages 1547–1554, 2003.

[7] Auke Jan Ijspeert, Jun Nakanishi, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Dynamical movement primitives: learning attractor models for motor behaviors. *Neural computation*, 25(2):328–373, 2013.

[8] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Robotics and Automation, 2002. Proceedings. ICRA'02. IEEE International Conference on*, volume 2, pages 1398–1403. IEEE, 2002.

[9] Martin Karlsson, Anders Robertsson, and Rolf Johansson. Autonomous interpretation of demonstrations for modification of dynamical movement primitives. In *Robotics and Automation (ICRA), 2017 IEEE International Conference on*, pages 316–321. IEEE, 2017.

[10] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. In *Autonomous robot vehicles*, pages 396–404. Springer, 1986.

[11] Zachary Kingston, Mark Moll, and Lydia E Kavraki. Sampling-based methods for motion planning with constraints. *Annual Review of Control, Robotics, and Autonomous Systems*, 1:159–185, 2018.

[12] Jens Kober, J Andrew Bagnell, and Jan Peters. Reinforcement learning in robotics: A survey. *The International Journal of Robotics Research*, 32(11):1238–1274, 2013.

[13] Steven M LaValle. From dynamic programming to rrts: Algorithmic design of feasible trajectories. In *Control Problems in Robotics*, pages 19–37. Springer, 2003.

[14] Bokman Lim, Syungkwon Ra, and Frank C Park. Movement primitives, principal component analysis, and the efficient generation of natural motions. In *Robotics and Automation, 2005. ICRA 2005. Proceedings of the 2005 IEEE International Conference on*, pages 4630–4635. IEEE, 2005.

[15] Rudolf Lioutikov, Oliver Kroemer, Guilherme Maeda, and Jan Peters. Learning manipulation by sequencing motor primitives with a two-armed robot. In *Intelligent Autonomous Systems 13*, pages 1601–1611. Springer, 2016.

[16] Franziska Meier and Stefan Schaal. A probabilistic representation for dynamic movement primitives. *arXiv preprint arXiv:1612.05932*, 2016.

[17] Yoshihiko Nakamura and Hideo Hanafusa. Inverse kinematic solutions with singularity robustness for robot manipulator control. *Journal of dynamic systems, measurement, and control*, 108(3):163–171, 1986.

[18] Bojan Nemec and Aleš Ude. Action sequencing using dynamic movement primitives. *Robotica*, 30(5):837–846, 2012.

[19] Alexandros Paraschos, Christian Daniel, Jan R Peters, and Gerhard Neumann. Probabilistic movement primitives. In *Advances in neural information processing systems*, pages 2616–2624, 2013.

[20] Dae-Hyung Park, Heiko Hoffmann, Peter Pastor, and Stefan Schaal. Movement reproduction and obstacle avoidance with dynamic movement primitives and potential fields. In *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*, pages 91–98. IEEE, 2008.

[21] Peter Pastor, Heiko Hoffmann, Tamim Asfour, and Stefan Schaal. Learning and generalization of motor skills by learning from demonstration. In *Robotics and Automation, 2009. ICRA'09. IEEE International Conference on*, pages 763–768. IEEE, 2009.

[22] Stefan Schaal. Dynamic movement primitives-a framework for motor control in humans and humanoid robotics. In *Adaptive motion of animals and machines*, pages 261–280. Springer, 2006.

[23] Charles W Wampler. Manipulator inverse kinematic solutions based on vector formulations and damped least-squares methods. *IEEE Transactions on Systems, Man, and Cybernetics*, 16(1):93–101, 1986.