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

15 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

To be able to plan and execute the motion is one of the primary requirements of an autonomous robot to accomplish a given task. Over the time, numerous solutions were presented for motion planning, which are good enough for solving the problem of motion planning for practical problems. In this work, we present a learning from demonstration framework, based on the dynamic motion primitives, which allows us to program the robot by visual demonstration of the motion. Biologically inspired dynamic motion primitives learn the control policy behind the demonstrated motion in terms of attractor dynamics of a non-linear second order differential equation. We evaluate the performance of dynamic motion primitives for generalization of numerous demonstrated trajectories on two five degrees of freedom manipulators. For boosting the manipulation capabilities of these manipulators, we propose the idea of using the mobile base motion along with the manipulator motion which lead to whole body motion control framework for manipulation. Proposed whole body motion control framework was integrated in current software solution for manipulation of Toyota HSR robot.

Contents

List of Figures

List of Tables

1

Introduction

The ease with which the humans and animals accomplish extremely complex motions has influenced the research in robotic motion planning and control at various levels, right from the motor control to high the 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 [39]. One key observation can be made that the biological motions are consisted of *motion primitives* (basic units of motion) perfected through experience over time[62]. 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 [33] and [10] 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 equations representing mass-spring damped system called *Dynamic Movement Primitives (DMP)* is particularly famous and this work uses the same.

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

attractor space of nonlinear dynamic equations.

An 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 which is hard to obtain. Need of skills and experience required for motion execution is replaced by the accurate model of robot and environment.

In this project, a learning from demonstration framework using dynamic movement primitives was implemented on KUKA YouBot mobile manipulator and Toyota HSR. Various experiments were performed to prove the usability of DMPs in RoboCup@Work and RoboCup@Home scenarios and identify the need for knowledge base for DMPs.

While conducting the experiments, limitations on executing the trajectories generated by DMPs were revealed which triggered the idea of combining mobile base motion with manipulator motion for tracking the trajectories which lead to the development of whole body motion control framework. Whole body motion significantly enhanced the manipulation capabilities of both the robots.

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 planning is a decades old problem 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 uses the gradient of a potential that guides a robot to its goal [30]. 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."[31] But complexity of these algorithm makes it hard to scale them for higher dimensions and adopt them for constraints [31].

Dynamic programming is one the algorithms which are proven to be good for

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 soften 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[31]. They also suffer from the increased dimensionality of the problem.

Sampling-based algorithms take a very different approach. They randomly sample the 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."[31] 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[31].

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) [37]. 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 of the robot and validating it using the predefined constraints. Not every solution found by RRT is feasible and optimal which makes method incomplete and non-optimal. But this method is quiet effective 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.

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[10].

Deisenroth et al.[10] 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[10].

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[10].

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 [33], 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." [54]

2.4.1 Motion Primitives Using Principle Component Analysis

In [39], 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 work, a task parser uses the description of the task to generate a sequence of the movements. Then the movement compiler chooses appropriate motion primitives to perform the 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 [54] 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 (DMP)

In this research project, dynamic motion primitives are used to learn the control policy behind a trajectory. So let's see the evolution of DMP in detail.

Ijspeert et. al. in [26] 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.

To meet above requirements, 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 Gaussian functions 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 Gaussian functions are learned by using locally weighted regression. This work was the necessary foundation of the theory of dynamic motion primitives.

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

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

and ψ_i is given by,

$$\psi = exp(-\frac{1}{2\sigma_i^2}(\tilde{x} - c_i^2)) \tag{2.3}$$

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

$$\dot{z} = \alpha_v(\beta_v(g - x) - v) \tag{2.4}$$

$$\dot{x} = v \tag{2.5}$$

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 [22] 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 [26]. Preliminary work in [26] 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 [62] 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 [56] 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 [40], 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 [51]. 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 [55] 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 [45]. 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.[29] 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 [18] 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 Demonstrated Trajectory Recorder

Over the time various methods for demonstration of the trajectory are developed. These methods mainly include:

- Teleoperation : A demonstration technique in which the teacher operates the robot learner platform and the robots sensors record the execution[3].
- 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[3].
- Sensors on teacher : An imitation technique in which sensors located on the executing body are used to record the teacher execution[3].
- External observation : An imitation technique in which sensors external to the executing body are used to record the execution[3].

3

Methodology

The Dynamic Motion Primitive is essentially a learning from demonstration (*LfD*) approach for robot programming. To use DMP framework for robotic motion planning and control, following components are needed to be implemented,

- A system for acquiring human demonstrations.
- Dynamic motion primitive framework.
- Motion controller.

All the methodologies and techniques which are necessary for implementing above mentioned systems, are explained in this chapter. The architecture and implementation details are discussed in the next chapter of this report.

3.1 Dynamic Motion Primitives

3.1.1 Advantages of Dynamic Movement Primitives

Before diving into the formulation of the dynamic motion primitives, lets summarize the advantages of using them because of which they became the obvious choice for learning control policy.

- 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.
- 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 [55]). Perturbations can be handled robustly due to this reason.
- Re-planning is not needed unless an event causing major disturbance in the environment occurs.

3.1.2 Formulation of Dynamic Movement Primitives

Dynamic motion primitives use second order nonlinear differential equations to model motion. These differential equations essentially represent a damped mass spring system where the attractor landscape of differential equations represent desired kinematic states of the robot. Point attractor and limit cyclic behaviors of the second order nonlinear differential equation are used to learn discrete (point to point) and rhythmic robot motion, respectively. Over the time, various versions of DMPs were presented which were slightly different than one another, modified for specific use case or scenario. The following formalization of DMPs is taken from [23].

A DMP can be represented by the 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 a first order representation of an autonomous non-linear second-order differential equation where $f(x)$ is a non-linear term. Upon solving these equations, we get the state $[\ddot{y}, \dot{y}, y]$ at each time instance. Normally, the Euler's integration method is used to solve this system of equations.

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"[23]. Such a system implements a stable but trivial pattern generator with g as single point attractor [23].

The idea is to use the eqs. 3.1 and 3.2 for encoding the control policy to generate kinematic commands for the robot (joint level motion commands or Cartesian motion commands). The kinematic state $[\ddot{y}, \dot{y}, y]$ obtained by solving these equations at each time step, is used as the kinematic commands (position, velocity and acceleration) for the robot manipulator (angular commands for individual joints or Cartesian commands for end-effector), which will become clear later.

With the term $f = 0$, evolution of system in eqs. 3.1 and 3.2 always be of same nature, i.e. y will eventually become g monotonically and asymptotically, given that system is critically damped. By introducing the term f, the path followed by the system in attractor landscape of differential equation from initial state to the goal state can be modified arbitrarily. This in-turn modifies the trajectory followed by a mobile robot or a robotic manipulator in the task or joint-space. This non-linear forcing function enables the DMP framework to learn almost any

arbitrary motion in end-effector space or joint space. It is essentially a normalized weighted sum of equally spaced Gaussian functions denoted by ψ (eq. 3.4) activated at each time step by the phase variable x. The forcing term $f(x)$ is learned from the demonstrated trajectories (discussed in detail in section 3.1.3). It should be noted that the forcing term f is modified by the separation between the 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 the *transformation system*, as they transform the current state of the system to the next state.

Equation 3.5 is called the *canonical system*. x acts as a phase variable modifying forcing term $f(x)$ and hence modifying the shape of the trajectory. This removes the explicit time dependency of the f, 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, which makes $f(x) = 0$ at the end of the motion, ensuring convergence of y 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" [23].

The 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 terms. The term c_i in eq. 3.4 is the center of i^{th} Gaussian function.

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 a learn forcing term $f(x)$ which is periodic itself and makes the system of equations 3.1 and 3.2 exhibit limit cyclic behavior. This can be achieved by choosing the 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:

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

At this point it becomes clear that the point attractor and limit cyclic 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.

In this 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 the original DMP formulation. Unlike the original formulation, the term τ^2 is now multiplied on right side of equation and the first derivative term (velocity) $\dot{y}$ is not multiplied by τ. Still the τ will affect the speed of evolution of the differential equation system as it scales the acceleration term.

Figure 3.1 taken from [23] shows the composition of the DMP framework. The *Coupling Terms* in the figure consists of feedback from the sensors, obstacle coupling terms, error in the execution speed of the DMPs, etc. The 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 the transformation system. The kinematic motion commands are then executed by robot controller (e.g. velocity controllers, torque controllers, etc.)

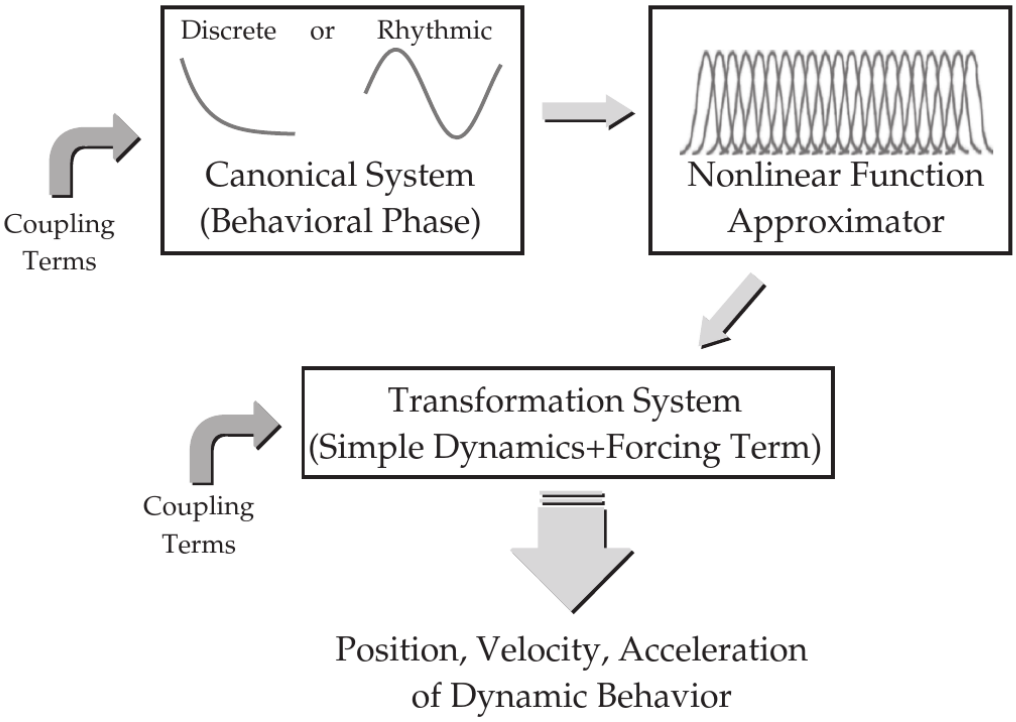

Figure 3.1: Dynamic Movement Primitive framework [23]

Discussion

To have an intuitive understanding of the DMP framework, imagine a critically damped mass-spring system, in which a point mass is attached to a spring which is fixed at one end. The length of the spring lie along the Y-axis, making the system strictly one dimensional. Initially, the system is at equilibrium point i.e. spring is not stretched and point mass is at position g, the equilibrium point. If the mass is stretched to position y and released, it travels along the Y-axis towards the equilibrium point of the system of i.e. till the spring attains its natural length. Since the system is critically damped, we don't observe any oscillations around the equilibrium point. As mentioned earlier, this system can be mathematically expressed by 3.1 and 3.2 when f is 0. These equations state the position, velocity and acceleration of point mass at each point in time. Same set of equations which governs the motion of mass-spring system along the Y-axis can be used for

encoding manipulator end-effector motion in X-axis, Y-axis and Z-axis, to create a 3-dimensional trajectory. Same set of equations can also encode the motion of single manipulator joint. Use of multiple equations as such, will allow us to encode motion of manipulator in 6 degrees of freedom in task space or the motion of every joint in the joint space. Solving all these equations step by time step, desired kinematic commands can be obtained and passed to robot controllers to be executed. It is also possible to solve these equations till they converge to equilibrium point g, which will give us entire trajectory that the end-effector should follow to attain the goal g (goal g will be a goal vector in this case).

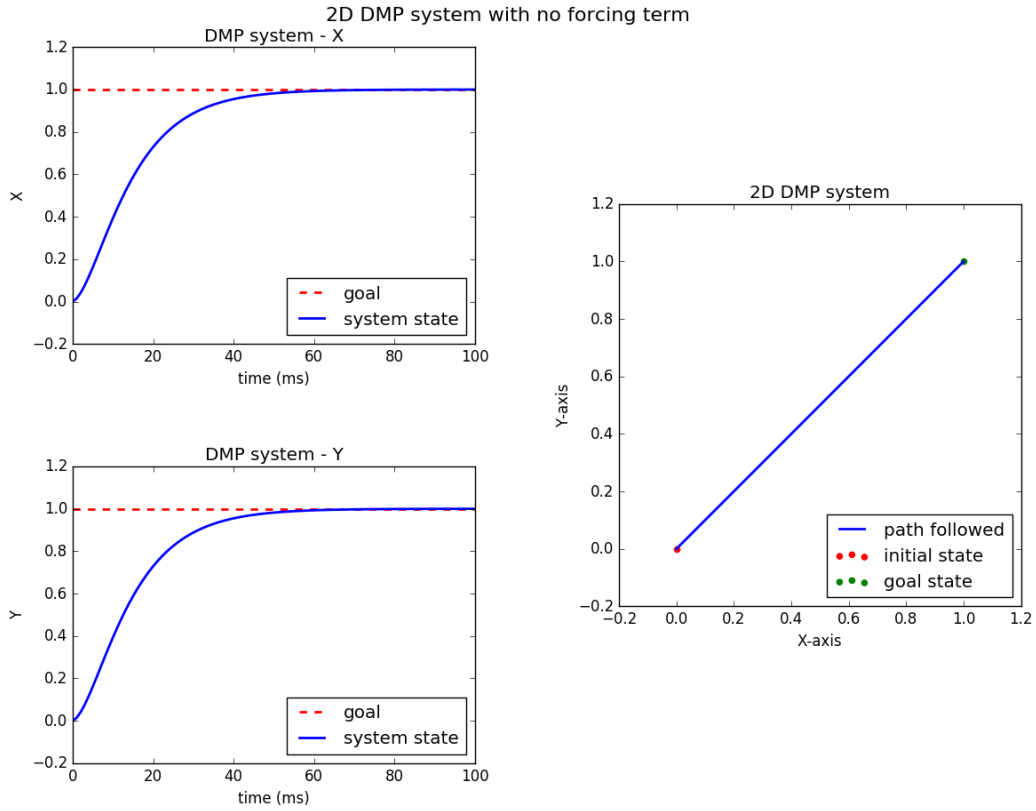

Figure 3.2: 2D DMP with no forcing term

Figure 3.2 shows the path generated in Cartesian space, by 2 degrees-of-freedom DMP with no forcing term. 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 motions generate a straight line motion in 2D space.

But rarely in real world, robots need to follow such straight line path. Numerous robotic tasks demand complex trajectories, and hence we use forcing term f to modify the behavior of trajectory (by adding f to the acceleration at each time step).

Now the question is, do we know the appropriate values of f to add in the acceleration term? No, we don't. But it seems that if we know the values of w's in eq. 3.3, we can estimate the the value of f. Again, do we know the appropriate value of w's? And again, we don't. So lets learn them.

3.1.3 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 [23] is linear in the weights w_i. So variety of learning algorithms can be used. [23] 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 be 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,

$$f = \ddot{y} - \alpha_z(\beta_z(g - y) - \dot{y}) \tag{3.10}$$

While learning the motion primitives, time scaling factor τ is set to 1 to ensure learning the motion primitive with same speed as demonstrated.

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

$$f_{target} = \ddot{y}_{demo} - \alpha_z(\beta_z(g - y_{demo}) - \tau\dot{y}_{demo}) \tag{3.11}$$

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,

$$J_i = \sum_{t=1}^{p} \psi_i(t)(f_{target}(t) - w_i\xi(t))^2 \tag{3.12}$$

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,

$$w = \frac{s^T \Gamma_i f_{target}}{s^T \Gamma_i s} \tag{3.13}$$

where,

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

and w is the weight vector.

Learned weights are stored in the library to be used later.

Discussion

Now that we know, how to learn weights to generate a forcing function that can enable DMP to learn any arbitrary trajectory, we can discuss the significance of the forcing term f. To analyses the roll played by forcing term, lets learn a step function like trajectory in 2D space, shown in the fig. 3.3.

Because of the step function like nature of the trajectory, it is interesting to observe the forcing term f associated with the DMP in y-axis, which is shown in fig. 3.4.

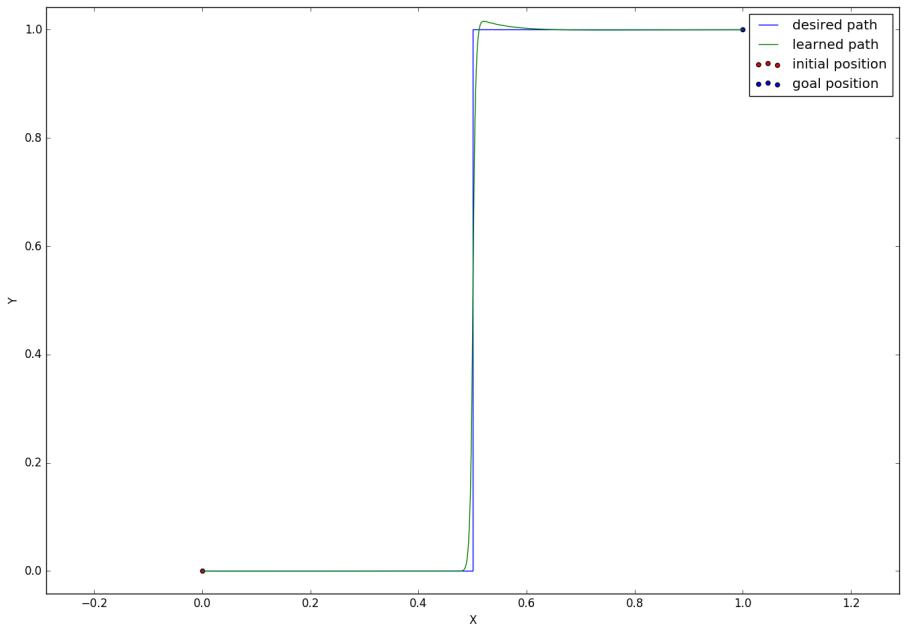

Figure 3.3: Step function trajectory path

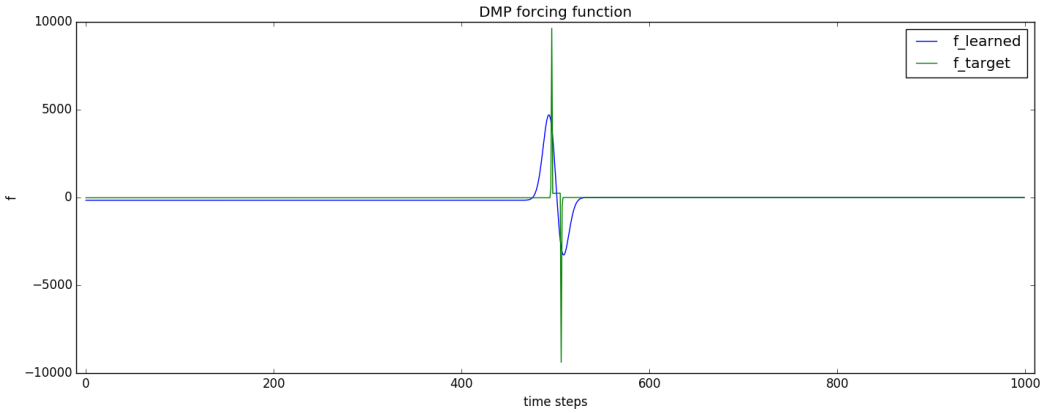

Figure 3.4: Forcing term for y-axis

We can observe spikes in both forcing terms, target forcing term and learned forcing term (a positive spike followed by negative spike). As the f is directly added in the acceleration of DMP, positive spike in forcing term produces massive

acceleration command in order to achieve step value ($y = 1$ in this case). Following negative spike brings acceleration back to 0 to settle the y at value 1.

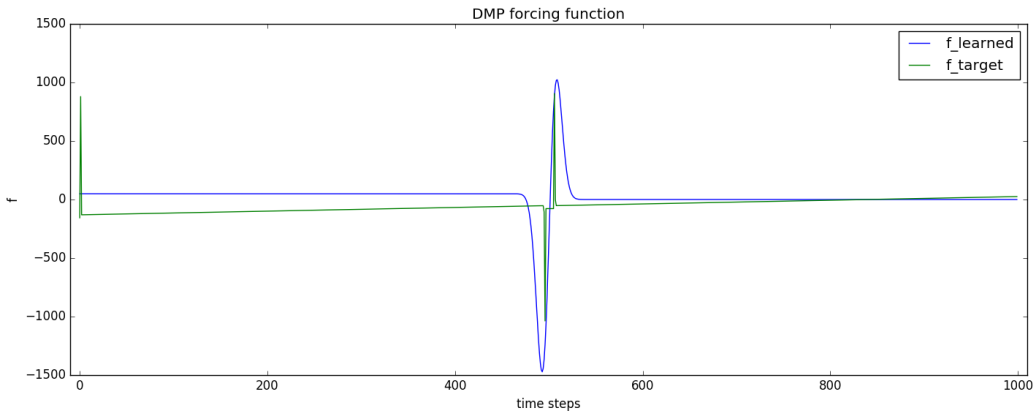

Figure 3.5: Forcing term for x-axis

At the same time, forcing term associated with DMP in x-axis (shown in fig. 3.5), slows x-velocity down for a while to let y settle at $y = 1$, and then again produces f to attain some velocity to continue till $x = 1$.

It should be noted that the learned f is relatively smoother than the target forcing function, which introduces error at the corners of the step function trajectory. The smoothness in the learned forcing function depends on number of basis function used.

Hence in conclusion, DMP learns the shape of trajectories by modifying relative accelerations of individual motion primitives in each degrees of freedom in synchronous fashion. The forcing term associated with each degree of freedom is synchronized with the help of *canonical system*.

In this project, DMPs are used to learn end-effector motion in Cartesian space. Motion in each degree of freedom is used as separate DMP and then superimposed to create 6D motion. Following figure shows the composition of the motion. Canonical system governs the synchronization between all the degrees of freedoms.

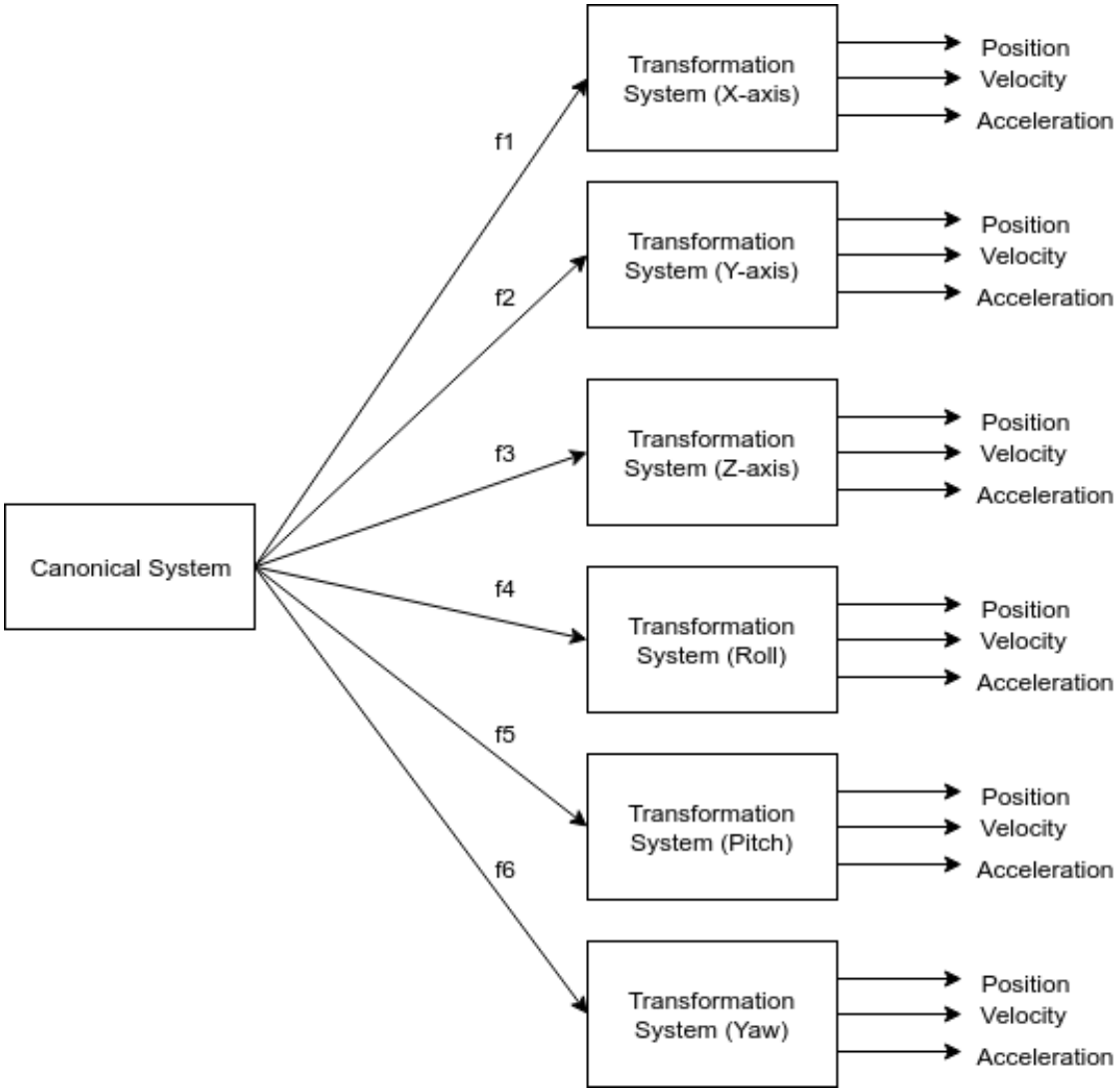

Figure 3.6: 6D DMP framework

3.1.4 Analysis of the effects of the parameters used in DMP

Apart from learning the weights, their are certain parameters in DMP framework which affect the learning and motion generation by DMP. Hence, for better understanding and effective use of DMP, analysis of the effects of these parameters is necessary.

For analyzing the effects of the various parameters used in DMP, a artificially generated step function trajectory, shown in figure 3.7, was learned and the parameters were changed in order to evaluate their effects on the DMP. The evaluation criterion was the closeness between the desired path and the path generated by DMP after learning. The error between the path was calculated a as normalized point-to-point distance between the discrete positions on two paths. More precisely , the error was calculated as follows:

$$error = \frac{1}{N}\sum_{i=1}^{N} min\{\|P_i - Pd_1\|, \|P_i - Pd_2\|, \|P_i - Pd_3\|, ..., \|P_i - Pd_M\|\} \tag{3.14}$$

Where,
P is the path generated by the 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, and hence does not consider the deviation of the trajectory at each time step. It also does not consider the deviation in velocity and acceleration.

Intuitive analysis of other undesired behaviors such as jumps was also done by observing the paths plotted in 2D graphs.

The generated artificial trajectory contains 100 2D poses sampled at equal time intervals of $0.01s$. Values of X and Y coordinate values lie between 0 and 1. Such trajectory cannot be used on a real robot due high values of velocities and accelerations associated with it. But this trajectory is very useful for analysis of

DMP system.

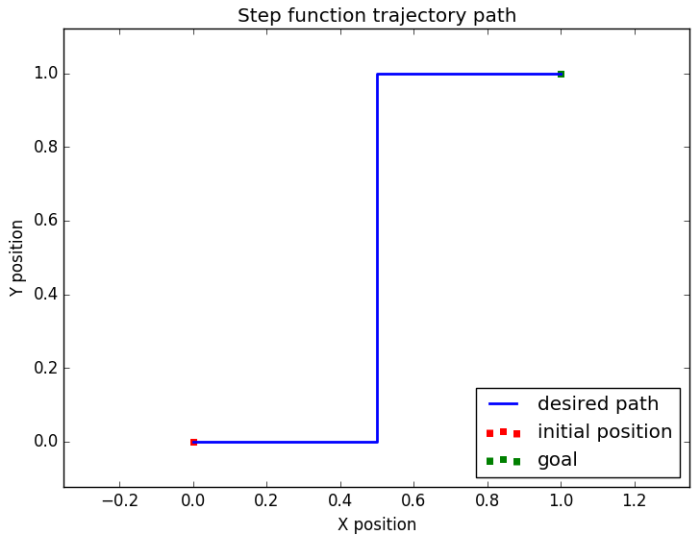

Figure 3.7: Step function trajectory path

Effect of the number of basis functions on the trajectory approximation

The number of basis functions used in a DMP affects the approximation of the non-linear forcing function which modifies the shape of motion: a higher number of basis functions results in a better approximation of the forcing function. A better approximation of the forcing function means that the shape of the trajectory generated by the DMP will be more similar to the original trajectory which was used for learning.

Figure 3.8 illustrates the effect of different numbers of basis functions on learning the step function trajectory. It can be observed that increasing the number of basis function results in a better approximation of the shape of the original trajectory.

The error in mimicking the trajectory is summarized in table 3.1.4. From this data, it can be concluded that the trajectories containing high frequency components can be better approximated with a high number of basis functions.

Fixed parameters :

$\tau = 1$

$dt = 0.001$

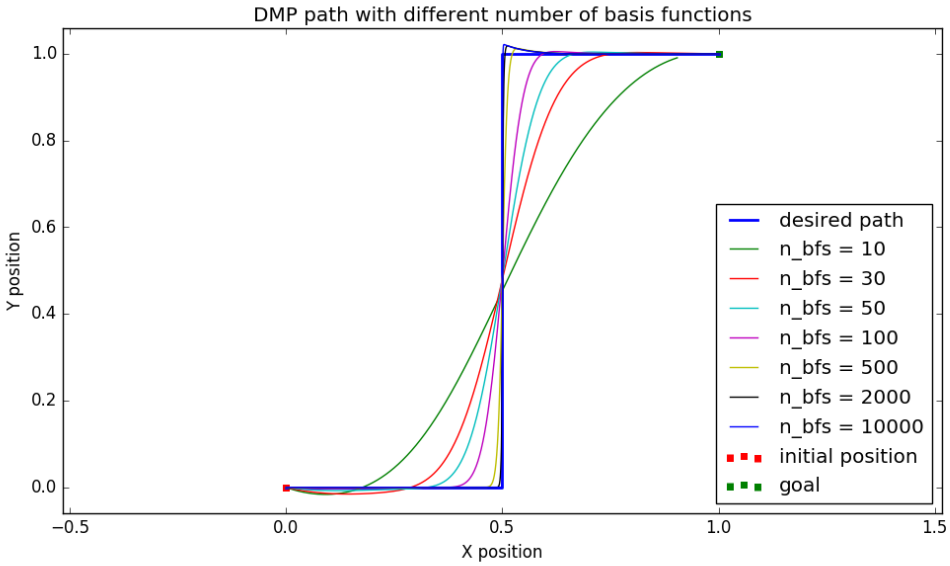

Figure 3.8: Effect of the number of basis functions (n_bfs) on the trajectory approximation

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

The 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 the hands of the teacher. If the number of basis function used is very high, the high frequency noise is also learned which may not be desired; low number of basis functions result into a smoother trajectory filtering out the noise.

The 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, the increase in learning time does not matter much, however, if it is required to update the motion commands at high frequency while executing the DMP, using a high number of basis functions is undesirable.

Effect of the time step size on the trajectory approximation

The motion commands and the trajectory generated by the 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 the step size affects the error in the generated trajectory. If the step size used is not small enough, DMPs can generate trajectories that are potentially undesirable and dangerous to be executed by a robot; in other words, large step sizes result in oscillations in the trajectories. Figure 3.9 illustrates the effect of various time step sizes on the trajectories generated by the DMP framework.

If the step size is not small enough, the 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 an opposite direction from the previous time step. This intuitive explanation signifies the need of a small enough time step for a stable behavior of DMPs.

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

Fixed parameters :
$n\_bfs = 2000$
$\tau = 1$

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

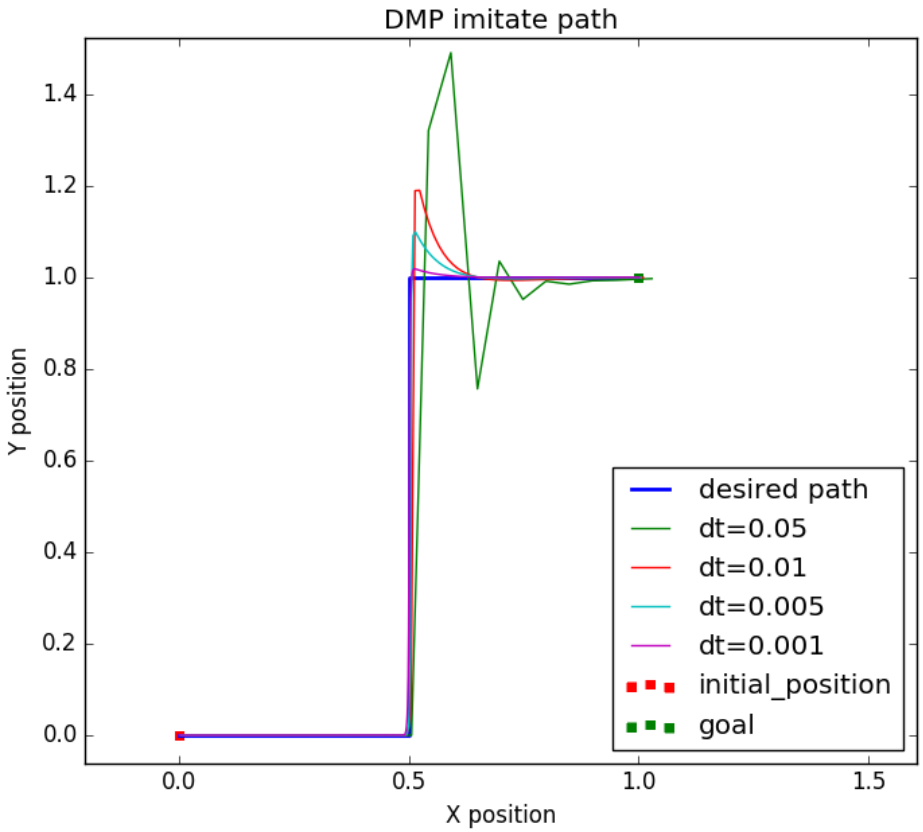

Figure 3.9: Effect of the time step size (dt) on the trajectory approximation

Effect of the time scaling factor τ on the trajectory approximation

The time scaling factor (τ) is used for modifying the speed of the execution of a DMP; in other words, it is a term which scales the acceleration term produced by the transformation system, which in turn results in scaling of the kinematic state at the next time step. If the value of τ is set to 1, then the execution speed of DMP is same as that of the demonstration.

If a large value of τ is used, undesirable oscillations can be observed in the robot trajectories. The reason behind these oscillations is same as that of the large time step; high values of τ cause large acceleration at certain time step causing the trajectory to overshoot from the desired path. On next time steps, acceleration in opposite direction is generated by the dynamics of the *transformation system*, which is again scaled by τ and hence trajectory overshoots in the other direction. The effect of large a τ can be nullified to a certain extent by choosing a sufficiently small step size.

Figure 3.10 a illustrates the effect of different values of τ on DMP. It should be noted that a value of τ less than 1, does not affect the stability of DMP. Figure 3.10 b shows an unstable behavior of the DMP due to a large value of τ.

Fixed parameters :
$dt = 0.001$
$\tau = 1$

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

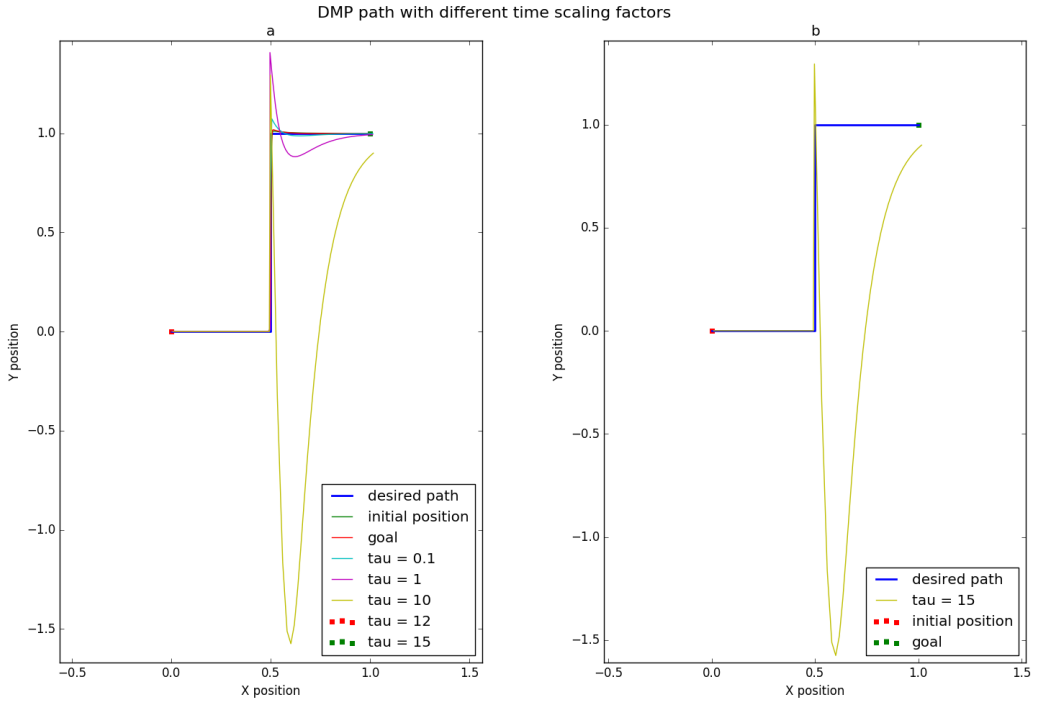

Figure 3.10: Effect of the time scaling factor τ on the trajectory approximation

3.1.5 Demonstration of Trajectories

In this project, *external observation* method was used to demonstrate the trajectories. In this method, the teacher moves arUco marker board in along the path to be demonstrated. A computer vision system ensures the recording of poses of the arUco marker board in robot *base_link* frame, at constant rate.

A *RGB* camera captures the images of demonstration at equal time interval. Each image is processed for estimating 6D pose of the arUco marker board in camera optical frame.

To estimate the pose of the arUco marker board, *detectMarkers()* and *estimatePoseBoard()* methods in class *aruco* provided by *opencv* library are used. Implementation of these functions are out of the scope of this project.

These functions internally use *solvePnP()* method which estimates the pose of the local frame of reference in camera optical frame using correspondence between

the actual points in the local co-ordinate frame of reference and their projection on the image plane.

solvePnP() function finds such a pose that minimizes re-projection error, that is the sum of squared distances between the observed projections in image and the projected points. For minimization of the error function uses Levenberg-Marquardt optimization method.

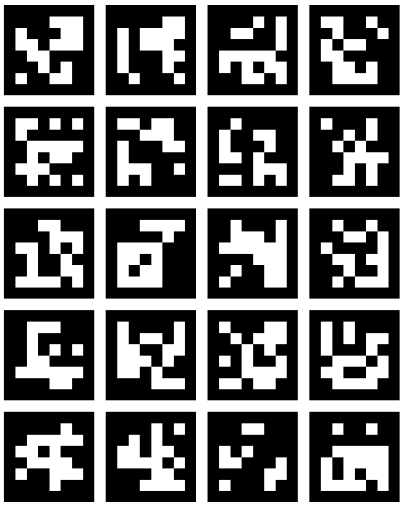

Figure 3.11: arUco marker board used for demonstrations

A arUco marker board was used instead of using single arUco marker becuase:

1. multiple arUco markers on a single board provide more data for pose estimation which results in increased accuracy.
2. even if the board is occluded or partially out of field of the view of the camera, it possible to estimate the pose.
3. due to above reason, it is possible to record poses at almost every discrete time instance, making trajectory continuous in time.

This method also has some drawbacks:

1. demonstrating rotational motion of end effector is very difficult because it non-intuitive for teacher to rotate the board as if it a end-effector.

2. while demonstrating the motion, high frequency noise is introduced in the trajectory due shaking of the hands off the teacher.

3.1.6 Inverse Kinematics Solver and Trajectory Controller

In order to execute the task-space trajectory generated by the DMP framework, Cartesian velocities need to be mapped to joint velocities with the help of inverse kinematic solver. The robot arms used in the experiments (a KUKA YouBot arm and Toyota HSR arm) has five degrees of freedom. Five degrees of freedom are not sufficient to carry out a 6 degrees of freedom Cartesian motion at all the time instances. 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, the *weighted damped least square pseudo inverse method* for computing joint velocities was chosen. This method allows us to loose the constraints on the individual degrees of freedom in task-space, which allowed us to ignore the velocity constraints on the 3 rotational degrees of freedom. This is called a user-defined accuracy method to control the manipulator. [6]

The relation between the joint space and task space velocities 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 matrix of manipulator configuration,
v is the task space velocity vector,
$\dot{\theta}$ is the joint space velocity vector.

When a manipulator is near a singularity, Jacobian matrix of the manipulator becomes ill-conditioned. Due to this, "large joint velocities may occur or degenerate directions may exist where end-effector velocity is not feasible"[6]. To overcome this situation, the damped least square method can be used, where a degraded solution is generated near the singularities proposed in [72] and [48]. A damping factor is introduced in eq. 3.15; by adjusting the damping factor, resulting Jacobian matrix can be made well-conditioned.

The equation 3.15 is modified by introducing a new term λ called damping factor,

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

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

The solution to above problem is,

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

It should be noted that if the value of λ in eq. 3.18 is set to 0, eq. 3.18 reduces to eq. 3.16.

Eq. 3.17 satisfies the condition,

$$\min_{\dot{\theta}}(\left\|v - J(\theta)\dot{\theta}\right\|^2 + \lambda^2 \left\|\dot{\theta}\right\|) \tag{3.19}$$

which evidences the possibility of trading off accuracy against feasibility of the joint velocity required to generate the given end-effector velocity. Therefore, it is essential to select suitable values for the damping factor: Small values of λ give accurate solutions but low robustness to the occurrence of singular and near-singular configurations. Large values of λ result in low tracking accuracy even when a feasible and accurate solution would be possible[6].

The effect of the value of λ on joint velocities can be analyzed further by the singular value decomposition. Using singular value decomposition, equation 3.18 can be re-written as,

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

In above equation, it can be observed that, if $\sigma \gg \lambda$, then λ has practically no effect on the joint velocities. But if σ is close to zero, then the joint velocities are greatly affected by the value of λ. The damping factor determines the degree of approximation introduced with respect to the pure least-squares solution.

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 specific rotational degrees of freedom with the help of the method called *weighted damped least square pseudo inverse*. In this method, Cartesian velocities are multiplied by a weighting matrix.

$$\tilde{v} = \mathrm{W}v \tag{3.21}$$

where W is the (6 x 6) task-space weighting matrix[72].

Substituting $\tilde{v}$ in eq. 3.15, we get,

$$\tilde{v} = \tilde{J(\theta)}\dot{\theta} \tag{3.22}$$

Where, $\tilde{J(\theta)} = \mathrm{W}J(\theta)$

In other words, by choosing appropriate W, ill-conditioned matrix J an be made well conditioned. Here we need to sacrifice the accuracy on particular degree of freedom which depends of choice of W. Further use of damped least square method described above will generate more feasible solutions for joint velocity.

3.1.7 Whole Body Motion

For the robotic manipulators with limited capabilities in terms of reachable workspace, it is not possible to execute the trajectory generated by a DMP which is outside the dexterous workspace. Hence it becomes necessary to use navigation functionality of the mobile platform on which manipulator is fixed in co-ordination

with the manipulator's end-effector motion to track the Cartesian trajectories by end-effector of the manipulator. Such integration of motion of base platform and motion of manipulator end-effector can be called as *whole body motion*.

To implement the whole body motion control, first a policy should be devised which will govern the distribution of motion between manipulator and mobile base platform. This policy should be able to co-ordinate and synchronize the motion executed by mobile base and manipulator. The necessary requirement for such policy design is to identify the capability of manipulator to execute instantaneous motion at each point in time. This capability can be judged by observing the singular values of manipulator's Jacobian matrix.

When a manipulator end-effector is operating near the boundary of reachable workspace, a velocity command which drives end-effector towards the boundary makes the Jacobian matrix of the manipulator more ill-conditioned and smallest singular value obtained by singular value decomposition of Jacobian matrix, tends to zero[72]. In other words, a manipulator's capability of executing motion commands in the direction towards the boundary of work-space decreases with decreasing value of smallest singular value. It should be noted that, velocity commands driving manipulator end-effector away from the boundary inside the reachable workspace, can be executed normally.

In this project, a linear motion distribution policy was developed which calculates the velocity commands for the base and manipulator end-effector by observing the smallest singular value of Jacobian matrix of the manipulator. The singular values can be obtained as follow:

The relation between the joint-space velocities and Cartesian velocities is given by equation 3.15, which is

$$v = J(\theta)\dot{\theta} \tag{3.23}$$

Where,
v is the velocity vector in global co-ordinate frame.

Singular values of Jacobian can be obtained by singular value decomposition,

$$J = \sum_{i=1}^{6} \sigma_i u_i \mathbf{v}_i^T \tag{3.24}$$

Where, σ_i are the singular values.

The policy for distribution of linear velocities amongst the mobile base platform and manipulator end-effector is given by,

$$m_{cap} = \frac{(\sigma_{min} - \sigma_l)}{(\sigma_h - \sigma_l)} \tag{3.25}$$

$$v_{ee} = m_{cap}.v \tag{3.26}$$

$$v_b = (1 - m_{cap}).v \tag{3.27}$$

Where,
m_{cap} is capability co-efficient of manipulator,
σ_{min} is the smallest singular value,
σ_l is the lower limit on σ_{min},
σ_h is the upper limit on σ_{min},
v is the desired linear velocity of manipulator end-effector in global frame of reference,
v_{ee} is the velocity command for end-effector in global frame of reference,
v_b is the velocity command for mobile base in global frame of reference.

It should be noted that above policy comes into effect only when $\sigma_{min} \leq \sigma_h$. Also, only linear motion is distributed between mobile base and manipulator end-effector, rotational motion is executed by the manipulator only.

It is also possible that in the environment in which the robot is operating, motion of the mobile base is not possible in particular direction because of the obstacles. Hence it is necessary to incorporate this information while distributing the motion commands. Above policy can be modified as follow to accommodate the constraints on the base movement.

$$m_{cap} = \frac{(\sigma_{min} - \sigma_l)}{(\sigma_h - \sigma_l)} \tag{3.28}$$

$$b_{cap} = \frac{(d - d_l)}{(d_h - d_l)} \tag{3.29}$$

$$v_{ee} = \frac{m_{cap}}{m_{cap} + b_{cap}}.v \tag{3.30}$$

$$v_b = \frac{b_{cap}}{m_{cap} + b_{cap}}.v \tag{3.31}$$

Where,

m_{cap} is capability co-efficient of manipulator,

σ_{min} is the smallest sigma value,

σ_l is the lower limit on σ_{min},

σ_h is the upper limit on σ_{min},

b_{cap} is capability co-efficient of mobile base,

d is the distance of the obstacle from base,

d_l is the lower limit on d,

d_h is the upper limit on d,

v is the desired velocity of end-effector in global frame of reference,

v_{ee} is the velocity command for end-effector of the manipulator in global frame of reference,

v_b is the velocity command for mobile base in global frame of reference.

Here, if the sum $(m_{cap} + b_{cap})$ drops below certain threshold value, it can be concluded that the further motion is not possible, as both the motion capabilities are reduced significantly.

4

Solution

In this project, a system was implemented which is able to the record human demonstrations as kinematic trajectories, learn the motion in the form of dynamic motion primitive, reproduce the motion using dynamic motion primitives and execute this motion with mobile manipulator. This chapter discusses the actual design and the implementation of the software system.

4.1 Software Components

Fig. 4.1 shows the architecture of the implemented software system. Let's discuss components of this architecture one-by-one.

4.1.1 demonstrated_trajectory_recorder

demonstrated_trajectory_recorder records the trajectory demonstrated by the teacher with the help of arUco marker board, in the Cartesian space. Recorded trajectory is stored in a nx6 matrix. This matrix is then stored on the disk in *yaml* file. Although all 6 degrees of freedom (3 linear and 3 rotational) are recorded during the demonstration, only linear degrees of freedom (X,Y,Z) are used for learning as demonstration of rotational degrees of freedom using arUco marker is difficult. Once the trajectory is recorded, it is also published on a *rostopic* as a *naviagation path* message for visualization in *rviz*.

4.1.2 ros_dmp

ros_dmp module consists of core DMP framework implementation called *pydmps*, which is taken from internet source [13]. Implementing core DMP framework is fairly simple and straight forward, hence instead of implementing it from scratch, efforts were focused towards adapting already available implementation for Robot Operating System and robots. Changes were made in the original package so that it can be integrated with Robot Operating System and can be used on the robot. Changes include:

- Addition of a method for retrieval of the weights of learned DMP.
- Modification in *roll()* method to allow passing already learned weights.
- Removing unnecessary code for visualization.
- Transforming package into a *catkin* package so that it can be used as library in ROS workspace.

Module *learn_motion_primitive* provides the interface for learning the motion primitives from recorded trajectories. It reads recorded trajectory stored in *yaml* file, uses the functionalities provided by the package *pydmps* to learn motion primitive, stores weights in the another *yaml* file and visualizes the learned DMP using *RVIZ* tool in ROS.

Module *roll_dmp* reads the weight file associated with desired DMP to be executed. It takes the target end-effector pose in Cartesian co-ordinates as goal input and generates trajectory using functionalities provided by *pydmps* package. It is also possible to solve the *transformation system* step by step at fixed time intervals to generate instantaneous motion commands. This is necessary in the scenario where feedback from the environment and robot is needed to be incorporated in the DMP. One example of such feedback is on-line goal modification.

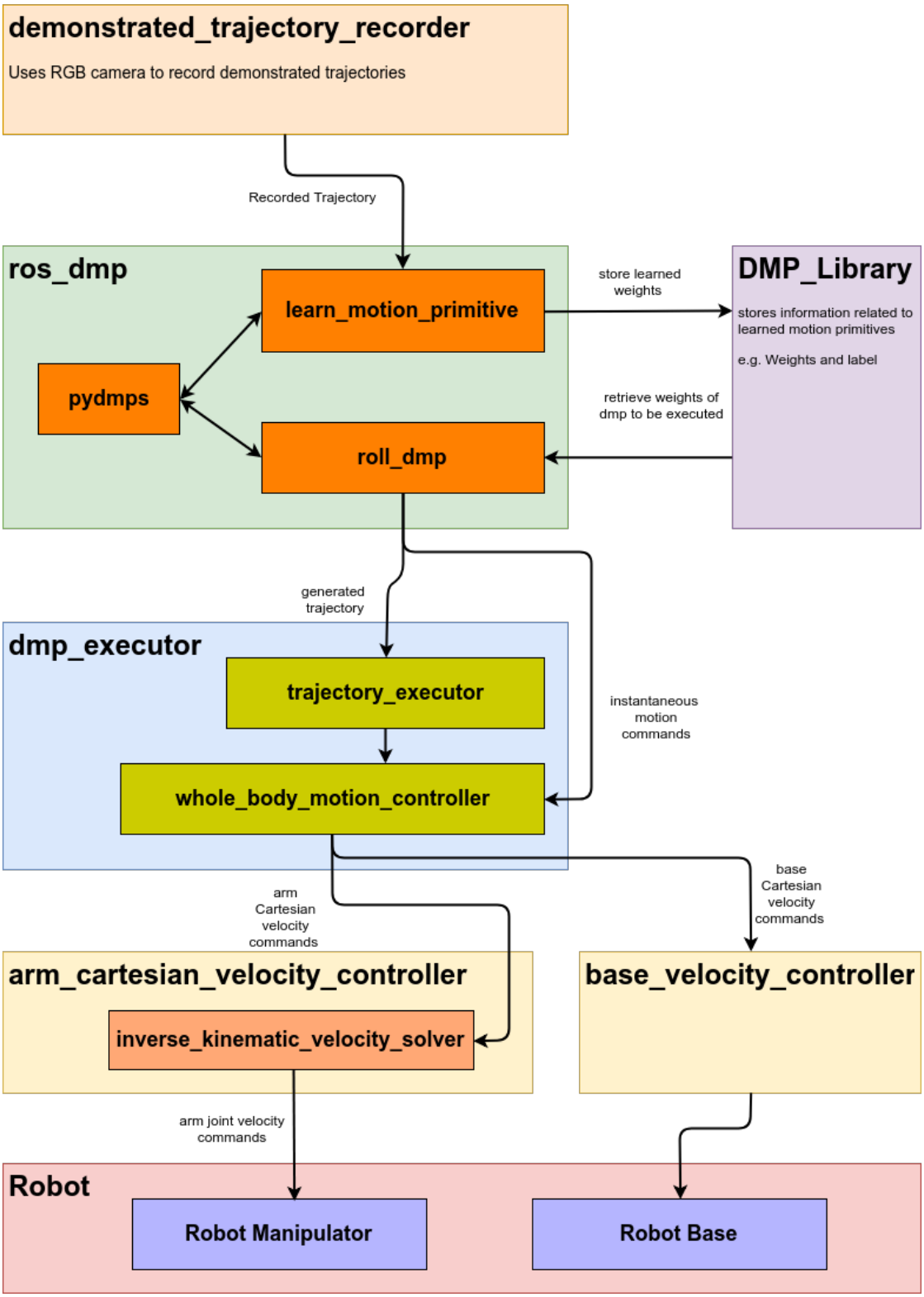

Figure 4.1: Learning from Demonstration framework

4.1.3 DMP_Library

DMP_Library is a collection of *yaml* files containing learned weights of DMPs. Files are labeled appropriately so that the weights can be retrieved whenever necessary.

4.1.4 dmp_executor

dmp_executor acts as the upper level control unit which executes trajectories or motion commands generated by *roll_dmp* module in *ros_dmp* package. *trajectory_executor* receives desired trajectory from *ros_dmp* and passes the velocity command to *whole_body_motion_control* unit, at each time step. Instead of using time as a velocity update parameter, it takes feedback of the pose of the end-effector and applies appropriate velocity command when end-effector reaches at particular position on the path.

whole_body_motion_control receives the velocity command from *trajectory_executor* or directly from *roll_dmp* module. The velocity command is then processed according to the policy described in section 3.1.7 to obtain end-effector velocity and base velocity. These velocities are then passed to respective low level controllers.

4.1.5 arm_cartesian_velocity_controller

This package is a upgraded version of the package *mcr_arm_cartesian_control* in *mas_common_robotics* library of b-it-bots. This package use *OROCOS KDL (Kinematics and Dynamics Library)* library for calculating joint velocities for required Cartesian velocities of the end-effector frame. Class KDL::ChainIkSolverVel_wdls in *OROCOS KDL* implements the method described in section 3.1.6.

To use the previously implemented package in this project, some crucial changes were needed in order to meet new requirements. These requirements are:

- Accessing singular values of Jacobian of the manipulator. (refer section 3.1.6 for singular values)
- A way to change task space weights used in method described in 3.1.6.

- Compatibility with new Toyota HSR robot.

Real time access to the singular values is necessary for calculating velocity commands for whole body motion control. Old package *mcr_arm_cartesian_control* uses old version of KDL which is the default version of KDL in ROS Kinetic. This version of KDL does not provide interface to access singular values in real time. Hence the old package is adopted for new version of KDL. A method was created to publish singular values on a *rostopic* in real time.

As both of the manipulators used in the experiments have 5 degrees of freedom, it is necessary to update task-space weights used to loosen the constraints on relatively less important motions in specific degree of freedom in real time. Hence a function was implemented which listen to a *rostopic* on which weights can be published. This function updates the weights upon receiving new weights.

Velocity message types used in Toyota HSR robot are different than those being used in KUKA YouBot, hence it was necessary to upgrade old package to use message types that can be used by both robots.

4.1.6 Robot

Once the joint velocity commands and the linear velocities for the base are generated, they are published on respective *rostopics*. These *rostopics* are subscribed by the respective hardware interface components, which are responsible for the execution of motion commands. Study and analysis of these components is out of the scope of this project.

5

Experimental Evaluation

In the section 3.1.4, a primary analysis was conducted to evaluate the effects of various parameters used in a DMP, on the trajectory approximation. This analysis gave insights of capabilities, limitations and precautions that are needed to be taken while using a DMP. But to evaluate the entire *Learning from Demonstration* framework, experiments are needed to be carried out on real robots. Numerous such experiments were conducted on KUKA YouBot and Toyota HSR, for evaluating the performance of DMP framework and proposed methodology. The experiments conducted on KUKA YouBot were mainly aimed at:

- evaluating generalization ability of DMPs,
- evaluating performance of whole body motion control with DMPs,

Two experiments were also conducted on Toyota HSR for:

- demonstrating the easiness of porting above solution from one robot to another,
- evaluating performance of whole body motion control with DMP,
- sequencing two DMPs to grasp an object.

5.0.1 Experimental Setup

Robot 1	:	KUKA YouBot
Robot 2	:	Toyota HSR
Camera	:	Asus Xtion Pro Live
Softwares and Libraries	:	OpenCV 3.1.0
	:	Robot Operating System (ROS) Kinetic
	:	OROCOS Kinematics and Dynamics library
Environment 1	:	RoboCup@Work lab, C025
Environment 2	:	RoboCup@Home lab, C069

5.0.2 Acquisition and Repetition of Dynamic Motion Primitives

Aim of this experiment was to evaluate the ability of above proposed software architecture, of acquiring human demonstrations, learning the motion primitives, and repeat the motion primitives multiple times for different goals, in order to evaluate generalization ability of motion primitives for different end-effector goal positions (hereafter, goal always refer to end-effector goal position) and the ability of accurately executing the motion primitives.

In this experiment, four different trajectories were demonstrated to the robot in task-space, motion primitives were learned and stored in the library. Then each motion primitive was used to generate trajectories with five different goals in task-space. Ten trials were conducted for each of the goals to check the repeatability. In each trial, the trajectory started from the same initial position. Lets discuss the experiments for each motion primitive, in detail.

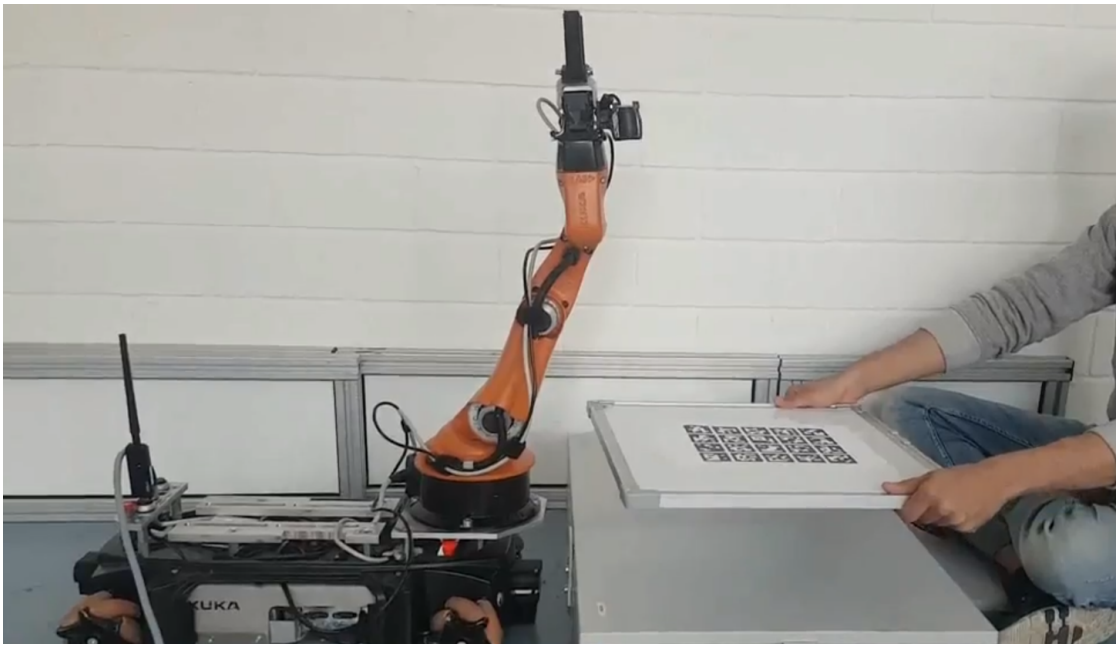

Figure 5.1: Demonstration of the trajectory using arUco marker board

Inverted Parabolic Trajectory 1

Fig. 5.2 shows the inverted parabola like trajectories for five different goals. Figure contains both, the planned trajectories generated by the DMP and the trajectories executed by the robot. All the trajectories are generated from one motion primitive.

Experimental parameters:

$n\_bfs = 50$ $\tau = 10$ $dt = 0.01$

Initial position = [0.522, -0.058, 0.042]m

Goal 1 = [0.537, 0.161, 0.044]m Goal 2 = [0.508, 0.161, 0.044]m

Goal 3 = [0.538, 0.131, 0.044]m Goal 4 = [0.508, 0.181, 0.044]m

Goal 5 = [0.568, 0.141, 0.044]m

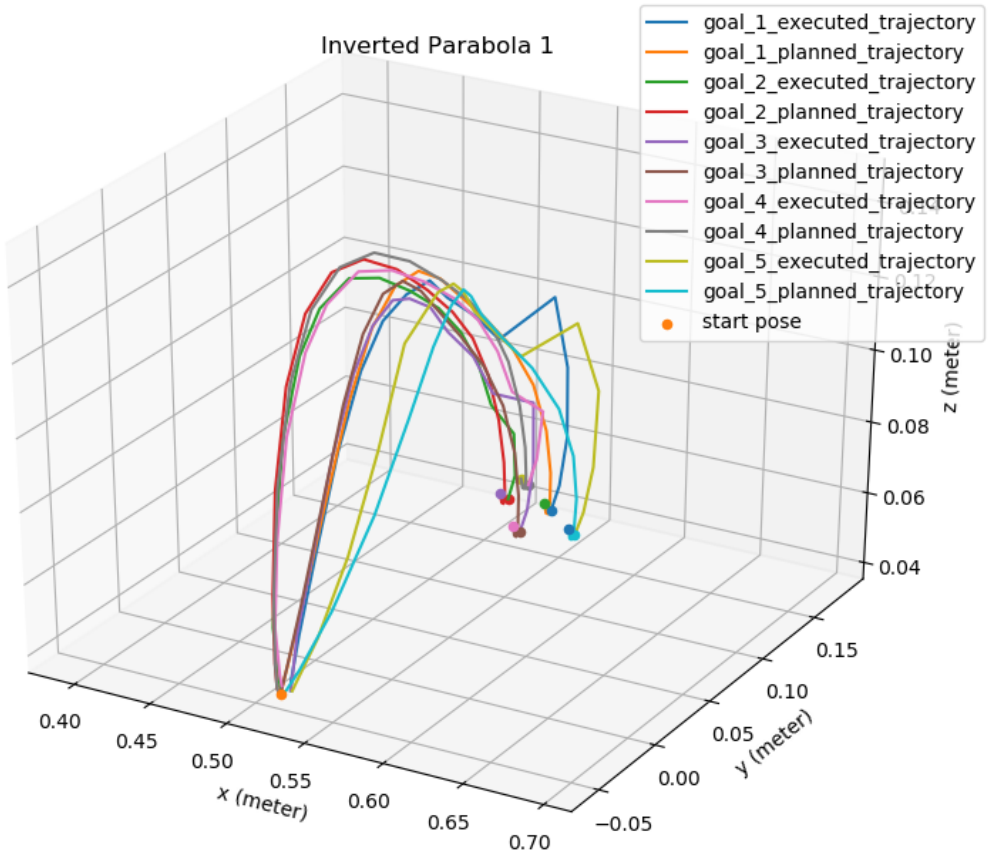

Figure 5.2: Inverted parabolic trajectory 1

In the figure, it can be observed that the executed trajectories differ from the planned ones, which suggest that the error is introduced by the trajectory controller as well as the joint velocity controllers in the robot. Despite of this, error was calculated as the normalized point-to-point error for the entire trajectory. In case of executed trajectories for goal 1 and goal 5, a sudden overshoot can be observed at end of the trajectory. This overshoot occurred because of this part of trajectory was out of the workspace of the manipulator. A degraded inverse kinematic solution is generated in such cases by the method described in the section 3.1.6. Despite of this, end-effector was able to reach the goal position because of the position feedback used in trajectory executor. This situation particularly triggered the idea of using base motion and hence the whole body motion to execute the trajectories.

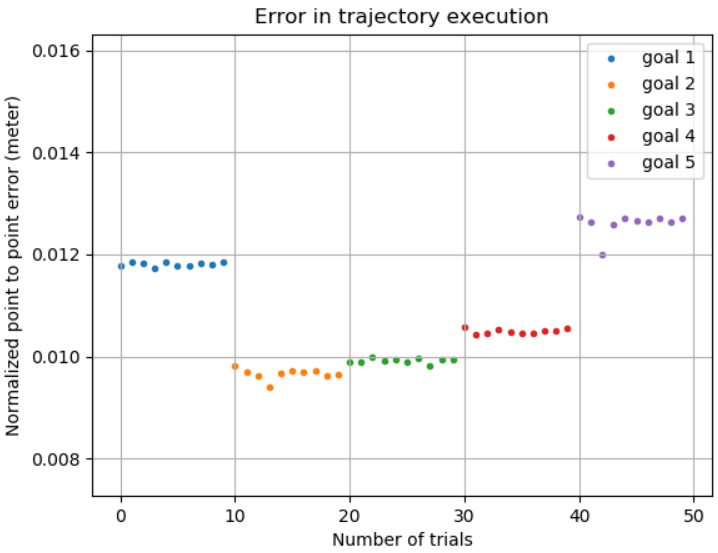

Figure 5.3: Error in the execution of trajectories

Fig. 5.3 shows the normalized point to point error in each executed trajectory calculated by eq. 3.14.

Inverted Parabolic Trajectory 2

Figures 5.4 and 5.5 respectively show another inverted parabola like trajectory generalized for 5 different goals and the error in the execution of same over ten trials per goal.

Experimental parameters:

$n\_bfs = 50$ $\tau = 10$ $dt = 0.01$

Initial position = [0.456, -0.234, 0.121]m

Goal 1 = [0.496, 0.100, 0.089]m Goal 2 = [0.476, 0.119, 0.058]m

Goal 3 = [0.451, 0.149, 0.043]m Goal 4 = [0.493, 0.179, 0.074]m

Goal 5 = [0.493, 0.228, 0.089]m

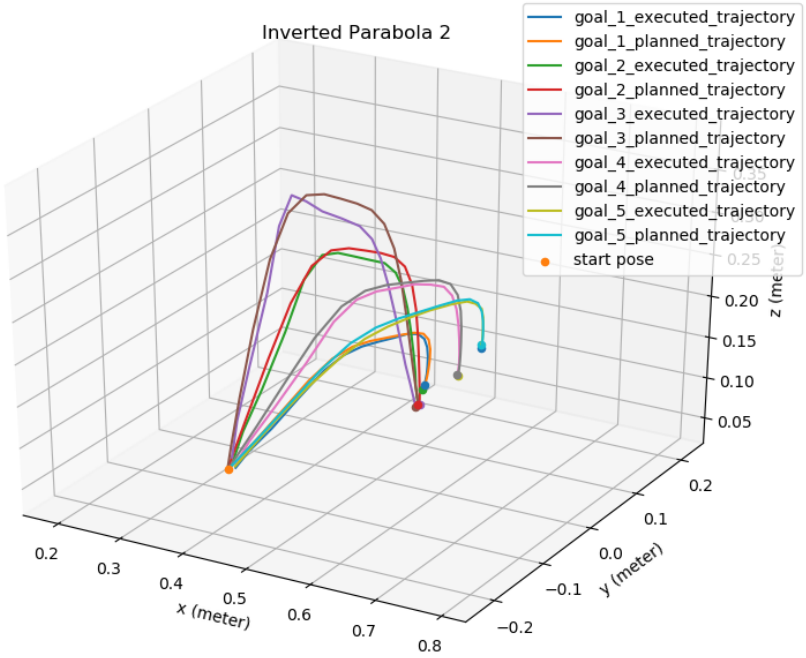

Figure 5.4: Inverted parabolic trajectory 2

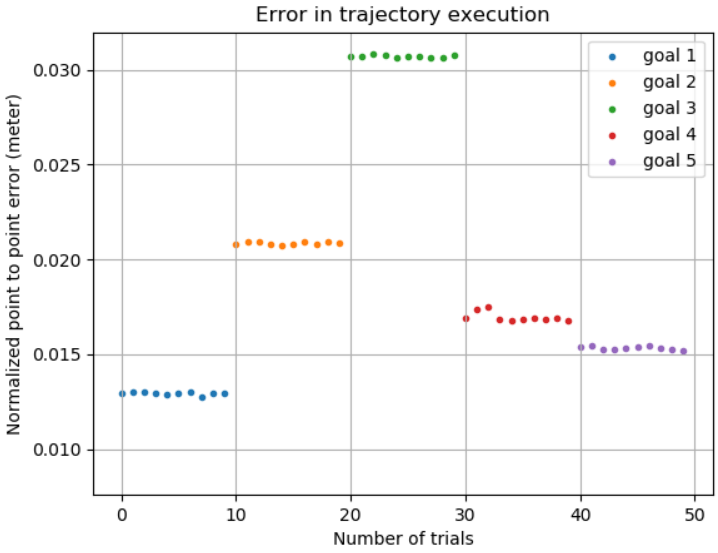

Figure 5.5: Error in the execution of trajectories

Inverted Parabolic Trajectory 3

Trajectories in fig. 5.6 show slightly different behavior. While executing these trajectories, a observation was made : if the sign of the difference between goal and the initial position (sign of $(g - y0)$) used in a DMP is opposite that of the trajectory from which the DMP was learned (sign of $(g_{demo} - y0_{demo})$), the shape of the newly generated trajectory is mirrored. This can be observed from planned and executed trajectory for goal 2.

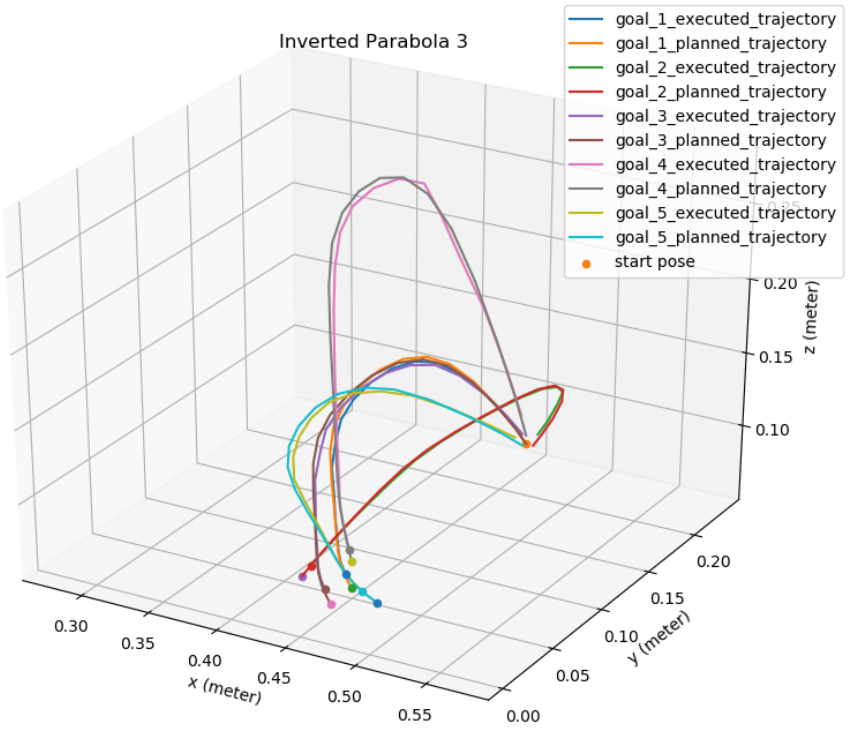

Figure 5.6: Inverted parabolic trajectory 3

The reason behind this behavior can be found in eq. 3.3, where term $(g - y)$ is multiplied to weighted sum of Gaussian functions to generate the forcing term f. Hence if the sign of $(g - y)$ is opposite, then the forcing term also carries the opposite sign and generates the accelerations in opposite directions so that shape of the generated trajectory is mirrored.

Experimental parameters:

$n\_bfs = 50$ $\tau = 10$ $dt = 0.01$

Initial position = [0.446, 0.230, 0.063]m

Goal 1 = [0.470, 0.023, 0.074]m Goal 2 = [0.434, 0.023, 0.074]m

Goal 3 = [0.470, 0.003, 0.074]m Goal 4 = [0.470, 0.023, 0.092]m

Goal 5 = [0.488, 0.023, 0.068]m

Figure 5.7 shows the error in the execution of above trajectories.

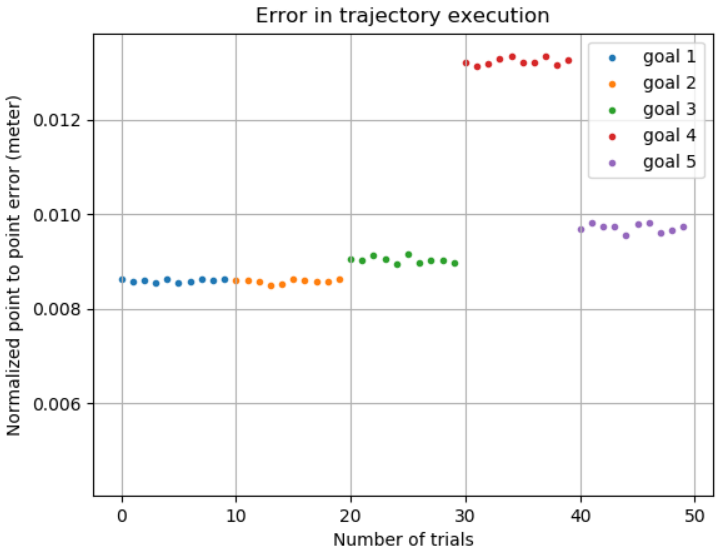

Figure 5.7: Error in the execution of trajectories

Step Function Trajectory

Fig. 5.8 shows the step function trajectories. This time, the step function like trajectory was demonstrated to the robot. Planned and executed trajectory for goal 2 is of special interest in this experiment. This trajectory was obtained by swapping the weights learned for DMPs in X and Y axis. It can be concluded from this experiment that, by combining DMPs learned in individual degrees of freedom, a new motion primitive can be synthesized. Another important observation in this experiment was on the use of time scaling factor τ. For higher values of τ, a overshoot was observed in the trajectory.

Experimental parameters:

$n\_bfs = 50$ $\tau = 10$ $dt = 0.01$

Initial position = [0.5, -0.137, 0.05]m

Goal 1 = [0.500, 0.172, 0.129]m Goal 2 = [0.561, 0.000, 0.129]

Goal 3 = [0.500, 0.172, 0.153]m Goal 4 = [0.500, 0.204, 0.145]

Goal 5 = [0.500, 0.118, 0.089]

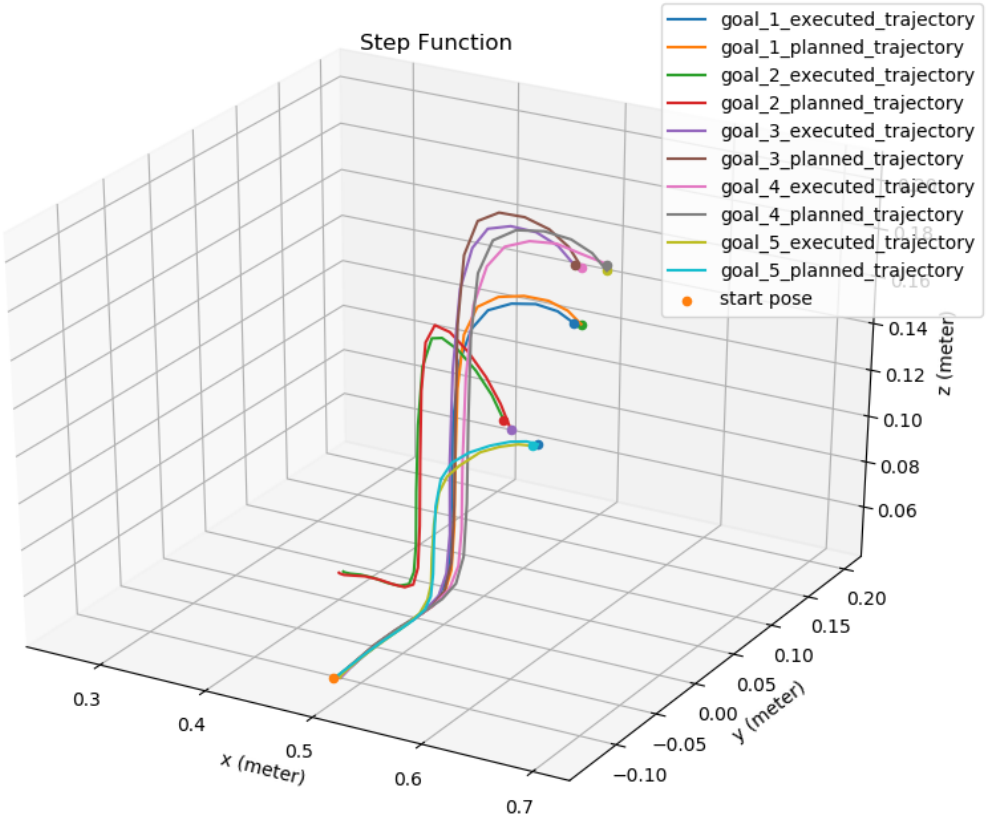

Figure 5.8: Step function trajectory

Fig. 5.9 shows the error in the execution of step function trajectories for different goals.

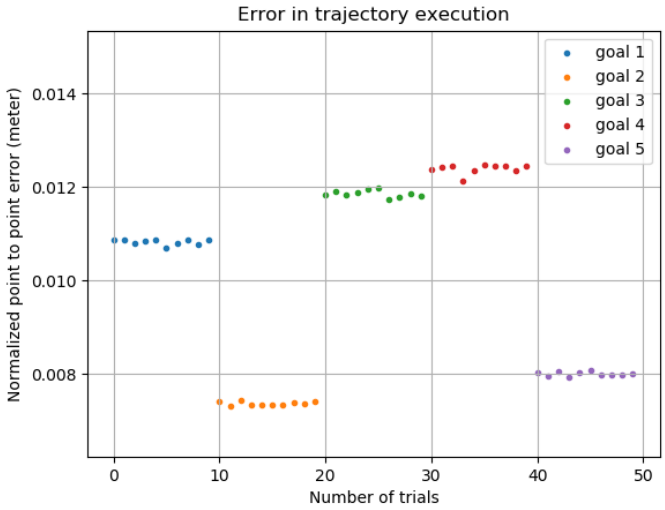

Figure 5.9: Error in the execution of step function trajectories

Square Wave Trajectory

An artificially generated square wave trajectory in X-Y plane (Z is constant) was learned using DMPs. Reason behind learning this particular trajectory was evaluate the ability to learn and execute motion with sharp turns. From earlier experience, the number of basis function was decided to be 500 because of the sharp turns which contribute to high frequency components. The controller, as expected, executed trajectory with oscillations at the corners of the path. These oscillations can be seen in fig. 5.10. Because of high number of basis functions and low τ, DMP generated a stable trajectory, but linear trajectory controller could not execute it perfectly; this marks the need of non-linear controller (torque controller) for more precise trajectory execution. However, the error in execution in trajectories is considerably low as seen in fig. 5.11, hence for practical purposes, the controller's performance is satisfactory.

A small peak along Z-axis can be observed on one of the corners of executed trajectory of goal 4; it occurred because the part of trajectory was near the boundary of dexterous workspace of the manipulator. As explained earlier, a degraded inverse kinematic solution was generated which led manipulator end-effector to deviate from the trajectory.

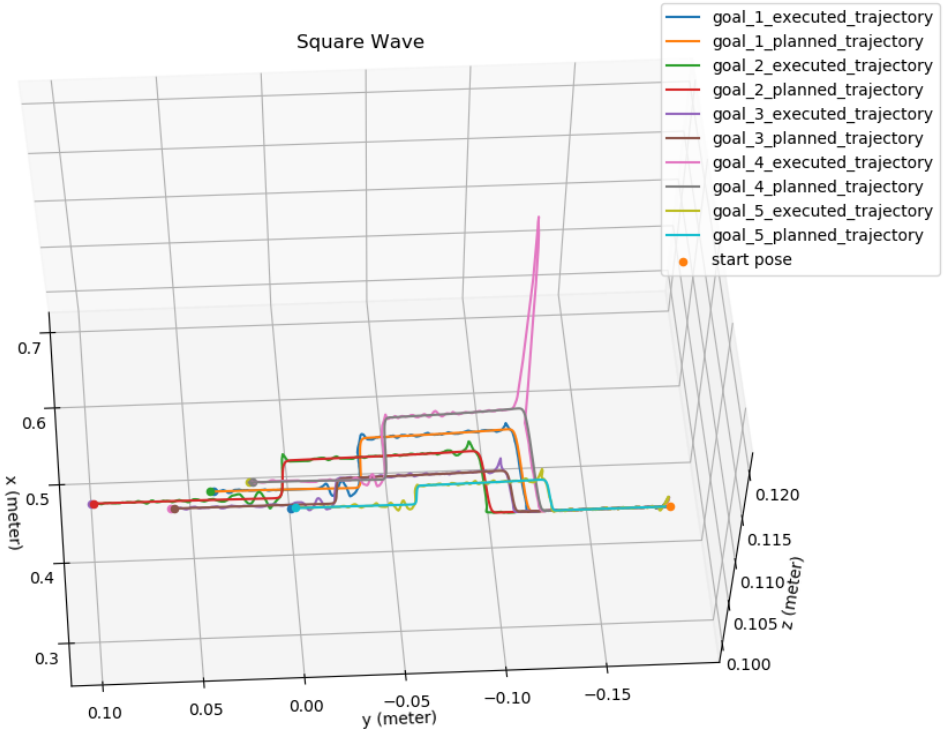

Figure 5.10: Square wave trajectory

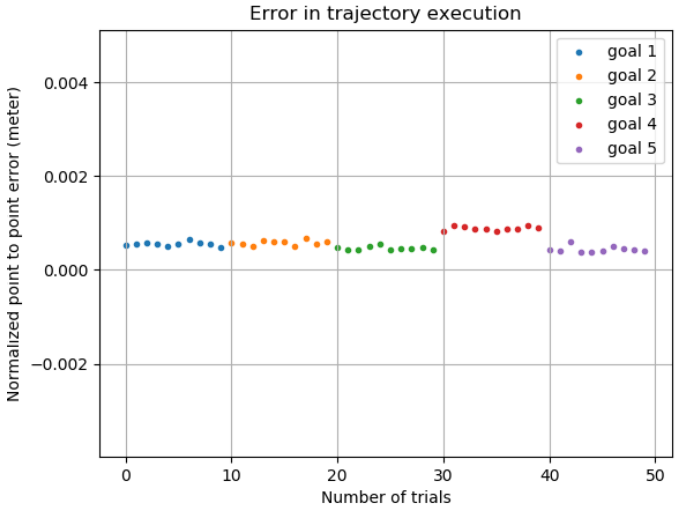

Figure 5.11: Error in the execution of square wave trajectories

5.0.3 Whole Body Motion Control on KUKA YouBot

This experiment was conducted to demonstrate the ability of the proposed *whole body motion control* architecture. 3 different trajectories were learned using DMP and used to generate the trajectories for five different task-space goals which are not in dexterous workspace of manipulator. Ten trials were conducted for each goal.

Aim of this experiment was to evaluate the trajectory tracking ability with whole body motion control algorithm in the presence of uncertainties like slip in the wheels of mobile base, friction provided by the floor, etc. Apart form the parameters used in the DMP, motion is now affected by the parameters used in the whole body motion control.

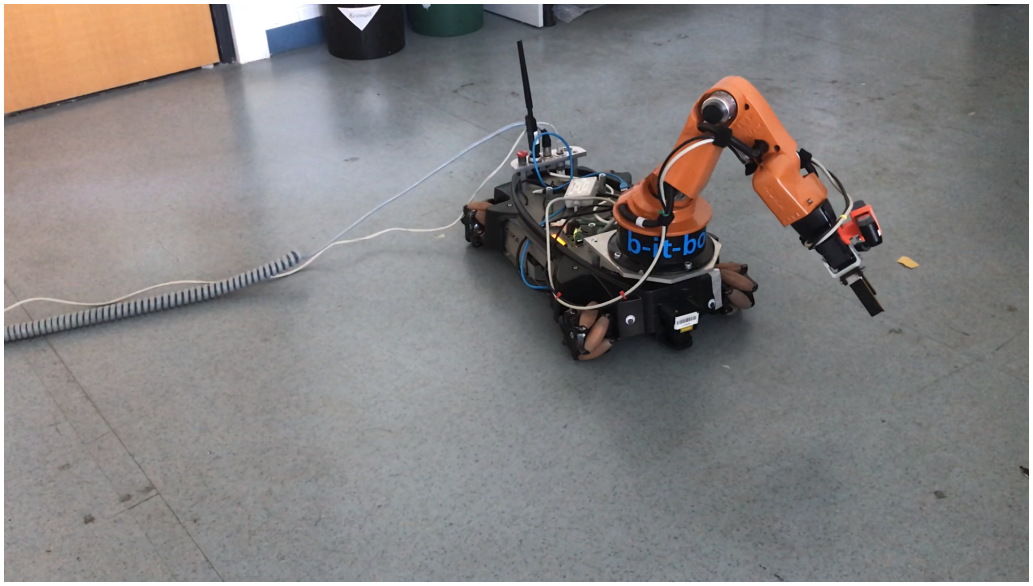

Figure 5.12: KUKA YotBot performing whole body motion

Inverted Parabolic Trajectory

Same as the first experiment, inverted parabolic trajectory was demonstrated to the robot, learned and generalized for five different goals which are out of the dexterous workspace of the robot manipulator. It should be noted that the only internal wheel encoder feedback was used while motion execution hence it is not possible to

compensate the error introduced by imprecise motion of the base. Hence, even if the error in execution as per the fig. 5.14 is small, actual motion was not precise in global frame of reference. It was evident from the visual observation that the end-effector positions, after finishing motion for same goal in two different trials, were not same.

Experimental parameters:

$n\_bfs = 500$ $\tau = 10$ $dt = 0.01$

Initial position = [0.423, -0.173, 0.110]m

Goal 1 = [0.229, 0.206, 0.062]m Goal 2 = [0.439, 0.358, 0.062]m

Goal 3 = [0.492, 0.446, 0.062]m Goal 4 = [0.535, 0.412, 0.062]m

Goal 5 = [0.639, 0.510 0.062]m

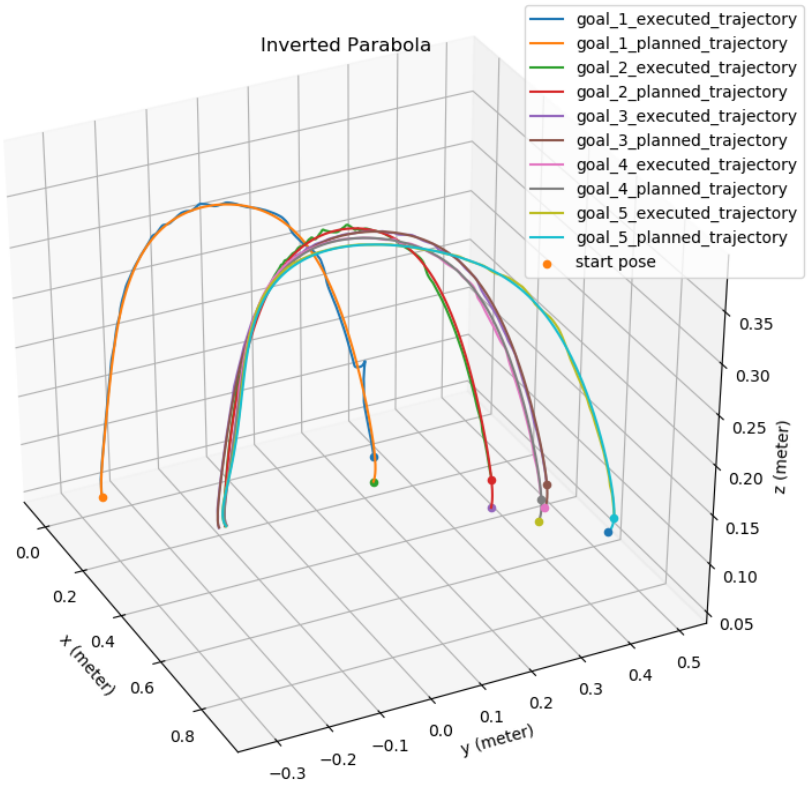

Figure 5.13: Inverted parabolic trajectory

Figure 5.14: Error in the execution trajectories

Square Wave Trajectory 1

A square wave like trajectory was demonstrated and learn. It was also generalized for five different goals. Interesting fact in this experiment was the imprecise execution of trajectories near the joint limits as observed in executed trajectories for goal 1 and 5 in fig. 5.15. Another important observation is, when the goal separation, $(g - y0)$, where $y0$ is the initial position, in certain degree of freedom is 0, then DMP does not take off i.e. does not leave the initial position. In planned and executed trajectories for goal 4, it can be observed that the goal separation in X-axis is 0, hence the shape of the trajectory is not as expected. The reason behind such behavior is the that the forcing term f which responsible for the shape of the trajectory is scaled by goal separation (refer eq. 3.3) and hence the 0 goal separation makes f equal to 0, at all the time.

Experimental parameters:

$n\_bfs = 500$ $\tau = 10$ $dt = 0.01$

Initial position = [0.45, -0.116, 0.11]m

Goal 1 = [0.394, 0.374, 0.11]m Goal 2 = [0.416, 0.392, 0.11]m

Goal 3 = [0.387, 0.493, 0.11]m Goal 4 = [0.445, 0.341, 0.11]m

Goal 5 = [0.342, 0.540, 0.11]m

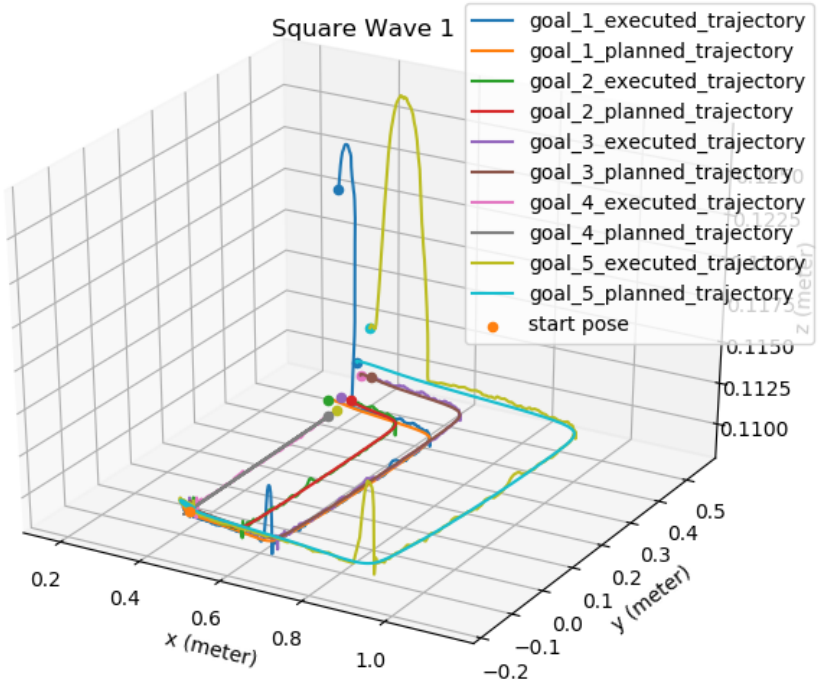

Figure 5.15: Square wave trajectory 1 (WBC)

Figure 5.16: Error in execution of trajectories

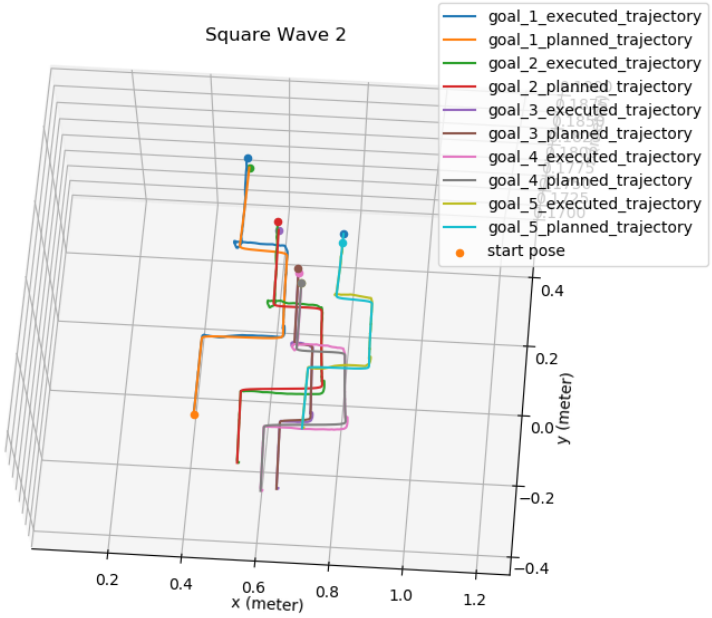

Figure 5.17: Square wave trajectory 2 (WBC)

Figure 5.18: Error in execution of trajectories

Fig. 5.17 and 5.18 respectively show another square wave like trajectory

execution and error in the execution.

Discussion

This was first experiment conducted on the robot which helped us get insights of the problems in using DMPs on real robot. Some important observation were made:

- If some part of a trajectory is out of the dexterous workspace or near its limits (e.g. the goal position is at the limits of the workspace), the motion is unpredictable due to degraded inverse kinematics solutions; this limits the ability to generalize a learned primitive.

- A similar observation holds near the joint limits.

- If the demonstrated trajectory has initial and final values of position on particular axis that are practically equal to each other, varying goal on that axis creates infeasible resulting trajectories; this primitive is thus limited to scenarios in which the initial and final position are the same on that axis. For example, if a motion primitive is learned for picking an object and placing it elsewhere at same height ($z_{initial} = z_{goal}$), and used for placing object at higher heights, it can create potentially dangerous trajectories. Reason behind this behavior is the scaling term $(g - y_0)$ in eq. 3.3; while learning if this scaling term is practically zero, and while generating new trajectory, it is relatively big (sign of this term does not matter), shape of the DMP in that particular direction is scaled.

- Increasing τ too much (e.g. 100) causes overshoots at sharp turns.

- At sharp turns, oscillations around the trajectory occur because of linear trajectory controller.

- For learning motion with sharp turns, use of high number of basis functions is desired.

5.1 Dynamic Motion Primitives on Toyota HSR

For demonstrating the usefulness of DMPs in RoboCup@Home scenario, the *Learning from Demonstration* framework was adopted for Toyota HSR and two experiments were conducted:

1. Sequencing two DMPs for pick and place task
2. Grasping an object

In these experiments, obstacles were present in the environment and thus the motion of the base was restricted. Hence feedback form the laser scanners was used in the whole body motion control.

5.1.1 Sequencing two DMPs for pick and place task

Figure 5.19: Toyota HSR performing pick and place task

In this experiment, two motion primitives were demonstrated to the robot; one for reaching to the object in straight line from a fixed initial position and another

for placing that object on the table. These two motion primitives can be seen in fig. 5.20 as well as in 5.21 from a different angle.

The learned motion primitives were used to pick a hypothetical object from a low height table and place on a relatively taller table (height = $0.78m$). Thirteen trials were conducted for same picking and placing position. Trajectory 1 in fig. 5.20 is for reaching to the object and trajectory 2 is for transporting the object to the placing location.

Fig. 5.22 and 5.23 shows the error in the execution of the generated trajectories in each trial. The error is here is relatively higher as compared to the KUKA YouBot. The reason behind it is the joint velocity controller in Toyota HSR; which is an emulated pseudo joint velocity controller which uses joint position controller to implement the joint velocity control, hence it does not provide precise velocity control.

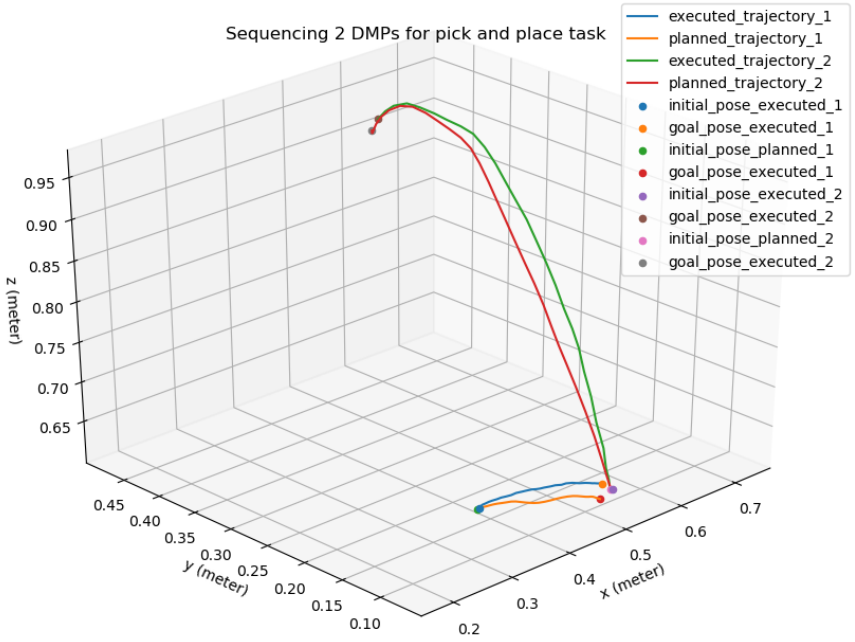

Figure 5.20: Sequencing two DMPs for pick and place task - 1

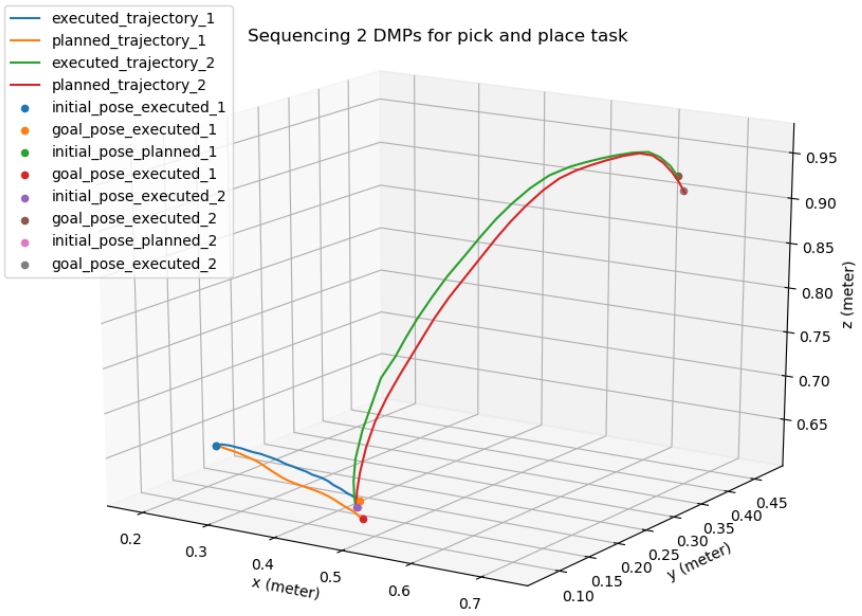

Figure 5.21: Sequencing two DMPs for pick and place task - 2

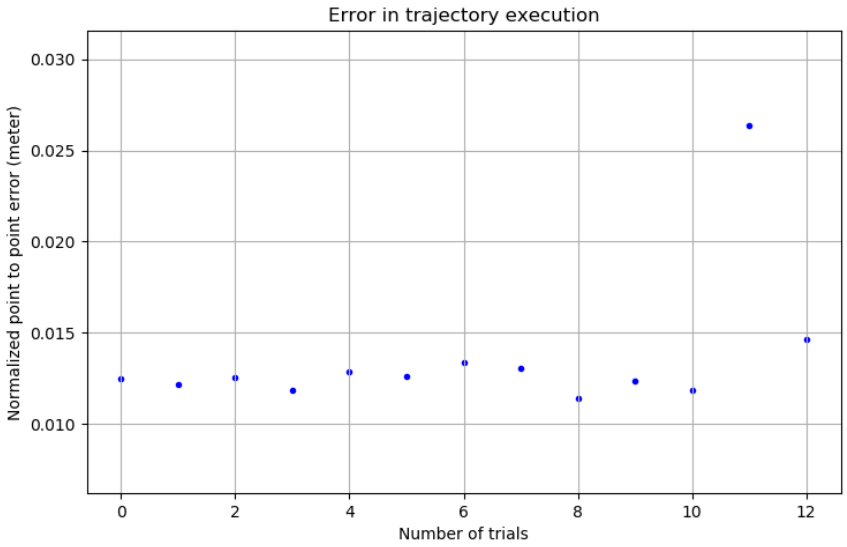

Figure 5.22: Error in execution of trajectory 1

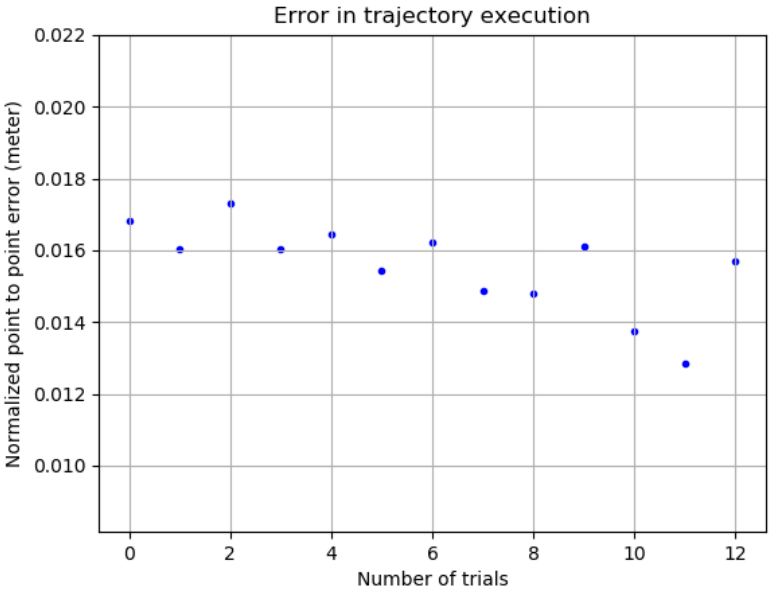

Figure 5.23: Error in execution of trajectory 2

5.1.2 Grasping an object

Figure 5.24: Toyota HSR grasping noodle box

Following figures show the performance of DMP framework integrated in current *move_arm* action server used in Toyota HSR. In this experiment, actual poses of the objects were obtained with the help of perception pipeline, and DMPs were used to generate the trejectories for grasping these objects. DMP generated trajectories for every goal given as opposed to the MoveIt! motion planning architecture, where solutions were not generated most of the time as the object was not in the reach of the arm. Execution of the trajectories generated by DMP was possible because of the whole body motion. In this experiment, executed trajectories show relatively higher deviations from planned trajectories. This happened because of the low update rate of the end-effector position. Error in the trajectory execution, shown in 5.31, is relatively low and hence all the grasps were successful.

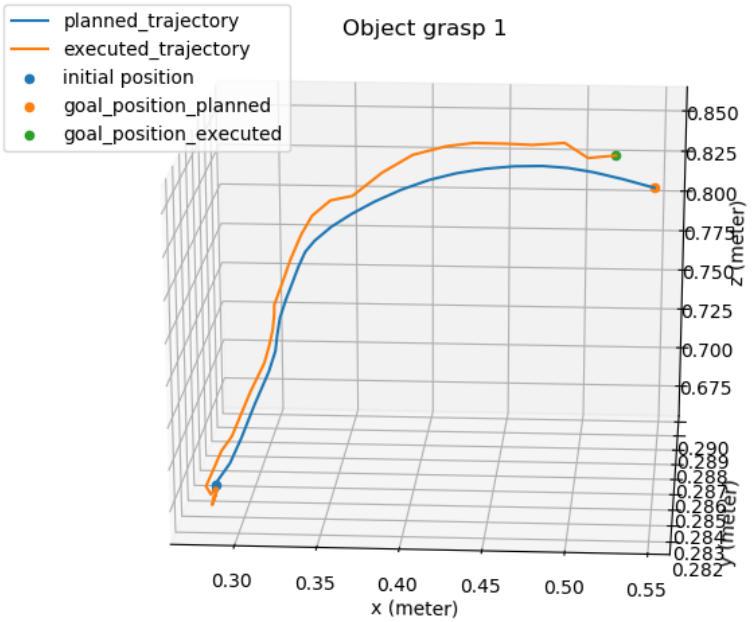

Figure 5.25: Grasp 1

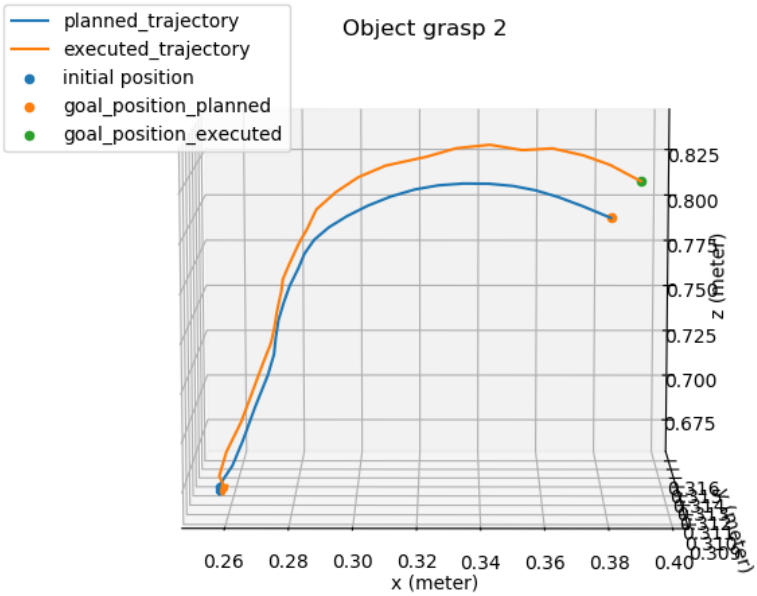

Figure 5.26: Grasp 2

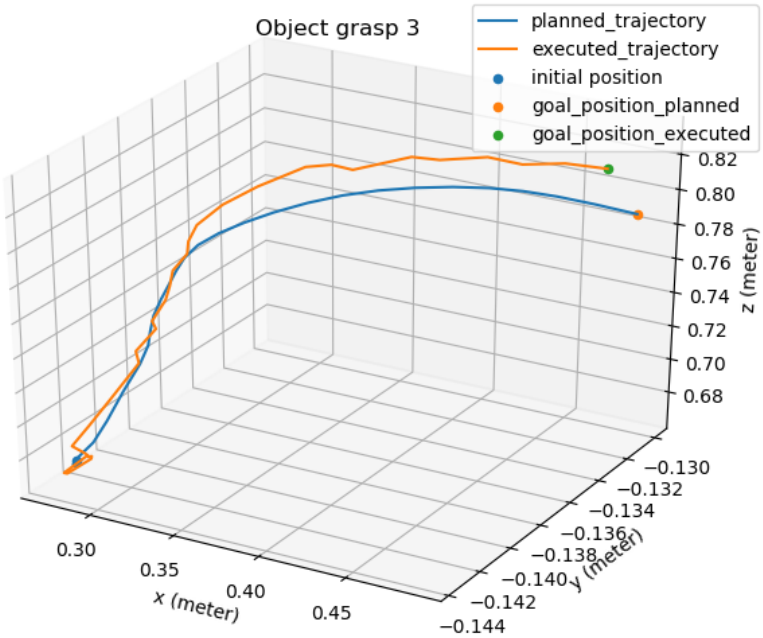

Figure 5.27: Grasp 3

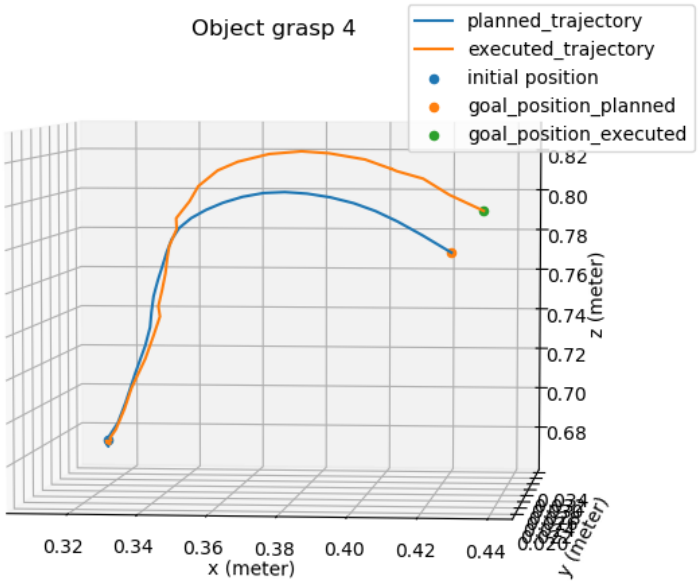

Figure 5.28: Grasp 4

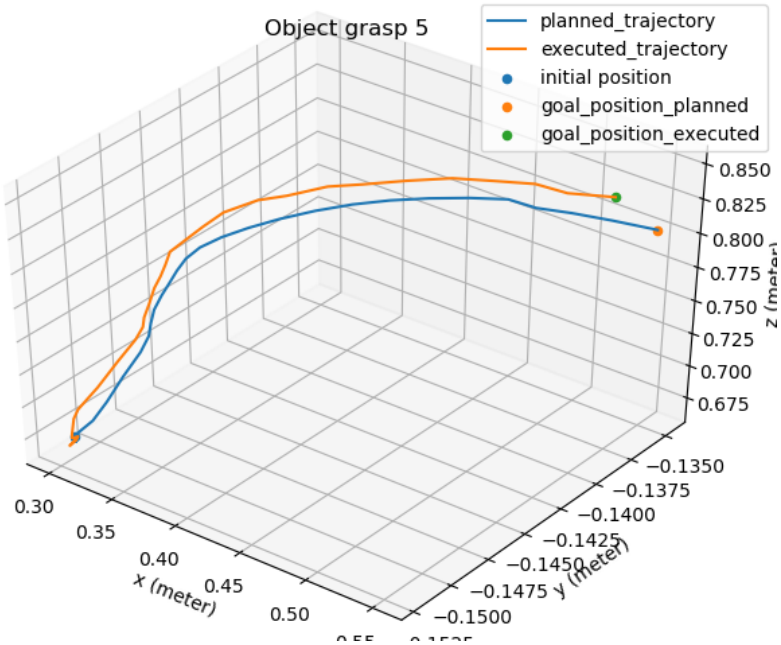

Figure 5.29: Grasp 5

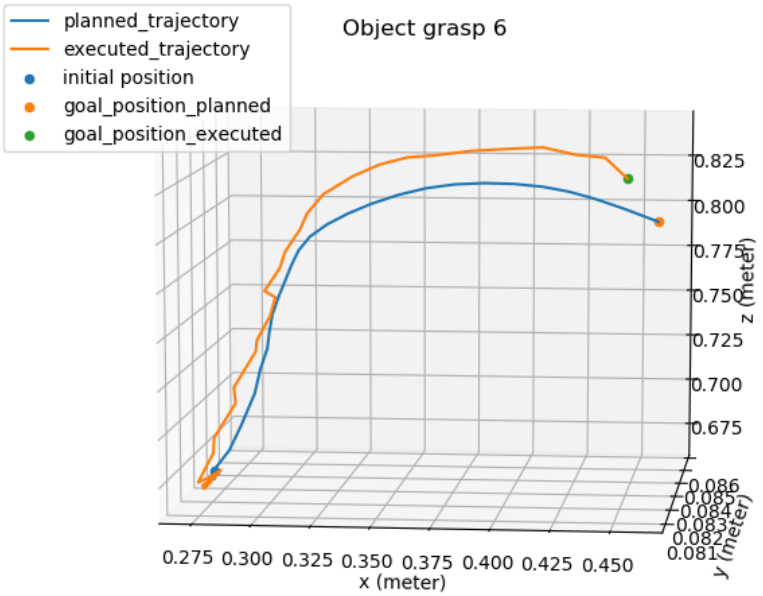

Figure 5.30: Grasp 6

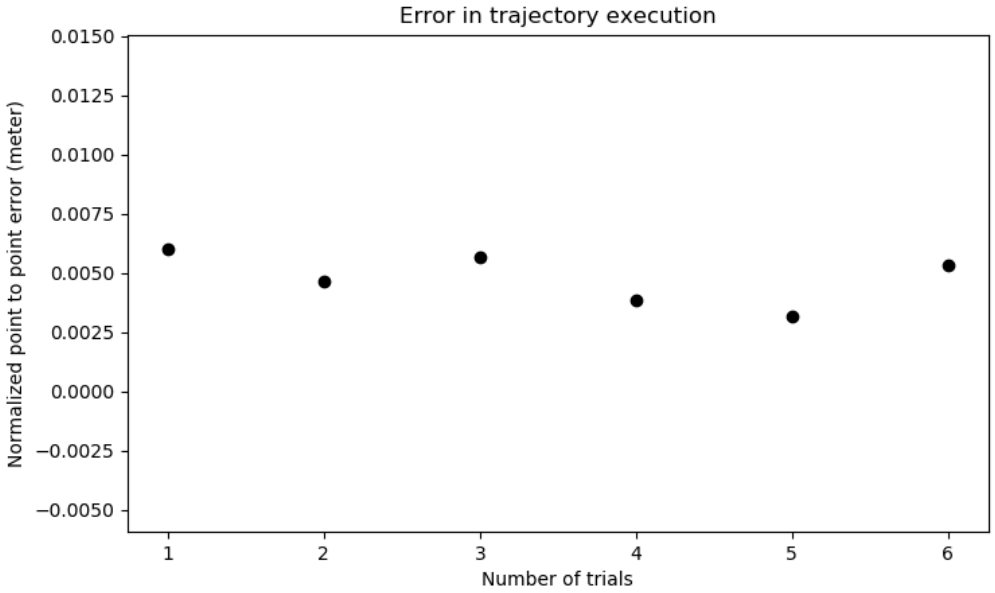

Figure 5.31: Error in execution of grasping trajectories

While performing the experiments, test reports were maintained so that these experiments can be reproduced if needed. Test reports contain all the necessary data describing experiment and experimental conditions. These test reports can be found in Appendix section.

6

Conclusions

In this project, a learning from demonstration architecture was developed for robot programming. Trajectories were demonstrated to the robot by visual demonstrations and control policies were learned using dynamic motion primitives. We provide an intuitive explanation of the working of the dynamic motion primitives. Dynamic motion primitives were, then evaluated for their performance by learning and generalizing an artificially generated step function trajectory. Various parameter were varied in order to evaluate their effect on the performance of the DMP. In next step, we evaluated the performance of the DMP framework along with the trajectory controller on KUKA YouBot manipulator. By observing limitations on the motion of the KUKA YouBot manipulator, we developed the concept of using motion policy encoded in DMP for performing whole body motion control. Developed whole body motion control architecture significantly increased the manipulation capabilities of the mobile manipulators KUKA YouBot and Toyota HSR. Whole body motion control has already been integrated into software stack used on Toyota HSR, where it replaces the MoveIt! framework for motion planning. With all these experiments, we gathered enough insights of the working of DMPs for building a library of DMP and use it for performing robotic tasks in RoboCup@Home scenario.

6.1 Contributions

Contributions of this project include:

- Evaluation of the DMP framework for learning from demonstration.

- Standalone ROS package *ros_dmp* which can be ported easily from one robot to another.
- Development of *whole body motion control framework* for mobile manipulators.
- Replacement for the MoveIt! motion planning framework on Toyota HSR.
- A improved software package for Cartesian velocity control of a manipulator.

By evaluating the DMP framework on robots through various experiments, important observations and conclusions were drawn which are necessary in integrating DMP framework in the RoboCup@Home scenario. These experiments gave important insights, based on which a design of knowledge base and DMP library is possible where a DMP will be learned, stored in the library along with the knowledge associated with it. The knowledge base, then can be used to choose particular DMP suitable for the situation.

DMP-based execution has already been integrated into our domestic repository, namely in the *move_arm* action.[1]

Videos of the experiments on KUKA YouBot[2] and Toyota HSR[3] can be accessed online.

6.2 Future work

While performing various experiments, we realized that the few improvements are necessary in implemented solution, in order to increase the efficiency and usability of it. These improvements are:

- Computed torque control should be used instead of velocity control for better tracking of trajectories.
- Implemented whole body motion control solution assumes the relation between smallest singular value of the manipulator and its manipulation capability to be linear, however this is not the case. A proper relation between the

[1](https://github.com/b-it-bots/mas_domestic_robotics/tree/kinetic/mdr_planning/mdr_actions/mdr_manipulation_actions/mdr_move_arm_action)

[2]https://www.youtube.com/watch?v=jEtlm96KAbA&t=44s

[3]https://www.youtube.com/watch?v=OC7vttt4-Jo

smallest singular value and the manipulation capability of a manipulator is needed to be established.

- Different techniques for the demonstration of the trajectories are needed to be explored so that rotational degrees of the freedom can be demonstrated to the robot and noise in the demonstrated trajectories can be minimized.

- Current implementation of DMP has the ability to accommodate external feedback like on-line goal modification and obstacle avoidance, but these abilities were not evaluated due to the lack of time. A comprehensive evaluation can be done in this direction.

A

Appendix

A.1 Test Reports Of The Experiments

Test report 23.05.2018

Date & Time	23.05.2018, 15:30–18:00
Tester	Abhishek Padalkar
Software versions	ROS Indigo
	OpenCV 3.1.0
Hardware configuration	KUKA youbot 4
	ASUS Xtion Pro Live
Tested feature	Acquisition and repetion of dynamic motion primitives
Test setup / environment	C025 lab, HBRS, RoboCup@Work arena
	15cm platform

Test procedure:

On the robot:

1. Switch on the youbot
2. Launch the robot bringup: `roslaunch mir_bringup robot.launch`
3. Launch moveit: `roslaunch mir_moveit_youbot_brsu_4 move_group.launch`

On an external machine:

1. Export the `ROS_MASTER_URI`: `export ROS_MASTER_URI=http://youbot-brsu-4-pc1:11311`
2. Run the script ***demonstrated_trajectory_recorder.py*** in order to record a trajectory: `rosrun demonstrated_trajectory_recorder demonstrated_trajectory_recorder.py`
3. In the command prompt, press ENTER to start recording; this will open a window that shows the current view of the camera
4. Demonstrate a trajectory by moving the aruco marker board
5. When done demonstrating, press q in the window and write a trajectory name in the command prompt
6. Copy the recorded file to `ros_dmp/data/recorded_trajectories/23_05`
7. Run the script `ros_dmp/src/learn_motion_primitive.py` to learn the weights of the motion primitive; the weights will be saved to `ros_dmp/data/weights/weights_<trajectory_name>.yaml`.

8. For safety reasons, move the robot to open space before repeating any demonstrated trajectories.
9. To repeat a demonstrated trajectory, change the initial and goal position in `ros_dmp/src/test.py` and then run `test.py`. If multiple goal poses are specified, the robot will go to each one of those; however, each pose will first be repeated a given number of times. After finishing each trial, press ENTER to go to the next trial. The data for each trial (both the planned and executed trajectories) are saved to a specified location.

Test assumptions:

- There are no obstacles in the environment
- The demonstrated trajectory is within the manipulator's dexterous workspace
- The camera used for recording the trajectory is calibrated
- The camera frame rate is low (~5fps), so the trajectory has to be demonstrated slowly
- The test environment is static
- Trajectories are recorded in the base link frame
- The joint encoders are working properly and their measurements are considered ground truth values (there is no external robot pose observer, so the encoder measurements are used for measuring the position of the manipulator)

Test 1:

Test objectives:

- Acquiring a demonstrated trajectory (inverse parabolic curve)
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated final position, y same, z same** goal 2: x – 3cm, y same, z same **goal 3: x same, y – 3cm, z same** goal 4: x – 3cm, y + 2cm, z same ** goal 5: x + 3cm, y – 2cm, z same

Test parameters:

- Number of basis functions: 50
- tau = 50
- Number of trials: 10

Observations:

- If some part of a trajectory is out of the dexterous workspace or near its limits (e.g. the goal position is at the limits of the workspace), the motion is unpredictable due to degraded inverse kinematics solutions; this limits the ability to generalise a learned primitive.
- A similar observation holds near the joint limits.
- Since the demonstrated trajectory has initial and final z values that are practically equal to each other, varying z creates infeasible resulting trajectoreis; this primitive is thus limited to scenarios in which the initial and final z position are the same (e.g. moving

an object from one side of a table to another)

Test 2:

Test objectives:

- Acquiring a demonstrated trajectory (step function in Y-Z plane)
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated final position, y same, z same** goal 2: x same, y - 3cm, z same **goal 3: x same, y same , z + 3cm** goal 4: x same, y + 3cm, z + 2cm ** goal 5: x same, y - 5cm, z - 5cm

Test parameters:

- Number of basis functions: 50
- tau = 50
- Number of trials: 10

Observations:

- If some part of a trajectory is out of the dexterous workspace or near its limits (e.g. the goal position is at the limits of the workspace), the motion is unpredictable due to degraded inverse kinematics solutions; this limits the ability to generalise a learned primitive.
- A similar observation holds near the joint limits.
- Since the demonstrated trajectory has initial and final x values that are practically equal to each other, varying x creates infeasible resulting trajectoreis; this primitive is thus limited to scenarios in which the initial and final x position are the same (e.g. cleaning a whiteboard)

General observations

- Increasing tau too much (e.g. 100) causes overshoots at sharp turns.

Test 3:

Test objectives:

- Acquiring a demonstrated trajectory (step function in Y-Z plane)
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated final position, y same, z same** goal 2: x - 2cm, y + 2cm, z -2cm **goal 3: x -5cm, y + 5cm , z - 3cm** goal 4: x same, y + 8cm, z - 1cm ** goal 5: x same, y + 13cm, z same

Test parameters:

- Number of basis functions: 50

- tau = 50
- Number of trials: 10

Observations:

- If some part of a trajectory is out of the dexterous workspace or near its limits (e.g. the goal position is at the limits of the workspace), the motion is unpredictable due to degraded inverse kinematics solutions; this limits the ability to generalise a learned primitive.
- A similar observation holds near the joint limits.
- Since the demonstrated trajectory has initial and final x values that are practically equal to each other, varying x creates infeasible resulting trajectoreis; this primitive is thus limited to scenarios in which the initial and final x position are the same (e.g. cleaning a whiteboard)

General observations

- Increasing tau too much (e.g. 100) causes overshoots at sharp turns.

Test 4:

Test objectives:

- Acquiring a demonstrated trajectory (step function in Y-Z plane)
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated final position, y same, z same** goal 2: x + 2cm, y same, z same **goal 3: x same, y - 2cm , z same** goal 4: x same, y same, z + 3.5cm ** goal 5: x + 2cm, y - 2cm, z - 1cm same

Test parameters:

- Number of basis functions: 50
- tau = 50
- Number of trials: 10

Observations:

- If some part of a trajectory is out of the dexterous workspace or near its limits (e.g. the goal position is at the limits of the workspace), the motion is unpredictable due to degraded inverse kinematics solutions; this limits the ability to generalise a learned primitive.
- A similar observation holds near the joint limits.
- Since the demonstrated trajectory has initial and final x values that are practically equal to each other, varying x creates infeasible resulting trajectoreis; this primitive is thus limited to scenarios in which the initial and final x position are the same (e.g. cleaning a whiteboard)

General observations

- Increasing tau too much (e.g. 100) causes overshoots at sharp turns.

Test report 07.07.2018

Date & Time	07.07.2018, 13:30-18:00
Tester	Abhishek Padalkar
Software versions	ROS Indigo
	OpenCV 3.1.0
Hardware configuration	KUKA youbot 4
	ASUS Xtion Pro Live
Tested feature	Acquisition and repetion of dynamic motion primitives that are partially outside of the robot's dexterous workspace
Test setup / environment	C025 lab, HBRS, RoboCup@Work arena
	lab floor (ground level)

Test procedure:

On the robot:

1. Switch on the youbot
2. Launch the robot bringup: `roslaunch mir_bringup robot.launch`
3. Launch moveit: `roslaunch mir_moveit_youbot_brsu_4 move_group.launch`

On an external machine:

1. Export the `ROS_MASTER_URI`: `export ROS_MASTER_URI=http://youbot-brsu-4-pc1:11311`
2. Run the script ***demonstrated_trajectory_recorder.py*** in order to record a trajectory: `rosrun demonstrated_trajectory_recorder demonstrated_trajectory_recorder.py`
3. In the command prompt, press ENTER to start recording; this will open a window that shows the current view of the camera
4. Demonstrate a trajectory by moving the aruco marker board
5. When done demonstrating, press q in the window and write a trajectory name in the command prompt

6. Copy the recorded file to `ros_dmp/data/recorded_trajectories/23_05`
7. Run the script `ros_dmp/src/learn_motion_primitive.py` to learn the weights of the motion primitive; the weights will be saved to `ros_dmp/data/weights/weights_<trajectory_name>.yaml`.
8. For safety reasons, move the robot to open space before repeating any demonstrated trajectories.
9. To repeat a demonstrated trajectory, change the initial and goal position in `ros_dmp/src/test.py` and then run `test.py`. If multiple goal poses are specified, the robot will go to each one of those; however, each pose will first be repeated a given number of times. After finishing each trial, press ENTER to go to the next trial. The data for each trial (both the planned and executed trajectories) are saved to a specified location.

Test assumptions:

- There are no obstacles in the environment
- The camera used for recording the trajectory is calibrated
- The camera frame rate is low (~5fps), so the trajectory has to be demonstrated slowly
- The test environment is static
- Trajectories are recorded in the base link frame
- The joint encoders are working properly and their measurements are considered ground truth values (there is no external robot pose observer, so the encoder measurements are used for measuring the position of the manipulator)

Test 1:

Test objectives:

- Acquiring a demonstrated trajectory (inverse parabolic curve) which is partially outside of the robot's dexterous workspace
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated final position, y same, z + 15cm** goal 2: x – 5cm, y same, z + 15cm **goal 3: x same, y + 10cm, z + 15cm** goal 4: x + 5cm, y + 5cm, z + 15cm ** goal 5: x + 15cm, y + 15cm, z + 15cm

Test parameters:

- Number of basis functions: 50
- tau = 1
- Number of trials: 10
- A height offset of 15cm is introduced in the z coordinate as the motion is controlled with respect to arm_link_5 (at the mount point of the gripper), but the demonstration was with respect to the tip of the gripper

Observations:

- Since the demonstrated trajectory has initial and final z values that are practically equal

to each other, varying z creates infeasible resulting trajectories; this primitive is thus limited to scenarios in which the initial and final z position are the same (e.g. moving an object from one side of a table to another)

Test 2:

Test objectives:

- Acquiring a demonstrated trajectory (rectangular shape) which is partially outside of the robot's dexterous workspace
- Repeating the demonstrated trajectory with five different goal locations. The different goal locations are specified as follows: **goal 1: x same as demonstrated, y same, z + 15cm** goal 2: x + 2cm, y same, z + 15cm **goal 3: x same, y + 10cm, z + 15cm** goal 4: x + 5cm, y – 5cm, z + 15cm ** goal 5: x – 5cm, y + 15cm, z + 15cm

Test parameters:

- Number of basis functions: 150
- tau = 1
- Number of trials: 10
- An offset of 5cm was introduced to the x position because the initial position of the demonstrated trajectory was very close to the body of the robot
- A height offset of 15cm is introduced in the z coordinate as the motion is controlled with respect to arm_link_5 (at the mount point of the gripper), but the demonstration was with respect to the tip of the gripper

Observations:

- Since the demonstrated trajectory has initial and final z values that are practically equal to each other, varying z creates infeasible resulting trajectories; this primitive is thus limited to scenarios in which the initial and final z position are the same (e.g. moving an object from one side of a table to another)

General observations

- Increasing tau too much (e.g. 100) causes overshoots at sharp turns.

Test report 07.07.2018

Date & Time	07.07.2018, 13:30–18:00
Tester	Abhishek Padalkar
Software versions	ROS Kinetic
	OpenCV 3.1.0
Hardware configuration	Toyota HSR
	ASUS Xtion Pro Live
Tested feature	Acquisition and repetion of a sequence of dynamic motion primitives that are partially outside of the robot's dexterous workspace
Test setup / environment	C069 lab, HBRS, RoboCup@Home arena
	lab floor (ground level)

Test procedure

On the robot

1. Switch on Lucy.
2. On the `lucy` account launch the `move_arm` action: `roslaunch mas_hsr_move_arm_action move_arm.launch` (This will also launch `arm_cartesian_control` `joint_velocity_control` and `moveit`).

On an external machine

1. Export the `ROS_MASTER_URI`: `export ROS_MASTER_URI=http://190.168.50.201:11311`
2. Run the script ***demonstrated_trajectory_recorder.py*** in order to record a trajectory: `rosrun demonstrated_trajectory_recorder demonstrated_trajectory_recorder.py`
3. In the command prompt, press ENTER to start recording; this will open a window that shows the current view of the camera

4. Demonstrate a trajectory by moving the aruco marker board
5. When done demonstrating, press q in the window and write a trajectory name in the command prompt
6. Repeat 3-5 for both trajectories
7. Copy both of the recorded file to `ros_dmp/data/recorded_trajectories/02_08`
8. Run the script `ros_dmp/src/learn_motion_primitive.py` twice to learn the weights of the two motion primitives; the weights will be saved to `ros_dmp/data/weights/weights_<trajectory_name>.yaml`.
9. For safety reasons, move the robot to open space before repeating any demonstrated trajectories.
10. To repeat the demonstrated trajectories, change the initial and goal positions in `ros_dmp/src/test.py` and then run `test.py`. If multiple goal poses are specified, the robot will go to each one of those; however, each pose will first be repeated a given number of times. After finishing each trial, press ENTER to go to the next trial. The data for each trial (both the planned and executed trajectories) are saved to a specified location.

Test assumptions

- There is an obstacle in front of the robot (a small table)
- The camera used for recording the trajectory is calibrated
- The camera frame rate is low (~5fps), so the trajectory has to be demonstrated slowly
- The test environment is static
- Trajectories are recorded in the base link frame
- The joint encoders are working properly and their measurements are considered ground truth values (there is no external robot pose observer, so the encoder measurements are used for measuring the position of the manipulator)

Test objectives

- Acquiring demonstrated trajectories (a straight-line motion and an inverse parabolic curve) which are partially outside of the robot's dexterous workspace
- Repeating the demonstrated trajectories in a sequential manner in order to execute a pick-and-place task (picking an object from a table and then placing it on another higher table).

Test parameters

- Number of basis functions: 50
- tau = 30
- Number of trials: 10

Observations

- Since the demonstrated trajectory has initial and final z values that are practically equal to each other, varying z creates infeasible resulting trajectories; this primitive is thus limited to scenarios in which the initial and final z position are the same (e.g. moving an object from one side of a table to another)

References

[1] *The international journal of robotics research.*

[2] J. Andrew Bagnell. Reinforcement Learning in Robotics: A Survey. *Springer Tracts in Advanced Robotics*, 97:9–67, 2014.

[3] 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.

[4] Arne Böckmann and Tim Laue. Kick motions for the NAO robot using dynamic movement primitives. *arXiv preprint arXiv:1606.00600*, 2016.

[5] Manu Chhabra and Robert A. Jacobs. Learning to Combine Motor Primitives Via Greedy Additive Regression. *Journal of Machine Learning Research*, 9:1535—-1558, 2008.

[6] 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.

[7] Benjamin J Cohen, Sachin Chitta, and Maxim Likhachev. Search-based Planning for Manipulation with Motion Primitives.pdf. pages 2902–2908, 2010.

[8] Benjamin J. Cohen, Gokul Subramanian, Sachin Chitta, and Maxim Likhachev. Planning for manipulation with adaptive motion primitives. *Proceedings - IEEE International Conference on Robotics and Automation*, pages 5478–5485, 2011.

[9] Marc Peter Deisenroth. A Survey on Policy Search for Robotics. *Foundations and Trends in Robotics*, 2(1-2):1–142, 2011.

[10] 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.

[11] Miha Deniša and Aleš Ude. Synthesis of New Dynamic Movement Primitives Through Search in a Hierarchical Database of Example Movements. *International Journal of Advanced Robotic Systems*, 12(10):137, oct 2015.

[12] Arati S Deo and Ian D Walker. Overview of Damped Least-Squares Methods for Inverse Kinematics of Robot Manipulators. *Journal of lntelligent and Robotic Systems*, 14:43–68, 1995.

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

[14] Keith L Doty, Claudio Melchiorri, and Claudio Bonivento. A Theory of Generalized Inverses Applied to Robotics.

[15] Anca D Dragan, Katharina Muelling, J Andrew Bagnell, and Siddhartha S Srinivasa. Movement primitives via optimization. In *Robotics and Automation (ICRA), 2015 IEEE International Conference on*, pages 2339–2346. IEEE, 2015.

[16] Andrej Gams, Bojan Nemec, Auke Jan Ijspeert, and Aleš Ude. Coupling movement primitives: Interaction with the environment and bimanual tasks. *IEEE Transactions on Robotics*, 30(4):816–830, 2014.

[17] Tsutomu Hasegawa, Takashi Suehiro, and Kunikatsu Takase. A Model-Based Manipulation System with Skill-Based Execution. *IEEE Transactions on Robotics and Automation*, 8(5):535–544, 1992.

[18] 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.

[19] Neville Hogan and Dagmar Sternad. Dynamic primitives in the control of locomotion. *Frontiers in Computational Neuroscience*, 7(June):1–16, 2013.

[20] Guy Hotson, Ryan J Smith, Adam G Rouse, Marc H Schieber, Nitish V Thakor, and Brock A Wester. High precision neural decoding of complex movement trajectories using recursive Bayesian estimation with dynamic movement primitives. *IEEE Robotics and Automation Letters*, 1(2):676–683, 2016.

[21] A.J. Ijspeert, Jun Nakanishi, and Stefan Schaal. Movement imitation with nonlinear dynamical systems in humanoid robots. In *Proceedings 2002 IEEE International Conference on Robotics and Automation (Cat. No.02CH37292)*, volume 2, pages 1398–1403, 2002.

[22] 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.

[23] 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.

[24] 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.

[25] Auke Jan Ijspeert, Jun Nakanishi, and Stefan Schaal. Learning Attractor Landscapes for Learning Motor Primitives. *Advances in Neural Information Processing Systems 15 (NIPS2002)*, pages 1547–1554, 2002.

[26] 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.

[27] Minghao Jiang, Yang Chen, Wenlei Zheng, Huaiyu Wu, and Lei Cheng. Mobile robot path planning based on dynamic movement primitives. *2016*

IEEE International Conference on Information and Automation (ICIA), pages 980–985, jul 2016.

[28] Iman Kardan, Iman Kardan, Alireza Akbarzadeh, Alireza Akbarzadeh, Ali Mousavi Mohammadi, and Ali Mousavi Mohammadi. Real-time velocity scaling and obstacle avoidance for industrial robots using fuzzy dynamic movement primitives and virtual impedances. *Industrial Robot: An International Journal*, 2017.

[29] 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.

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

[31] 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.

[32] J. Kober and J. Peter. Policy search for motor primitives in robotics. *Springer Tracts in Advanced Robotics*, 97:83–117, 2014.

[33] 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.

[34] Petar Kormushev, Sylvain Calinon, and Darwin G Caldwell. Robot motor skill coordination with EM-based reinforcement learning. In *IEEE/RSJ 2010 International Conference on Intelligent Robots and Systems, IROS 2010 - Conference Proceedings*, pages 3232–3237, 2010.

[35] Norbert Krüger, Christopher Geib, Justus Piater, Ronald Petrick, Mark Steedman, Florentin Wörgötter, Aleš Ude, Tamim Asfour, Dirk Kraft, Damir Omrčen, Alejandro Agostini, and Rdiger Dillmann. ObjectAction Complexes:

Grounded abstractions of sensorymotor processes. *Robotics and Autonomous Systems*, 59(10):740–757, 2011.

[36] Tomas Kulvicius, Kejun Ning, Minija Tamosiunaite, and Florentin Wörgötter. Joining movement sequences: Modified dynamic movement primitives for robotics applications exemplified on handwriting. *IEEE Transactions on Robotics*, 28(1):145–157, 2012.

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

[38] John C Leonidas, Ver o ppt diretamente, and John C Leonidas. Please wait ... *Radiology*, 242(1):2, 2007.

[39] 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.

[40] 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.

[41] Z. Z. Liu, H. Y. Liu, and Z. Luo. Inverse Kinematics Analysis of 5-DOF Robot Manipulators Based on Virtual Joint Method. *Applied Mechanics and Materials*, 143:265–268, 2012.

[42] Guilherme Maeda, Marco Ewerton, Rudolf Lioutikov, Heni Ben Amor, Jan Peters, and Gerhard Neumann. Learning interaction for collaborative tasks with probabilistic movement primitives. In *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*, pages 527–534. IEEE, 2014.

[43] Simon Manschitz, Jens Kober, Michael Gienger, and Jan Peters. Learning movement primitive attractor goals and sequential skills from kinesthetic demonstrations. *Robotics and Autonomous Systems*, 74:97–107, 2015.

[44] Takamitsu Matsubara, Sang Ho Hyon, and Jun Morimoto. Learning parametric dynamic movement primitives from multiple demonstrations. *Neural Networks*, 24(5):493–500, 2011.

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

[46] Franziska Meier, Evangelos Theodorou, Freek Stulp, and Stefan Schaal. Movement segmentation using a primitive library. In *Intelligent Robots and Systems (IROS), 2011 IEEE/RSJ International Conference on*, pages 3407–3412. IEEE, 2011.

[47] Albert Mukovskiy, Christian Vassallo, Maximilien Naveau, Olivier Stasse, Philippe Soueres, and Martin A Giese. Adaptive synthesis of dynamically feasible full-body movements for the humanoid robot HRP-2 by flexible combination of learned dynamic movement primitives. *Robotics and Autonomous Systems*, 91:270–283, 2017.

[48] 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.

[49] Jun Nakanishi, Rick Cory, Michael Mistry, Jan Peters, and Stefan Schaal. Operational space control: A theoretical and empirical comparison. *International Journal of Robotics Research*, 27(6):737–757, 2008.

[50] S. Nakaoka, A. Nakazawa, K. Yokoi, H. Hirukawa, and K. Ikeuchi. Generating whole body motions for a biped humanoid robot from captured human dances. *2003 IEEE International Conference on Robotics and Automation (Cat. No.03CH37422)*, 3:3905–3910.

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

[52] Scott Niekum, Sarah Osentoski, George Konidaris, and Andrew G. Barto. Learning and generalization of complex tasks from unstructured demonstra-

tions. *IEEE International Conference on Intelligent Robots and Systems*, pages 5239–5246, 2012.

[53] Jacob Nielsen, Anders Stengaard Sørensen, Thomas Søndergaard Christensen, Thiusius Rajeeth Savarimuthu, and Tomas Kulvicius. Individualised and adaptive upper limb rehabilitation with industrial robot using dynamic movement primitives. In *ICRA 2017 Workshop on Advances and challenges on the development, testing and assessment of assistive and rehabilitation robots: Experiences from engineering and human science research*, volume 1, page 40, 2017.

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

[55] 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.

[56] 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.

[57] Peter Pastor, Mrinal Kalakrishnan, Franziska Meier, Freek Stulp, Jonas Buchli, Evangelos Theodorou, and Stefan Schaal. From dynamic movement primitives to associative skill memories. *Robotics and Autonomous Systems*, 61(4):351–361, 2013.

[58] Paul G Pl. Generalisation and Modification of Dynamic Motion Primitives. 26(5):1–2, 2010.

[59] Ruohan Wang, Yan Wu, Wei Liang Chan, and Keng Peng Tee. Dynamic Movement Primitives Plus: For enhanced reproduction quality and efficient trajectory modification using truncated kernels and Local Biases. In *2016*

IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS), pages 3765–3771. IEEE, oct 2016.

[60] Stefan Schaal. Dynamic Movement Primitives -A Framework for Motor Control in Humans and Humanoid Robotics. *Adaptive Motion of Animals and Machines*, pages 261–280.

[61] Stefan Schaal. Is imitation learning the route to humanoid robots? *Trends in cognitive sciences*, 3(6):233–242, 1999.

[62] 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.

[63] Stefan Schaal, Peyman Mohajerian, and Auke Ijspeert. Dynamics systems vs. optimal control - a unifying view. *Progress in Brain Research*, 165(1):425–445, 2007.

[64] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Control, Planning, Learning, and Imitation with Dynamic Movement Primitives. *Workshop on Bilateral Paradigms on Humans and Humanoids, 2003 IEEE International Conference on Intelligent Robots and Systems IROS*, pages 1–21, 2003.

[65] Stefan Schaal, Jan Peters, Jun Nakanishi, and Auke Ijspeert. Learning movement primitives. *Robotics Research*, 15(2005):1–10, 2005.

[66] Aaron D ' Souza, Sethu Vijayakumar, and Stefan Schaal. Learning Inverse Kinematics.

[67] Freek Stulp, Evangelos Theodorou, and Stefan Schaal. Preprint of " Reinforcement Learning with Sequences of Motion Primitives for Robust Manipulation " Reinforcement Learning with Sequences of Motion Primitives for Robust Manipulation. *IEEE Transactions on Robotics*, 28(6):1360–1370, 2012.

[68] Minija Tamosiunaite, Bojan Nemec, Ale Ude, and Florentin Wörgötter. Learning to pour with a robot arm combining goal and shape learning for dynamic movement primitives. *Robotics and Autonomous Systems*, 59(11):910–922, 2011.

[69] Moritz Tenorth, Jan Bandouch, and Michael Beetz. The TUM kitchen data set of everyday manipulation activities for motion tracking and action recognition. In *Computer Vision Workshops (ICCV Workshops), 2009 IEEE 12th International Conference on*, pages 1089–1096. IEEE, 2009.

[70] Aleš Ude, Andrej Gams, Tamim Asfour, and Jun Morimoto. Task-specific generalization of discrete and periodic dynamic movement primitives. *IEEE Transactions on Robotics*, 26(5):800–815, 2010.

[71] Aleš Ude, Bojan Nemec, Jun Morimoto, and Others. Trajectory representation by nonlinear scaling of dynamic movement primitives. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 4728–4735. IEEE, 2016.

[72] 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.

[73] Janson Wolfe, Bhaskara Marthi, and Stuart Russell. Combined task and motion planning for mobile manipulation. *International Conference on Automated Planning and Scheduling*, (Icaps):254–257, 2010.

[74] Eric Wolter and Thorsten Baark. Learning to bounce a ball with a robotic arm. In *Conference/Book Title Proceedings of Projektpraktikum Robot Learning.*

[75] You Zhou, Martin Do, and Tamim Asfour. Coordinate Change Dynamic Movement PrimitivesA leader-follower approach. In *Intelligent Robots and Systems (IROS), 2016 IEEE/RSJ International Conference on*, pages 5481–5488. IEEE, 2016.