# Learning Multi-Goal Reachability in a Humanoid Robot using Deep Reinforcement Learning

Thesis submitted in partial fulfillment
of the requirements for the degree of

*Master of Science by Research*
*in*
*Electronics and Communications Engineering*

by

Phaniteja Singamaneni
201231009
singamaneniphani.teja@research.iiit.ac.in

ROBOTICS RESEARCH CENTER
International Institute of Information Technology
Hyderabad - 500 032, INDIA
May 2018

International Institute of Information Technology
Hyderabad, India

# CERTIFICATE

It is certified that the work contained in this thesis, titled ”Learning Multi-Goal Reachability in a Humanoid Robot using Deep Reinforcement Learning” by Phaniteja Singamaneni, has been carried out under my supervision and is not submitted elsewhere for a degree.

| | |
|---|---|
| Date | Adviser: Prof. K. Madhava Krishna |
| Date | Adviser: Prof. Abhishek Sarkar |

To my mother and my sister who stood by my side always

# Acknowledgments

Robotics Research Center had been an integral part of my college life. Besides learning robotics, I learnt many life lessons and made friends for life at RRC. I sincerely extend my gratitude to Dr. K. Madhava Krishna for giving me an opportunity to work at his lab and for his support and guidance through out my research. I thank my guide Dr. Abhishek Sarkar for his guidance during my thesis. I also take this opportunity to thank Dr. Suril V. Shah for his support and inspiration whenever I approached him with a doubt. I am also grateful to Dr. Ravidran Balaraman for aspiring me in the field of Reinforcement Learning and made this work possible. I am deeply grateful to Parijat Dewangan, Divyanshu Goel and Pooja Guhan for their unwavering support without which none of this was remotely possible. I am also grateful to all the people at RRC who had been with me in times of thick and thin and always lent a helping hand - Sri Harsha Turlapati, Mihir Shah, Vignesh Prasad, Nazrul, Yaswanth, Roopal, Rishab Khawad, Meha Kaushik, Nayan Joshi, Mahtab, Josyula Gopala Krishna, Saket, Bhargav, Venugopal. I express my profound gratitude to my family for believing in me and supporting me in times dire and tough. Specifically I am grateful to my sister, Anusha Singamaneni, for bearing me all through my tough times and making me move forward. I would also like to extend my sincere thanks Sravya Gurram, Bharath Kota, Srikar Allu and Suresh Santhanam for being there for me always and supporting me throughout the journey. Finally I would like to acknowledge all the people who were a part of this journey and become good friends in this process.

# Abstract

Inverse Kinematics (IK) problem deals with finding an appropriate actuator configuration that makes the end effector Cartesian coordinates and pose match the given coordinates and pose. In most of the cases, it is enough if we can solve IK for the given coordinates whatever may be the pose. Real time calculation of inverse kinematics with dynamically stable configuration is of high necessity in humanoid robots as they are highly susceptible to lose balance. General Inverse Kinematic solvers may not guarantee real-time control of the end-effectors in external coordinates along with maintaining stability. This work proposes a methodology to generate joint-space trajectories of stable configurations for solving inverse kinematics using Deep Reinforcement Learning (RL). Our approach is based on the idea of exploring the entire configuration space of the robot and learning the best possible solutions using an actor-critic based policy learning in continuous action spaces, Deep Deterministic Policy Gradient (DDPG). The proposed strategy was evaluated on the highly articulated upper body of a humanoid model with 27 degree of freedom (DoF). The trained model was able to solve inverse kinematics for the end effectors with 90% accuracy while maintaining the balance in double support phase.

Following the success of learning a general IK solver for goal reachability tasks of a single hand of humanoid, a more challenging problem of solving IK for both hands of the humanoid simultaneously is taken up. For addressing this problem, the proposed methodology was extended with significant changes in the state vector and reward function modelling. The extended strategy was then evaluated on the highly articulated upper body of the given humanoid for learning multi-goal reachability tasks of both hands along with maintaining stability in double support phase. Results show that the trained model was able to solve inverse kinematics for both the hands, where the articulated torso contributed to both the tasks. However it was observed that DDPG was unstable and in some cases even though it was stable initially, the cumulative reward started degrading later. Consequently, this works moves on to address these instability issues by using multi-task reinforcement learning.

However, most reinforcement learning algorithms are inefficient for learning multiple tasks in complex robotic systems, where actions between different tasks are not fully separable and share a set of actions. In case of tasks where actions are fully separable, each task policy can be learnt independently using different policy networks. However when the actions are not fully separable, the policies cannot be learnt independently. In such environments a compound policy may be learnt with shared neural network parameters, which performs multiple tasks concurrently. However such compound policy may get biased towards a task or the gradients from different tasks negate each other, making the learning

unstable and sometimes less data efficient. In this work, we propose a new approach for simultaneous learning of multiple tasks sharing a set of common actions in continuous action spaces, which we call as DiGrad (**Di**fferential Policy **Grad**ient). The proposed framework is based on differential policy gradients and can accommodate multi-task learning in a single actor-critic network. We also propose a simple heuristic in the differential policy gradient update in case of partially separable actions to further improve learning. In order to show the efficiency of framework, the proposed architecture was tested on the given humanoid for learning multi-goal reachability tasks (each end effector has a different goal) and the results were compared with that of DDPG. Training results show that DiGrad converges faster than DDPG and removes the instability in training. Finally a comparative analysis with different settings in DiGrad along with DDPG was shown.

# Contents

# List of Figures

# List of Tables

*Chapter 1*

# Introduction

In robotic systems, the tasks are usually defined in coordinate space, whereas the control commands are defined in actuator space. In order to perform task level robot learning, an appropriate transformation from coordinate space to actuator space is required. If the intrinsic coordinates of a manipulator are defined as a vector of joint angles $\boldsymbol{\theta} \in \mathbf{R^n}$, and the position and orientation vector of the end effector as a vector $\mathbf{x} \in \mathbf{R^m}$, then the forward kinematics function can be given by the following equation

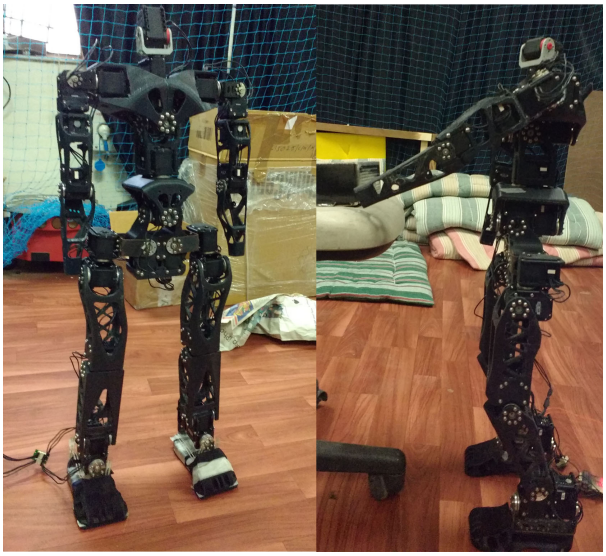
$$\mathbf{x} = f(\boldsymbol{\theta}) \tag{1.1}$$

The inverse kinematics problem [2,3] is to find a mapping from the end-effector coordinates to actuator space which can be represented as

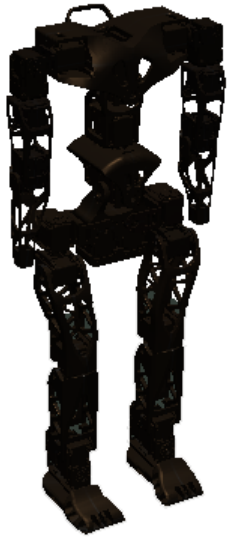
$$\boldsymbol{\theta} = f^{-1}(\mathbf{x}) \tag{1.2}$$

For redundant robotic systems, that is, when the dimension of the task-space is smaller than the dimension of the joint-space $(n > m)$, $f^{-1}(.)$ is not a unique mapping. Given a task-space position $\mathbf{x}$, there can be many corresponding joint-space configurations of $\boldsymbol{\theta}$. Thus, learning inverse kinematics relates to learning multi-valued function. Some applications where IK is used include pick and place [4], physics engines [5–9] and human-robot interactions like tele-operating a robot to grasp objects [10], or execute a series of coordinated gestures [11, 12].

Inverse kinematic approaches can be broadly divided into two categories, namely closed-from analytical methods and numerical methods. In analytical methods, the joints positions are written as mathematical equations in terms of joint angles and solving inverse kinematics deals with putting the equations in closed form and solving them. However this may not be practically feasible when the system equations are very complex. Numerical methods offer a very good alternative in such scenarios by making use of simple linear algebraic approaches. Some examples of numerical methods are BFGS [13], pseudo-Jacobian inverse [2, 14, 15], Jacobian transpose [2, 15, 16], Damped Least Square method (DLS) [15, 17], and Cyclic Coordinate Descent (CCD). In Jacobian based methods or BFGS, the time for the calculation of IK mainly depends on the time for calculation of matrix transpose or pseudo inverse. Unlike closed-form analytical methods, the convergence time of numerical methods may vary and the results may not be repeatable (CCD). Furthermore, articulated structures having more



(a) Real Robot



(b) Simulated Robot

Figure 1.1: 27 DoF humanoid robot with articulated torso.

than six DoF might not have a closed form solution. One of the main problems with numerical solvers like [13–15, 17] is that they provide unique solutions without exploring the redundancy existing in the robots. This limits their application to humanoid robots where constraints like stability and self collision avoidance has be ensured along with the inverse kinematic solution. Several redundancy resolution methods [18, 19] had been proposed in the case of manipulators, but these methods are usually computationally expensive due to the optimizations involved (null space optimization, torque optimization etc.). Hence the proposed methods may not yield a real time solution due to presence of large number of DoF in humanoid robots.

In complex and redundant robotic systems like humanoids, finding an inverse kinematic solution is not straightforward and many a times it is a highly ill-posed problem. This is because of the non-uniqueness of IK solution. Also, in humanoids multiple kinematic chains exist and hence in any existing IK methods, each chain is solved independent of the other chain. However, this assumption may lead to non-practical configurations or imbalance and hence there is need to devise a method, that takes all the kinematic chains into account, even while solving for a single end effector. A much more complex problem to explore in this setting is to solve IK for multiple end effectors simultaneously. One direction would be using the above mentioned solvers in sequential manner at each step of IK. In order to enable simultaneous solving of multiple end effectors in the presence of constraints, augmented Jacobian [20] and extended Jacobian [21] were proposed. However these methods suffer from algorithmic singularities [22] apart from above mentioned limitations. Learning based methods [23] serve as a very good alternative that can overcome these limitations as well as solve the IK in real time satisfying the required constraints. Especially, reinforcement learning [1] provides a suitable framework for exploring the entire solution space and learning a generalized IK solver.

Some of the recent advancements in RL like Deep $Q$-learning (DQN) [24], Guided Policy Search [25], Trust region policy optimization [26] and Deep Deterministic Policy Gradient (DDPG) [27] provide us many frameworks for learning a generalized IK solver. Among these frameworks, DDPG was proposed in the context of learning continuous control tasks and hence it becomes a natural choice to solve the problem of IK. Reinforcement Learning works on the experienced data, and thus would avoid problems due to matrix inversions which may occur while solving general inverse kinematics. Therefore, learning would never demand impossible postures which occurs due to ill-conditioned matrix inversions. Also, the most important criteria for a humanoid robot is stability and hence IK solver should ensure that this is not violated in the final solution. This can be easily incorporated into any RL framework by simply adding a large negative reward when the robot becomes unstable. The authors of DDPG applied the framework to learn manipulation, reachability and locomotion in simple MuJoCo [5] environments, extending its scope to robotic control. In a later work, this algorithm clubbed with asynchronous updates [28] was used to solve the problem of door opening using a six DoF manipulator and the final results were demonstrated on a real robot. However there is no work that learns these kind of manipulation or reachability tasks in humanoid robots.

The main focus of this work is on learning an efficient IK solver using deep RL for reachability tasks in a humanoid robot. Following points are addressed in this work in the context of humanoid robots:

1. Modelling of Robot and RL environment for learning an IK solver that addresses reachability tasks of humanoid robot in double support phase [29, 30].

2. A deep RL (DDPG) based framework for learning IK solver which provides stable and self-collision free joint trajectories besides the final configuration.

3. Multi-goal IK solver that can simultaneously solve IK (along with joint trajectories) of both arms of humanoid. In this context two types of frameworks are proposed:

   (a) An extension to the existing DDPG based framework.

   (b) A novel RL framework, DiGrad.

Zero Moment Point (ZMP) [30, 31] is used as the measure of stability and self-collision detection is performed using the bounding box approach [32] in this work. The humanoid robot used in this work is shown in Fig. 1.1.

This chapter discusses the technologies involved and methods that are used in the subsequent chapters. Section 1.1 explains Markov Decision (MDP) process and RL followed by Policy Gradient and Actor-Critic Algorithm in Section 1.2. Further Deterministic Policy Gradient and DDPG are discussed in Sections 1.3 and 1.4 respectively.

## 1.1 Markov Decision Process

Markov Property states that the effect of an action taken in a state is dependent only on the present state and not on any of the previous states in the history. A decision problem that is modelled mathematically assuming Markov Property is called a Markov Decision Process [33] (MDP). A formal definition of MDP is a tuple $(S, A, P, r, \gamma)$, where

- $S$ is a finite set of states
- $A$ is a finite set of actions
- $P$ is a state transition probability matrix that states the probability of reaching state $s'$ after performing an action $a$ in the state $s$,
  $P(s'|s, a) = \mathbb{P}[s_{t+1} = s'|s_t = s, a_t = a]$
- $r$ is a reward function,
  $r(s, a) = \mathbb{E}[r_{t+1}|s_t = s, a_t = a]$ where $\mathbb{E}$ denotes the estimated value.
- $\gamma$ is a discount factor, $\gamma \in [0, 1]$

A state captures the information that is available at any given time, $t$ about its environment. The set of actions can be either discrete or continuous. In this work, we consider only continuous actions. In the literature that follows, state-action pairs are considered as feature vectors.

### 1.1.1 Return

The return $G_t$ is the total discounted reward from time step t,

$$G_t = \sum_{t=0}^{\infty} \gamma^t r_{t+1}(s_t, a_t). \tag{1.3}$$

### 1.1.2 Policy

A Policy $\pi$ fully defines the behaviour of an agent. It is used to choose an action in the given state to reach the next state. It can be either stochastic or deterministic. A stochastic policy $\pi$ is a probability distribution over actions given states,

$$\pi(a|s) = \mathbb{P}[a_t = a|s_t = s].$$

A deterministic policy is a mapping from state to action, $\pi : S \to A$,

$$\pi(s) = a.$$

### 1.1.3 State Value Function

The state-value function $V_\pi(s)$ of an MDP is defined as the expected return starting from state $s$, and then following policy $\pi$,

$$V_\pi(s) = \mathbb{E}_\pi[G_t|s_t = s].$$

### 1.1.4 Action Value Function

The action-value function $Q_\pi(s, a)$ or $Q$-function or $Q$-value is defined as the expected return starting from state $s$, performing an action $a$, and then following the policy $\pi$,

$$Q_\pi(s, a) = \mathbb{E}_\pi[G_t|s_t = s, a_t = a]$$

The optimal value function in any MDP is the maximum value-function over all the policies. It is denoted by $V^*$ (or $Q^*$). The optimal policy can be found by maximising over $Q^*(s, a)$.

## 1.2 Reinforcement Learning

Reinforcement learning [34] (RL) is an area in machine learning where an agent ought to take actions in an $environment$ in order to maximise some notion of cumulative $reward$. RL is a method in which an agent is made to learn from the through experiences without the use of any external data. The $environment$ in RL is typically formulated as an MDP.

In a standard reinforcement learning setup, an agent interacts with the environment $E$ in discrete time steps. At each time step $t$, the agent takes a state $s_t \in S$ as input and performs an action $a_t \in A$ and receives the observations. These observations consist of reward, denoted by $r_t$ ($\in R$) and the future state, denoted by $s'_t$. The actions are taken according to the policy $\pi : S \to A$. In all the environments considered the actions are real valued $a_t \in \mathbb{R}^N$. In general, the environments may be partially observed so that the entire history of observations and action pairs i.e., $s_t = \{x_1, a_1, ..., a_{t-1}, x_t\}$ may be required to describe the state. However, here, we assumed that the environment is fully-observed so $s_t = x_t$

The environment $E$, is modelled as a fully observable MDP [1] with state space $S$, action space $A \in \mathbb{R}^N$, a reward function $r(s_t, a_t)$ and and transition dynamics $p(s_{t+1}|s_t, a_t)$. The initial state, $s_1$ is assumed to be drawn from distribution $p(s_1)$. We also assume an infinite horizon with a discount factor $\gamma$. The discounted state visitation distribution for a policy $\pi$ is denoted by $\rho^\pi$. In this work, we consider only deterministic policies, i.e., $\pi(s_t) = a_t$.

### 1.2.1 SARSA and Q-learning

The goal in reinforcement learning is to learn a policy $\pi$ which maximizes the expected return $G_t$. The effectiveness of an action chosen using policy $\pi$ is evaluated using action-value function, $Q(s_t, a_t)$.

$Q$-value which is used in many of the RL algorithms is defined as the expected return after taking an action $a_t$ in the state $s_t$ and thereafter following the given policy $\pi$. The reward $r_t$ and the next-state $s_{t+1}$ are dependent only on the environment and hence the action-value function can be written as:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{r_t, s_{t+1} \sim E}[r_t | s_t, a_t].$$

Thereby it can be said that maximizing the action-value function results in maximization of the expected return. Hence Reinforcement Learning algorithms like Dynamic Programming (DP), Monte Carlo (MC) and Temporal Difference (TD) have the notion of maximising $Q$-value at one point or the other.

Specifically, TD learning has the characteristics of both MC and DP. It can learn from raw experiences, without a model of the environment like MC and it uses DP like approaches for action-value updates. Suppose an action $a$ is performed in state $s$ and a reward $r$ and new state $s'$ are obtained. Let $a'$ be the action that needs to be performed in state $s'$ according to policy $\pi$, which was used to choose action $a$, then the TD update is given by:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma Q(s', a') - Q(s, a)], \tag{1.4}$$

where $\alpha$ is the learning rate. This is the simplest form of TD leaning and is called SARSA (State-Action-Reward-State-Action). In SARSA, the next action $a'$ is sampled using the current policy $\pi$ and hence it is called an on-policy learning. Therefore it can learn only the value function for a given policy unlike $Q$ Learning which learns the policy.

From Eq. (1.4), it can be observed that the expectation depends only on the environment and hence $Q^{\pi}$ can be learnt off-policy, using transitions from a different behavioural policy $\beta$:

$$Q^{\pi}(s_t, a_t) = \mathbb{E}_{s \sim \rho^{\beta}, a_t \sim \beta, r_t \sim E}[r_t | s_t, a_t].$$

$Q$ Learning is one such off-policy algorithm that uses a greedy policy $\pi(s) = \arg\max_a Q(s, a)$ with the following value function update:

$$Q(s, a) \leftarrow Q(s, a) + \alpha[r + \gamma \max_{a'} Q(s', a') - Q(s, a)], \tag{1.5}$$

where $a'$ is sampled using the behaviour policy $\beta$ and $\alpha$ is learning rate.

**Value Function Approximation**

In the presence of large state-action spaces, the lookup table approach [1] usually slows down the learning process. Therefore, to scale up the model-free methods for large state-action spaces, function approximations are used for value functions $\hat{v}(s, w) = v_{\pi}(s)$ (or $\hat{q}(s, a, w) = q_{\pi}(s, a)$). The use of large, non-linear function approximators for learning value or action-value functions has often been avoided in the past since theoretical performance guarantees are impossible, and practically learning tends to be unstable. Recently, [35] adapted the Q-learning algorithm in order to make effective use

of large neural networks as function approximators. Their algorithm was able to learn to play Atari games from pixels. In order to scale Q-learning they introduced two major changes: the use of a $replay\ buffer$, and a $separate\ target\ network$ for calculating target $y_t = r + \gamma \max_{a'} Q(s', a')$

## 1.3 Policy Gradient and Actor-Critic Algorithm

Policy based reinforcement learning is an optimization problem. Consider a parametrised policy $\pi_\theta(s, a)$, with parameters $\theta$. Let the quality of the policy $\pi_\theta$ is evaluated using an objective function $J(\theta)$, which is usually a value function (can be either start value, average value or average reward per time step). In order to learn an optimal policy, the objective function $J(\theta)$ needs to be maximised given $\theta$ i.e., arg $\max_\theta J(\theta)$.

Among several methods which can be used to solve this optimization problem, Policy Gradient algorithms search for a local maximum in $J(\theta)$ by ascending the gradient of the policy with respect to the parameters $\theta$:

$$\Delta\theta = \alpha \nabla_\theta J(\theta) \tag{1.6}$$

where $\nabla_\theta J(\theta)$ is the Policy Gradient and $\alpha$ is a step-size parameter.

### 1.3.1 Policy Gradient Theorem

For any differentiable stochastic policy $\pi_\theta(s, a)$, for any of the policy objective functions $J(\theta)$, the policy gradient is

$$\nabla_\theta J(\theta) = \mathbb{E}_{\pi_\theta}[\nabla_\theta \log \pi_\theta(s, a) Q^{\pi_\theta}(s, a)]. \tag{1.7}$$

This theorem [1] can be used only in case of the stochastic policies, deterministic case is discussed in the subsequent subsections.

### 1.3.2 Actor-Critic Algorithm

An actor-critic based policy learning algorithm implements a generalized policy iteration that alternates between a policy evaluation and a policy improvement step. The critic evaluates the current policy and the actor tries to improve the current policy using feedback from the critic. This process is shown diagrammatically in Fig 1.2.

A simple actor-critic algorithm based on action-value critic and using a linear function approximation $Q_w(s, a) = \phi(s, a)^T w$, where $w$ are weights, is shown in Algorithm 1.

### 1.3.3 Deterministic Policy Gradient Theorem

Consider a deterministic policy $\mu_\theta : S \to A$ with parameter vector $\theta \in \mathbb{R}^N$. Define a probability distribution $p(s \to s', t, \mu)$ and discounted state distribution $\rho^\mu(s)$ analogously to the stochastic case.

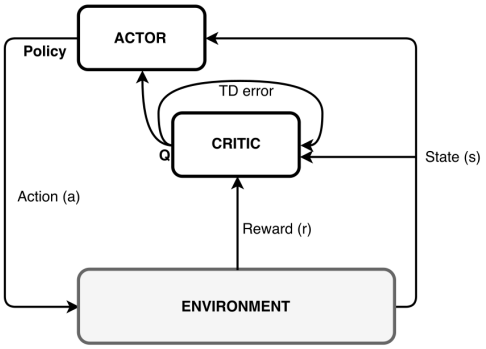



Figure 1.2: Actor Critic Algorithm. Actor (Policy) is updated using Policy Gradient. Critic (Q) is updated using Temporal Difference (TD) [1] method.

**Algorithm 1** Actor-Critic

Initialize $s$, $\theta$
Sample $a$ according to $\pi_\theta$
**for** each step **do**
  Perform action $a$ and get the reward $r$ and transition $s'$.
  Sample action $a'$ using policy $\pi_\theta$.
  $\delta = r + \gamma Q_w(s', a') - Q_w(s, a)$.
  $\theta = \theta + \nabla_\theta \log \pi_\theta(s, a) Q_w(s, a)$.
  $w \leftarrow w + \beta \delta \phi(s, a)$.
  $a \leftarrow a'$, $s \leftarrow s'$
**end for**

Then the performance objective can be written as,

$$J(\mu_\theta) = \int_S \rho^\mu(s) r(s, \mu_\theta(s))\, ds = \mathbb{E}_{s\sim\rho^\mu}[r(s, \mu_\theta(s))].$$

**Deterministic Policy Gradient Theorem** [36]
If $p(s'|s,a), \nabla_a p(s'|s,a), \mu_\theta(s), \nabla_\theta \mu_\theta(s), r(s,a), \nabla_a r(s,a), p1(s)$ are continuous in all parameters and variables s, a, s' and x, then $\nabla_a Q^\mu(s,a)$ exists and the deterministic policy gradient exists and is given by:

$$\nabla_\theta J(\mu_\theta) = \int_S \rho^\mu(s) \nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}\, ds = \mathbb{E}_{s\sim\rho^\mu}[\nabla_\theta \mu_\theta(s) \nabla_a Q^\mu(s,a)|_{a=\mu_\theta(s)}]. \quad (1.8)$$

This theorem lets us define a policy gradient for deterministic cases and hence solving a problem with continuous actions like robotic control becomes feasible.

## 1.4 Deep Deterministic Policy Gradient (DDPG)

Deep Deterministic Policy Gradeint (DDPG) algorithm uses the underlying idea of DQN in the continuous state-action space. It is an actor-critic policy learning [37] method with added target networks

to stabilize the learning process. DDPG uses experience replay which addresses the issue of data being dependent and non-identically distributed as most optimization algorithms need samples that are identical and independently distributed. Transitions are sampled from the environment according to the given exploration policy and the tuple $(s_t, a_t, r_t, s_{t+1})$ are stored in a replay buffer of finite size. When this buffer becomes full, oldest samples are discarded. A mini-batch of samples, $m_b$ is used to update the network. The critic network ($\theta^{\mu}$) is updated using the same loss function as in $Q$-learning [38], and the actor network ($\theta^{Q}$) is updated using the DPG as in Eq. (1.8). In order to avoid the divergence of neural networks in $Q$-learning, target networks($\theta^{\mu'}$, $\theta^{Q'}$) are used which track the original networks slowly.

**Algorithm 2** DDPG

Randomly initialize critic network $Q$ and actor network $\mu$ with weights $\theta^{Q}$ and $\theta^{\mu}$ respectively.
Initialize target networks $Q'$ and $\mu'$ with weights $\theta^{Q'} \leftarrow \theta^{Q}$, $\theta^{\mu'} \leftarrow \theta^{\mu}$
Initialize replay buffer R.
**for** $ep = 1$ to MaxEpisodes **do**
  Initialize a random process $N$ for action exploration.
  Receive initial state $s_1$.
  **for** $t = 1$ to MaxStep **do**
    Select an action $a_t = \mu(s_t|\theta^{\mu}) + N_t$ as per the current policy and exploration noise.
    Execute the action $a_t$ and observe reward $r_t$ and new state $s_{t+1}$
    Store transition $(s_t, a_t, r_t, s_{t+1})$ in R.
    **if** (Size(R) $>$ batch_size) **then**
      Sample a mini-batch of $M$ transitions from R.
      Update critic by minimizing following loss:

$$\frac{1}{M}\sum_{i}(Q(s_i, a_i|\theta^{Q}) - r_i(s_i, a_i) + \gamma Q'(s_{i+1}, \mu'(s_{i+1}|\theta^{\mu'})|\theta^{Q'}))^2.$$

      Update actor using the policy gradient:

$$\theta^{\mu} = \theta^{\mu} + \eta\frac{1}{M}\sum_{i}\nabla_a Q(s_i, a_i|\theta^{Q})|_{a_i=\mu(s_i|\theta^{\mu})}\nabla_{\theta^{\mu}}\mu(s_i|\theta^{\mu}).$$

      Update the target networks as given in Eq. (1.12)
    **end if**
    **if** (terminal_state) **then**
      break
    **end if**
  **end for**
**end for**

Suppose $Q(s, a|\theta^{Q})$, $\mu(s|\theta^{\mu})$ represent critic and actor networks respectively and $Q'(s, a|\theta^{Q'})$, $\mu'(s|\theta^{\mu'})$ represent their corresponding target networks. Given this parametrization, the critic network weights are optimized by minimizing the following loss function:

$$L(\theta^{Q}) = (Q(s_t, a_t|\theta^{Q}) - y_t)^2 \tag{1.9}$$

where,

$$y_t = r(s_t, a_t) + \gamma Q'(s_{t+1}, \mu'(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{1.10}$$

The update on actor network with learning rate is $\eta$ is given by:

$$\theta^{\mu} = \theta^{\mu} + \eta \nabla_a Q(s_t, a_t|\theta^Q)|_{a_t=\mu(s_t|\theta^{\mu})} \nabla_{\theta^{\mu}} \mu(s_t|\theta^{\mu}). \tag{1.11}$$

The target networks follow slow updates according to Eq. (1.12), where $\tau << 1$.

$$\theta^{Q'} \leftarrow \tau\theta^{Q} + (1-\tau)\theta^{Q'} \qquad \theta^{\mu'} \leftarrow \tau\theta^{\mu} + (1-\tau)\theta^{\mu'} \tag{1.12}$$

As it can be observed from Eq. (1.10), reward function is an integral part of the network update and hence the underlying policy that is learnt by the network. Therefore reward function should be modelled carefully so that the RL agent learns the policy correctly. The entire algorithm for DDPG is shown in Algorithm 2.

The distribution of consecutive chapters is as follows. The model of the humanoid robot used in this work is explained in Chapter-2 along with its kinematic model. Chapter-3 explains RL based IK solver for single hand reachability goals along with maintaining stability. Chapter-4 explains the extension of the presented IK solver to dual goal reachability tasks using both hands. A novel RL framework to learn dual goal reachability tasks in a robust manner is presented in Chapter-5 followed by Conclusions in Chapter-6.

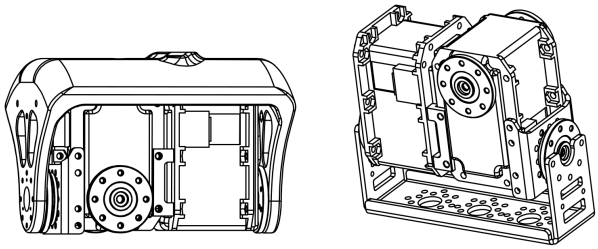
*Chapter 2*

# Humanoid Robot Model

This chapter discusses about the details of the 3D printed model of the robot and the derivation of the mathematical model that was used in rest of the of this work. The sections are organised as follows. Section 2.1 describes the humanoid model followed by Kinematic model of the robot in 2.2. In the following section the calculation of Zero Moment Point (ZMP) is discussed along with an algorithm to calculate the inverse dynamics.

## 2.1 Robot Model

The overall humanoid platform is small, with total height 84 cm, total weight under 5 Kg and 27 DoF. The entire robot is 3D printed with PLA material which has density 1250 kg/m3, breaking tensile strength is 48.8 MPa and thus it is sturdy without adding up too much weight to the total structure keeping it lightweight. The humanoid body is divided into different sections - legs, hands, torso and head. Shoulder has two motors emulating universal joints and elbow too has a similar setup of universal joint. The lower body can be considered as a single entity, without upper body, allowing us to simplify the planning for locomotion. Thus, we can see the robot in two halves maintaining CoM of each separately and easily.



(a) (b)

Figure 2.1: Universal joint at (a) abdomen (b) ankle

The torso has five joints emulating a simpler vertebral column. As designed in Poppy, the vertebral column can be viewed as a joining link between shoulders and pelvis having two universal joints, con-

taining two revolute joints as shown in Fig. 2.1(a) and a single revolute joint in the middle. These five joints allow us to free the lower body from the constraints of the upper body and adjust the CoM without solely relying on the pelvic joint. Additionally this design also increases the effective work space of the entire arm assembly and brings it one step closer to achieve universal reach within the workspace in order to pick and place objects. In this work, we have tried to explore the usage of articulated torso for performing reachability tasks with one or both hands.

Each leg emulates a spherical joint at the hip by using three servo motors, a single revolute joint at the knee and emulating a universal joint at ankle allowing for sagittal and frontal motion. Since there is no universal joint in the ankle of Poppy it is difficult to adapt to uneven terrain. So we added an extra DoF at both the ankles as shown in Fig. 2.1(b).

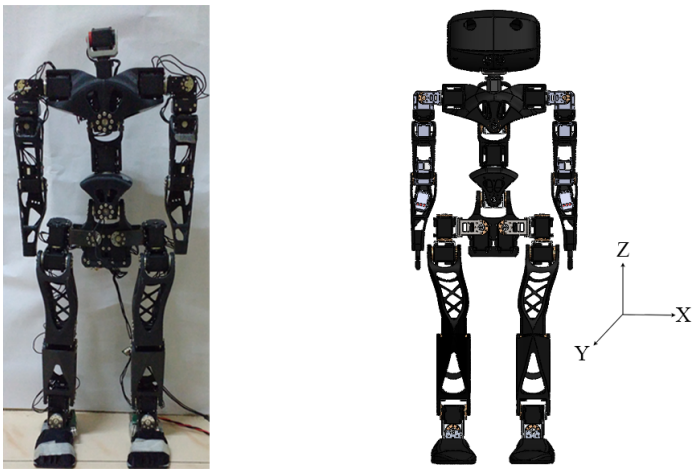



Figure 2.2: 3D printed Humanoid (left) and CAD model (right)

### 2.1.1 Changes from the design of Poppy

The problem with Poppy robot was that the motor output torque was low and as a result the robot was unable to stand properly if the links were changed or made heavier. However, as done in Poppy we are using servo motors but in order to increase the capacity of the robot, various motors of Poppy are replaced with the higher torque variety model. The motors used in the modified humanoid are Dynamixel MX-64 (6 N-m torque) instead of Dynamixel MX-28 (2.5 N-m torque) model. The motor has contact-less magnetic sensor of 0.1° resolution and can give feedback of angular velocity and angular acceleration directly. It also has in-built PID controller and the global feedback loop gives us angular position, velocity and acceleration of motors. Therefore, compliance of the motor joints can be simulated by changing the PID values [39].

The above discussed change also constitutes changes in geometry and design of the whole body to make space for these motors and make the links stronger for higher load carrying capacity, while still retaining the necessary bend in the legs [40], which allows closer placement of CoM in the support polygon for

greater stability. The foot thickness was very small in the original Poppy design. We have redesigned the foot as a firm sole. We also have separated the toe and introduced a torsion spring to allow for elastic property in foot. We introduced an extra DoF at ankles allowing motion in frontal plane as well, which was not originally present in Poppy thus allowing to tilt the robot sideways.

The arms were also modified in order to have an actuated hand controlled by a single motor in each hand to grasp small objects. The backlash problem in servo at ankle and knee joints in humanoid in Poppy is solved by using springs but we have not used them as its negligible for small links. The flexibility of our structure allows us to do more complex tasks in complex environments as it allows the robot to adapt to the environment. In comparison to other similar robots, the presented design is simple, lightweight, and easily modifiable allowing change of links, in case we need to update them.

## 2.2 Forward Kinematic Model

The forward kinematic model of the robot is represented using Denavit-Hartenberg (DH) convention of attachment of reference frames with the base frame at the right leg sole and the first joint angle starting from right ankle. The DH representation of joint frames causes a considerable amount of streamlining and simplification of the forward kinematics calculation. In this convention, each [1]homogeneous transformation $T_i$ is represented as a product of four basis transformations.

$$T_i = Rot_{z_i,\theta_i} Trans_{s_i,d_i} Trans_{x_i,a_i} Rot_{x_i,\alpha_i} \tag{2.1}$$

$$= \begin{bmatrix} cos(\theta_i) & -sin(\theta_i) & 0 & 0 \\ sin(\theta_i) & cos(\theta_i) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & a_i \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & cos(\alpha_i) & -sin(\alpha_i) & 0 \\ 0 & sin(\alpha_i) & cos(\alpha_i) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

where $a_i$ is called the **length**, $d_i$ is called the **offset**, $\alpha_i$ is called the **twist**, $\theta_i$ is called the **angle**. These four quantities are the parameters of a given link $i$ as well as joint $i$ and are defined as follows [41].
$a$: length of common normal. If the joint is revolute, then this is the radius about previous $z$.
$d$: offset along previous $z$ to common normal.
$\alpha$: angle about common normal, from old $z$ axis to new $z$ axis.
$\theta$: angle about previous $z$, from old $x$ to new $x$.

In our humanoid robot, all the joints are revolute joints. Hence $\theta_i$ is the only variable for each joint and rest of the parameters are constant for given link lengths and frame assignments. Starting from the base, a coordinate frame is defined at each joint and at the end of each end-effector (hands and left leg) according to DH convention. The complete axes assignment along with numbering is shown in Fig. 2.3.

In Fig. 2.3, all frames are right handed and hence only X and Z axes are shown for the frames in order to have a simpler representation. Y axes can be easily identified by using right hand thumb rule. The world frame is located at the right foot sole and is oriented as shown in Fig. 2.3.

[1] homogeneous transformations are the combined representations of rotations and translation of coordinate frames

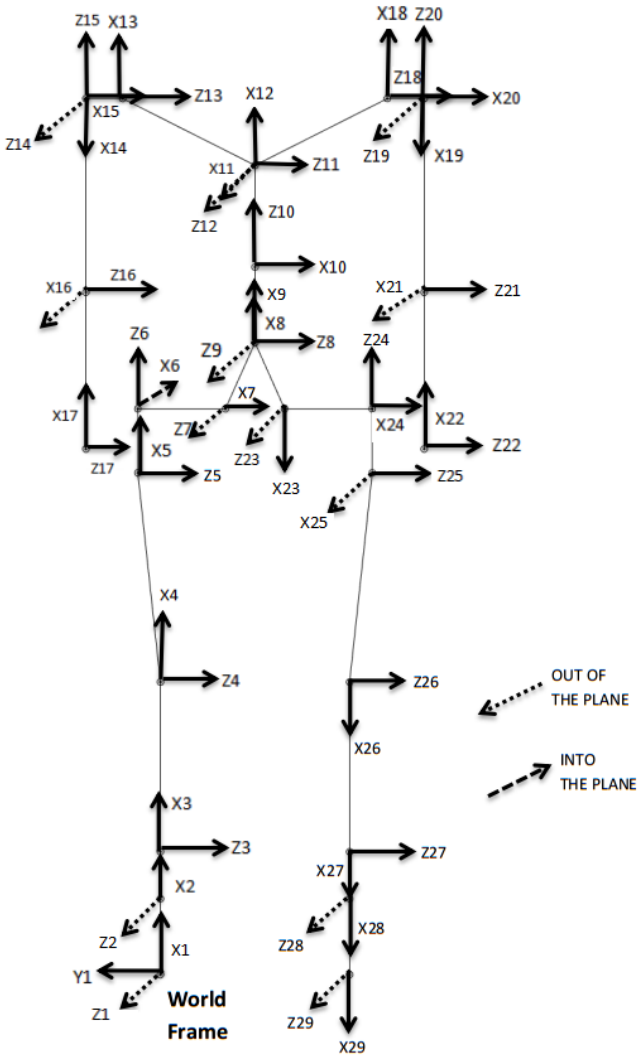



Figure 2.3: Kinematic model of the robot

There are three forward kinematic chains in the defined model - base to right hand, base to left hand and base to left foot sole. In all three chains base to hip is a common chain and in the upper body hip to chest is a common chain. The link and offset lengths are given in Table 2.1. The DH parameters for the common chains are given in the Tables 2.2 and Table 2.3, and for chest to right hand, chest to left hand and hip to left foot sole is given in Table 2.4, Table 2.5 and Table 2.6.

These DH parameters are used for the calculation of Forward Kinematics in the initial stages. Later, Unified Robot Description Format (URDF) was built using the same axes conventions as given in Fig. 2.3 and used for the Forward Kinematic calculations. All these implementations are done using MATLAB in this work.

Table 2.1: Parameter List

| Link Name | Length (in cm) | Offset Name | Length (in cm) |
|---|---|---|---|
| $l_0$ | 58.1 | $d_3$ | 17.3 |
| $l_1$ | 36 | $d_4$ | 14 |
| $l_2$ | 130.15 | $d_5$ | 22.6 |
| $l_3$ | 160.2 | $d_6$ | 10.7 |
| $l_4$ | 49 | $d_8$ | 4.2 |
| $l_5$ | 67 | $d_9$ | 4.6 |
| $l_6$ | 50.7 | $d_{14}$ | 28.2 |
| $l_8$ | 57.8 | $d_{15}$ | 10 |
| $l_9$ | 77.9 | | |
| $l_{11}$ | 51.1 | | |
| $l_{12}$ | 65.3 | | |
| $l_{13}$ | 101.1 | | |
| $l_{15}$ | 38.8 | | |
| $l_{16}$ | 109.5 | | |
| $l_{17}$ | 120 | | |

Table 2.2: Common Kinematic Chain - 1: Base to Hip

| Parameter | $2 \to 1$ | $3 \to 2$ | $4 \to 3$ | $5 \to 4$ | $6 \to 5$ | $7 \to 6$ | $8 \to 7$ |
|---|---|---|---|---|---|---|---|
| $\alpha$ | 0 | $\pi/2$ | 0 | 0 | $-\pi/2$ | $\pi/2$ | $\pi/2$ |
| $a$ | $l_0$ | $l_1$ | $l_2$ | $l_3$ | $d_4$ | $l_5$ | $l_6$ |
| $d$ | 0 | 0 | 0 | $-d_3$ | 0 | $l_4$ | $d_6$ |
| $\theta$ | 0 | $\theta_1$ | $\theta_2 - 0.0623$ | $\theta_3 + 0.0623$ | $\theta_4 - \pi/2$ | $\theta_5 - \pi/2$ | $\theta_6 + \pi/2$ |

## 2.3 Stability and Zero Moment Point

Zero Moment Point (ZMP) [31], [30] is one of the widely used dynamic stability measures which was proposed by Vukobratović and Stepanenko in 1972. In bipeds, a configuration is said to be stable if ZMP lies inside the convex hull of the feet, which defines the support polygon [42].

In double support phase when the robot is stationary, ZMP is equal to the center of mass (CoM) projection in the support polygon. However when the robot is not stationary, ZMP might deviate from the CoM projection even in double support phase. In order to have an accurate ZMP point we need to

Table 2.3: Common Kinematic Chain - 2: Hip to Chest

| Parameter | $9 \to 8$ | $10 \to 9$ | $11 \to 10$ | $12 \to 11$ |
|---|---|---|---|---|
| $\alpha$ | $-\pi/2$ | $-\pi/2$ | $-\pi/2$ | $-\pi/2$ |
| $a$ | 0 | 0 | $-d_9$ | 0 |
| $d$ | $d_5$ | $-d_8$ | $l_8 + l_9$ | 0 |
| $\theta$ | $\theta_7$ | $\theta_8 - \pi/2$ | $\theta_9 - \pi/2$ | $\theta_{10} - \pi/2$ |

Table 2.4: Chest to Right hand

| Parameter | $13 \rightarrow 12$ | $14 \rightarrow 13$ | $15 \rightarrow 14$ | $16 \rightarrow 15$ | $17 \rightarrow 16$ |
|---|---|---|---|---|---|
| $\alpha$ | $\pi/2$ | $\pi/2$ | $-\pi/2$ | $-\pi/2$ | $0$ |
| $a$ | $l_{11}$ | $0$ | $0$ | $d_5$ | $-l_{17}$ |
| $d$ | $0$ | $-(l_{13}+d_{14})$ | $0$ | $-(l_{15}+l_{16})$ | $0$ |
| $\theta$ | $\theta_{11}$ | $\theta_{12}+\pi$ | $\theta_{13}+\pi/2$ | $\theta_{14}-\pi/2$ | $\theta_{15}-\pi/2$ |

Table 2.5: Chest to Left hand

| Parameter | $13 \rightarrow 12$ | $14 \rightarrow 13$ | $15 \rightarrow 14$ | $16 \rightarrow 15$ | $17 \rightarrow 16$ |
|---|---|---|---|---|---|
| $\alpha$ | $\pi/2$ | $\pi/2$ | $-\pi/2$ | $-\pi/2$ | $0$ |
| $a$ | $l_{11}$ | $0$ | $0$ | $d_5$ | $-l_{17}$ |
| $d$ | $0$ | $+(l_{13}+d_{14})$ | $0$ | $-(l_{15}+l_{16})$ | $0$ |
| $\theta$ | $\theta_{11}$ | $\theta_{16}+\pi$ | $\theta_{17}+\pi/2$ | $\theta_{18}-\pi/2$ | $\theta_{19}-\pi/2$ |

include momentum and angular momentum into the calculation. Hence the ZMP should be calculated as [43]:

$$\begin{aligned} p_x &= \frac{Mgx - \dot{L_y}}{Mg + \dot{P_z}} \\ p_y &= \frac{Mgy + \dot{L_x}}{Mg + \dot{P_z}} \end{aligned} \tag{2.2}$$

where $p_x$ and $p_y$ are the ZMP coordinates, $x$, $y$ are the x and y coordinates of center of mass (CoM), M is the total mass of the robot, $g$ is acceleration due to gravity and [$L_x$, $L_y$, $L_z$], [$P_x$, $P_y$, $P_z$] are the angular and linear momentum respectively with respect to base frame.

### 2.3.1 Recursive Newton Euler Algorithm

Recursive Newton-Euler Algorithm (RNEA) [44], an inverse dynamics algorithm, was used for the calculation of momentum and angular momentum required for ZMP calculation. For the implementation of this algorithm, spatial vectors [44] were used to simplify the calculations.

**Spatial Vectors** [45]

Given any point $O$, the velocity of a rigid body can be described by a pair of 3-D vectors, $\underline{\omega}$ and $\underline{v_O}$, which specify the body's angular velocity and linear velocity of the body-fixed point currently at $O$. Note that the $\underline{v_O}$ is not velocity of $O$ itself, but rather the velocity of body-fixed point that currently coincides with $O$.

The velocity of this same rigid body can also be described by a single spatial motion vector, $\mathbf{v} \in M^6$ (motion vector space). Introduce a coordinate frame, $O_{xyz}$ with its origin at $O$. This frame defines

Table 2.6: Hip to Left foot sole

| Parameter | $23 \rightarrow 8$ | $24 \rightarrow 23$ | $25 \rightarrow 24$ | $26 \rightarrow 25$ | $27 \rightarrow 26$ | $28 \rightarrow 27$ | $29 \rightarrow 28$ |
|---|---|---|---|---|---|---|---|
| $\alpha$ | $\pi/2$ | $-\pi/2$ | $-\pi/2$ | $0$ | $0$ | $\pi/2$ | $0$ |
| $a$ | $l_6$ | $l_5$ | $d_4$ | $l_3$ | $l_2$ | $l_1$ | $l_0$ |
| $d$ | $2d_5$ | $-d_6$ | $-l_4$ | $-d_3$ | $0$ | $0$ | $0$ |
| $\theta$ | $\theta_6 + \pi$ | $\theta_{20} + \pi/2$ | $\theta_{21} - \pi/2$ | $\theta_{22} + \pi/2$ | $\theta_{23} - 0.0623$ | $\theta_{24} + 0.0623$ | $\theta_{25}$ |

a Cartesian coordinate system for $\underline{\omega}$ and $\underline{v_O}$, and also Plücker coordinate system [46] for **v**. Now, the coordinate vector representing **v** in $O_{xyz}$ can be written as

$$\mathbf{v_O} = \begin{pmatrix} \omega_x \\ \omega_y \\ \omega_z \\ v_{Ox} \\ v_{Oy} \\ v_{Oz} \end{pmatrix} = \begin{pmatrix} \underline{\omega} \\ \underline{v_O} \end{pmatrix},$$

where $\omega_x, ..., v_{Oz}$ are the Cartesian coordinates of $\underline{\omega}$ and $\underline{v_O}$ in $O_{xyz}$.

The definition of spatial force is very similar. Given any point $O$, any system of forces acting on a single rigid body is equivalent to a single force $\underline{f}$ acting on a line passing through $O$, together with a pure couple, $\underline{n_O}$, which is the moment of the force system about $O$. This same force can also be described by a single spatial force vector, $\mathbf{f} \in F^6$ (force vector space). Introducing the frame $O_{xyz}$, the coordinate vector representing **f** in $O_{xyz}$ can be written as

$$\mathbf{f_O} = \begin{pmatrix} n_{Ox} \\ n_{Oy} \\ n_{Oz} \\ f_x \\ f_y \\ f_z \end{pmatrix} = \begin{pmatrix} \underline{n_O} \\ \underline{f} \end{pmatrix},$$

where $n_{Ox}, ..., f_z$ are the Cartesian coordinates of $\underline{n_O}$ and $\underline{f}$ in $O_{xyz}$.

**Vector Products in Spatial Vectors**

In the context of spatial vectors, two types of vector products are defined. The first takes two motion-vectors and produces a motion-vector result as shown below:

$$\mathbf{m_1} \times \mathbf{m_2} = \begin{pmatrix} \underline{m_1} \\ \underline{m_{1O}} \end{pmatrix} \times \begin{pmatrix} \underline{m_2} \\ \underline{m_{2O}} \end{pmatrix} = \begin{pmatrix} \underline{m_1} \times \underline{m_2} \\ \underline{m_1} \times \underline{m_{2O}} + \underline{m_{1O}} \times \underline{m_2} \end{pmatrix}. \tag{2.3}$$

The second takes a motion vector as left-hand argument and a force vector as right-hand argument, and produces a force-vector result. It is defined by the formula

$$\mathbf{m} \times \mathbf{f} = \begin{pmatrix} \underline{m} \\ \underline{m_O} \end{pmatrix} \times \begin{pmatrix} \underline{f_O} \\ \underline{f} \end{pmatrix} = \begin{pmatrix} \underline{m} \times \underline{f_O} + \underline{m_O} \times \underline{f} \\ \underline{m} \times \underline{f} \end{pmatrix}. \tag{2.4}$$

Given all these definitions, RNEA is shown in Algorithm 3. Here $\mathbf{v}_0$ , $\mathbf{a}_0$ are the spatial velocity and ac-

**Algorithm 3** Recursive Newton Euler Algorithm (RNEA)

$\mathbf{v}_0 = \mathbf{0}$
$\mathbf{a}_0 = -\mathbf{a}_{\mathrm{g}}$
**for** $i$ = 1 to $N_B$ **do**
    $\mathbf{v}_i = \mathbf{v}_{p(i)} + \boldsymbol{\Phi}_i \dot{\boldsymbol{q}}_{\boldsymbol{i}}$
    $\mathbf{a}_i = \mathbf{a}_{p(i)} + \boldsymbol{\Phi}_i \ddot{\boldsymbol{q}}_{\boldsymbol{i}} + \dot{\boldsymbol{\Phi}}_i \dot{\boldsymbol{q}}_{\boldsymbol{i}}$
    $\mathbf{h}_i = \mathbf{I}_i \mathbf{v}_i$
    $\mathbf{f}_i = \mathbf{I}_i \mathbf{a}_i + \mathbf{v}_i \times \mathbf{h}_i - \mathbf{f}_i^e$
**end for**
**for** $i$ = $N_B$ to 1 **do**
    $\tau_i = \boldsymbol{\Phi}_i^{\mathrm{T}} \boldsymbol{f}_{\boldsymbol{i}}$
    **if** $p(i) \neq 0$ **then**
        $\mathbf{f}_{p(i)} = \mathbf{f}_{p(i)} + \mathbf{f}_i$
    **end if**
**end for**

celeration respectively. Spatial acceleration is defined as the derivative of the spatial velocity. $N_B$ is the number of bodies present in the kinematic chain. $\mathbf{v}_i$ is the spatial for $i^{th}$ body while $\mathbf{v}_{p(i)}$ is the velocity of the parent of $i^{th}$ body, $\mathbf{a}_i$ and $\mathbf{a}_{p(i)}$ are defined in a similar manner. $\boldsymbol{\Phi}_i$ are the Plücker coordinates and for a revolute joint $\boldsymbol{\Phi}_i = [0, 0, 1, 0, 0, 0]^{\mathrm{T}}$. Also, $\mathbf{h}_i$, $\mathbf{f}_i$, $\mathbf{I}_i$ and $\tau_i$ represent the spatial momentum, force, inertia and torque of the $i^{th}$ body. Finally $\dot{\boldsymbol{q}}_{\boldsymbol{i}}$, $\ddot{\boldsymbol{q}}_{\boldsymbol{i}}$ represent the joint velocity and acceleration of $i^{th}$ joint while $\mathbf{f}_i^e$ is the external force acting on the $i^{th}$ body.

The spatial momentum $\mathbf{h}_i$ is calculated as shown in the above algorithm for each joint and finally all the joints moments are summed up, after expressing them in the base frame, to get the final spatial momentum $\mathbf{h}$. This gives the linear and angular momentum that are required to calculate ZMP according to Eq. (2.2). During training, these ZMP values are used to check for the stability of the humanoid robot.

*Chapter 3*

# Learning Stable Inverse Kinematics

## 3.1 Introduction

In this chapter we demonstrate how deep RL can be used to learn generalized solutions for inverse kinematics of the given humanoid robot. We propose a DDPG based IK solver which takes into account criteria of stability and self-collision avoidance while generating configurations. We validated the method by applying this framework to learn reachability tasks in the double support phase [29, 30].

The rest of the chapter is organized as follows. Section 3.2 discusses the need for learning based IK solvers in humanoid robots. Section 3.3 explains the proposed methodology to learn the stable IK solver along with self-collision avoidance. It details the state vector, reward function, network architecture and finally describes the environment modelling along with a pseudo code for training. Following that the results of training are shown in Section 3.4 along with numerical simulations. Finally conclusions and future work are discussed in Section 3.5.

## 3.2 General Inverse Kinematics and Stability

Most of the general inverse kinematics solvers work in the velocity domain and solve for inverse kinematics using Jacobian or gradient descent method. Some of the famous solvers include Jacobian based methods like pseudo Jacobian Inverse [15], damped least square method [15] (LevenbergMarquardt algorithm [17]), and Hessian based methods like BFGS [13]. Although constraints like singularity avoidance and joint limits can be included in these methods, stability criteria cannot be included directly in IK solver. Hence the resultant solution of inverse kinematics may not be stable in case of humanoids.

One way to avoid such scenarios is to keep on checking the intermediate configurations and reiterating until a stable configuration is achieved by the IK solver. Generally, this takes very long time to converge and in some cases, the solver might not be able to find a stable configuration at all. Therefore, there is a need for an IK solver which takes stability into account while solving and doesn't require any external checking. This kind of solver can be easily learnt using learning based inverse kinematics [23].

RL requires only the kinematic model of the robot for learning a generalized solver. Using this idea, a generalized IK framework can be defined for complex robots like humanoids where balance and posture plays a great role apart from reaching the goal.

## 3.3 Deep Deterministic Policy Gradient (DDPG) for Inverse Kinematics

DDPG is an actor-critic based policy learning algorithm that has the capability of learning continuous control tasks as discussed in [27]. A detailed description of the algorithm was already discussed in Chapter 1. In this section, the application of DDPG for learning IK solver in humanoid is discussed. We explain how state vector, reward function and environment should be modelled for the training, followed by a pseudo code of the training loop.

### 3.3.1 State vector and network architecture

The chosen state vector consists of joint angles ($\mathbf{q}$), the end effector coordinates and the goal position coordinates. The action vector is a set of joint velocities, $\dot{\mathbf{q}}$. Hence the policy learns a mapping from configuration space to velocity space. The state vector is of 21 dimensions and the action vector is of 13 dimensions. Since we are learning the reachability tasks in double support phase, the joint angles and positions of both the legs are not included in the state vector.

A 2 layered network consisting of fully connected layers with 400, 300 hidden units is used for both actor and critic networks. $cRelu$ [47] is used as activation function and $\tau$ is taken as 0.001. The output layer in critic doesn't have any activation function whereas, the output layer of actor has $Tanh$ activation. Batch normalization is used in the network to avoid over-fitting and handle the scale variance problems.

### 3.3.2 Reward function

The main objective of an IK problem is to provide a set of angles ($\mathbf{q}$) that are needed to reach the given position and orientation. Most of the Jacobian based methods solve for this using gradient descent and the solution is minimized in terms of $\dot{\mathbf{q}}$. Therefore $min(\dot{\mathbf{q}})$ is included as a part of the reward function. In order to ensure that the configurations given out by the solver are within the stability region, a large negative reward is given whenever it goes out of stability bounds. The final reward function is shown in Eq. 3.1.

$$r = \begin{cases} -\alpha dist - \beta\sqrt{\sum_i (\Delta q_i)^2} & if\ stable\ and\ collision free \\ -\kappa & if\ unstable \end{cases} \tag{3.1}$$

where $\alpha, \beta, \kappa$ are the normalization constants, $dist$ is the absolute distance between goal position and the current end effector position and $\Delta q_i$ is the angular difference between the starting configuration and the current configuration of the $i_{th}$ joint. In our case, $\alpha$ is $\frac{1}{70}$, $\beta$ is $\frac{10}{2\pi}$ and $\kappa$ is 20.

### 3.3.3 Environment modelling and Training

**Algorithm 4** Humanoid Environment

**function** $Reset()$
  $config \leftarrow Set\ random\ initial\ configuration$
  $goal \leftarrow Set\ random\ goal\ position$
  $s \leftarrow GetState(config)$
**return** $s$

**function** $GetState(config)$
  $EnfPos \leftarrow ForwKin(config)$
  $state \leftarrow concat(config, EnfPos, goal, done)$
**return** $state$

**function** $Step(action)$
  $action \leftarrow clip(action, ActionBound)$
  $config \leftarrow config + action$
  $ForwKin(config)$ ▷ Updates the kinematic model
  $r, done \leftarrow Reward(config)$
  $s = GetState(config)$
**return** $s, r, done$

**function** $Reward(config)$
  $ZMP \leftarrow CalZMP(config)$
  **if** $ZMP\ in\ support\ polygon$ **and** $collision\ free$ **then**
    $r = -\alpha dist - \beta\sqrt{\sum_i (\Delta q_i)^2}$
  **else**
    $r = -\kappa$
  **end if**
  **if** $goal\ is\ reached$ **then**
    $r = r + \lambda$ ▷ Add large positive reward
    $done = True$
  **else**
    $done = False$
  **end if**
**return** $r, done$

Modelling of the environment is a very crucial part for any RL algorithm. In order to learn a generalized inverse kinematics solution, the entire configuration space needs to be spanned while training.

This is achieved by randomly sampling both start configuration and goal position for every episode. Algorithm 4 shows the environment used for the training.

The actor and critic networks are trained using the given humanoid environment. In DDPG, policy is learnt by the actor network and Q-value function is learnt by the critic network. Target networks are used for both actor and critic and these are updated very slowly using $\tau$ as in Eq. (1.12). We used a Replay buffer of size $5 \times 10^5$. The learning rates for both actor and critic were 0.0001 with a batch size of 64. The discount is taken as 0.99 for training. The entire code was done in Python using Tensorflow code bade and RMSProp optimizer is used for training. The pseudo code for training is given in Algorithm 5. A normally distributed decaying random noise is used for the exploration noise which is observed to provide good results in training. Critic and actor networks are updated as given in Eqs. (1.9) and (1.11) respectively. The training results and their evaluations are shown in the subsequent sections.

**Algorithm 5** IK learning using DDPG

$Randomly\ initialize\ Actor\ and\ Critic\ Networks$
$TargetActorNet \leftarrow ActorNet$
$TargetCriticNet \leftarrow CriticNet$
**for** $i = 1\ to\ MaxEpisodes$ **do**
    $s \leftarrow Reset()$
    **for** $j = 1\ to\ MaxStep$ **do**
        $action \leftarrow Policy(s)$ ▷ Get action using ActorNet
        $action \leftarrow action + N$ ▷ Add Exploration Noise
        $s', r, done \leftarrow Step(action)$
        $ReplayBuffer \leftarrow Store(s, a, s', r)$
        **if** $size(ReplayBuffer) > BSize$ **then**
            $batch \leftarrow RandSample(ReplayBuffer, BSize)$
            $Q \leftarrow Update(CriticNet, batch)$
                ▷ Q-value function update
            $Policy \leftarrow Update(ActorNet, batch, Q)$
                ▷ Policy update
            $Update\ Target\ networks\ using\ \tau$
        **end if**
        **if** $done$ **then**
            $break$
        **end if**
    **end for**
**end for**

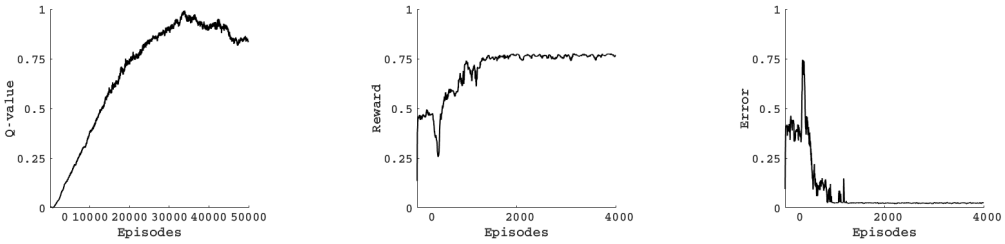
## 3.4 Results and Simulations

The humanoid model was trained taking into account all the criteria explained in the previous sections. Training was run for 50000 episodes with 150 steps in each episode, totalling 7.5 million steps.

Table 3.1: Performance of the IK Solver

| Accuracy | Episodes | | | | | | | | |
|---|---|---|---|---|---|---|---|---|---|
| | 900 | 1200 | 1500 | 1800 | 2100 | 2700 | 3000 | 3900 | 9900 |
| Min | 23% | 61% | 72% | 75% | 79% | 80% | 76% | 88% | 87% |
| Max | 36% | 69% | 78% | 82% | 85% | 84% | 83% | 89% | 93% |
| Mean | 30% | 66.33% | 75% | 78.33% | 82.66% | 82% | 79.66% | 88.66% | 90% |

### 3.4.1 Training results




(a) Normalized Q-value (b) Normalized Reward (c) Error

Figure 3.1: Results of Training on 7.5 million steps

Figs. 3.1a and 3.1b show the normalized $Q$-value and reward of training. In Fig. 3.1a, the plot started to nearly saturate after 30000 episodes showing the attainment of optimal $Q$-value function. Error is defined as the difference between the end-effector and goal position at the end of an episode. The corresponding normalized error plot is shown in Fig. 3.1c. The error goes on decreasing with training and reaches a minimum value soon after 1500 episodes showing that the network has learnt the required policy. The same is reflected in the reward function as shown in Fig. 3.1b.

The trained model is tested for reachability tasks by giving random start configurations and goal positions. The accuracies of the network obtained by using 100 random samples over 3 different seeds at different points of learning are documented in Table. 3.1. It can be observed from the table, that the accuracy goes on increasing with the training and oscillates in a small region after 2000 episodes. The highest mean accuracy obtained is 90% at 9900 episodes.

### 3.4.2 Simulated Experiments

The trained IK solver is tested in the dynamic simulator of MSC Adams environment. The joint trajectories generated by the solver are given as input to the simulator for testing the solution. A set of three experiments which have high probability of losing balance are chosen in order to demonstrate the efficiency of the learnt IK solver and also to explore the advantages of an articulated torso.

In the first task, it had to reach a point in the far right end where it needs to use its spine to bend towards the right, as shown in Fig. 3.2a. In the second task, as shown in Fig. 3.3a, it has to reach a

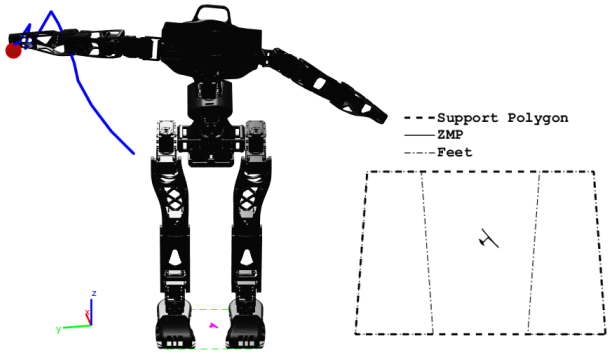



Figure 3.2: Trajectory and ZMP plot for Task 1

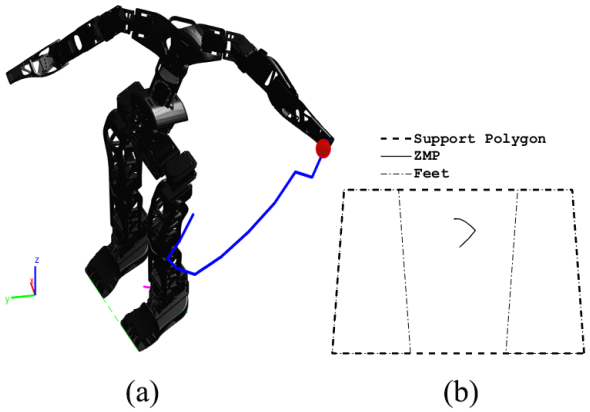



Figure 3.3: Trajectory and ZMP plot for Task 2

point in the left-back side, where the chest motion is tested. In the last task, it had to reach a point below its knee where it tried to explore the limitation of the pelvis and abdomen joints which is shown in Fig. 3.4a.

Figs. 3.2a, 3.3a and 3.4a show the end effector trajectories along with the final posture of the robot. The corresponding ZMP plots for tasks are shown in Figs. 3.2b, 3.3b and 3.4b. It was observed that the ZMP stays within the support polygon while performing each of these tasks.

In all of the three tasks, it was observed that the robot used the other hand to balance itself and stay within the stability region and also avoided self collision. The vertebral column played an important role in making the postures similar to humans, which can be observed from the Figs. 3.2, 3.3 and 3.4.

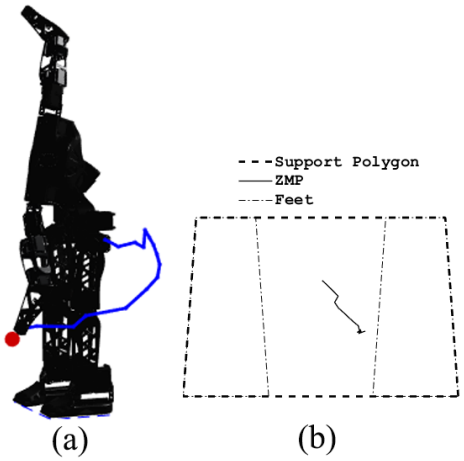



Figure 3.4: Trajectory and ZMP plot for Task 3

## 3.5 Conclusions

This work proposes a methodology for generating dynamically stable inverse kinematic solutions using deep RL. The approach was able to learn a robust IK solver within 2000 episodes. The robustness of the model was tested by giving various complex tasks. It was able to reach most of the points in its configuration space without losing its balance. Also, the solver converges to an inverse kinematics solution in less number of iterations as compared to general inverse kinematic solvers in most of the cases. Although the proposed model has limitations on precision, this model can serve as a good prototype for inverse kinematics solver of highly redundant manipulators.

*Chapter 4*

# Learning Multi-Goal Inverse Kinematics

## 4.1 Introduction

In this chapter, we propose DDPG based Inverse Kinematic (IK) solver for computing IK of both the hands of humanoid simultaneously. The proposed IK solver will also take into account, the criteria of stability and self-collision avoidance while generating joint space configurations. It should be noted that simultaneous dual goal reaching problem is very complex to tackle and IK is not straightforward.

The rest of the chapter is organized as follows: The proposed framework for multi-goal inverse kinematics is explained in section 4.2. Consequently, section 4.3 presents the results and simulations for the proposed framework. Finally conclusions are presented in section 4.4.

## 4.2 Learning Multi-goal Inverse Kinematics for Humanoids

Motion planning for humanoids is a daunting task given that they typically have 25 or more DoF. The problem is further complicated by the fact that humanoids must be controlled carefully in order to maintain dynamic stability. Thus, existing motion planning techniques may not be applicable directly on humanoid robots. Hence, there is need for efficient IK solvers for humanoids that can be easily employed into motion planning algorithms. In this section, we explain the proposed methodology and the underlying modelling to learn inverse kinematics for multi goal reachability tasks in humanoid robots.

In the previous chapter, the application of DDPG for solving Inverse Kinematics for a single manipulator/hand is shown while the other hand and torso of humanoid try to maintain the balance. In this chapter, we make the necessary changes to the previously proposed methodology for adopting it to the multi-goal scenario i.e., two hands trying to reach two different goals simultaneously. The modifications mainly exist in the state vector and reward function formulation. There are significant modifications to the network architecture as well.

### 4.2.1 State vector and network architecture

The chosen state vector consists of all the joint angles ($\mathbf{q}$) of the upper body, end-effectors coordinates and both goal coordinates. It also contains flags related to collision, stability and coordinates of last joint of spine where both arms are connected. The actor network is fed in with state vector and it outputs the action vector that contains angular velocities of all the joints needed to move towards the goals. The critic network takes state-action vector as input and outputs its corresponding action-value. The state and action vector dimensions used in our model are 30 and 13 respectively.

Both actor and critic networks are fully connected two hidden layers each. In both networks, the first hidden layer consists of 700 units and the second hidden layer consists of 400 units with $CReLu$ activation in both the layers. Batch normalization and a drop-out of 20% is present in both hidden layers of actor. $Tanh$ activation is used in the output layer of actor. In critic network, first hidden layer has a drop-out of 30% and batch normalization, whereas the second hidden layer has a drop-out of 20% with no batch normalization. L2 regularization of 0.01 is used in all layers of the critic network. The output layer in critic don't have any activation function.

The network architecture for the actor and critic is given in Fig 4.1. Note that use of $CReLu$ doubles the depth of the activations.

### 4.2.2 Reward function

In deep RL frameworks, reward function is an integral part of the network update and hence the underlying policy that is learnt by the network. For learning multi-goal inverse kinematics, we need to devise a global reward function such that it addresses reachability tasks of both the hands simultaneously. The main objective of an IK problem is to provide a set of angles ($\mathbf{q}$) that are needed to reach the given goal position.

In order to ensure that the configurations given out by the solver are within the stability region, a large negative reward is given whenever it goes out of stability bounds. The final reward function is shown below.

$$r_i = \begin{cases} -\alpha dist_i & if\ stable\ and\ collision free \\ -\kappa & if\ unstable\ or\ collides \end{cases} \tag{4.1}$$

$$if\ hand_i\ reaches\ goal\ then\ r_i = r_i + \lambda \tag{4.2}$$

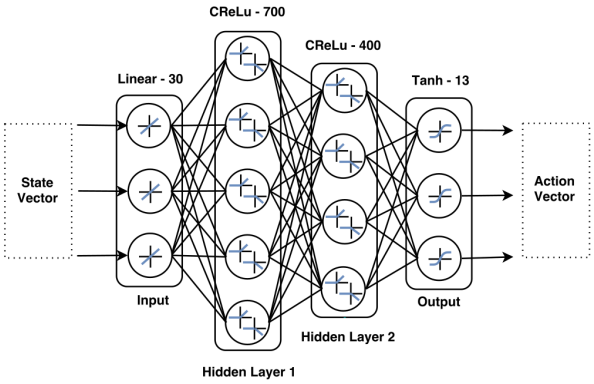
If both the hands reach the goal simultaneously then

$$r_i = \beta \qquad \forall i = 1, 2$$

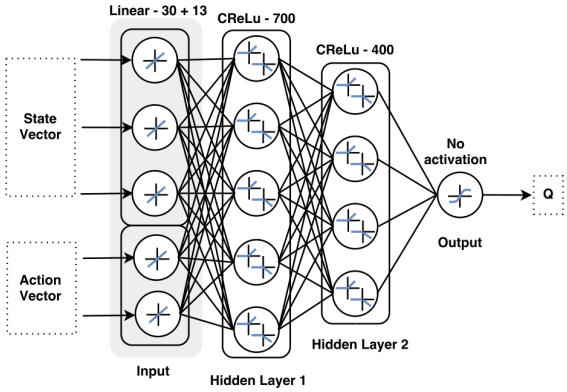
where $r_i$ denotes reward of $i_{th}$ task. The global reward returned by the environment is given by the sum of rewards of both the tasks, that is,

$$r = r_1 + r_2$$

where $\alpha, \beta, \kappa$ are the normalization constants, $dist_i$ is the absolute distance between goal position and end effector position for hand $i$ where $i = 1, 2$.




(a) Actor network : Takes state vector as input and outputs action vector. The activation function and the number of units are shown on the top of each layer.




(b) Critic network: Takes concatenated state-action vector as input and outputs action value function, Q. The activation function and the number of units are shown on the top of each layer.

Figure 4.1: Actor and Critic Network Architectures

### 4.2.3 Environment setting and Training

The environment and training loop are very similar to those shown in Chapter 3 (Algorithm 4 and 5). In order to learn a generalized inverse kinematics, the entire workspace need to be spanned. To ensure that, the start configuration and goal positions are sampled randomly at the start of each episode. This is done in the Reset() step of the environment. In Step($a$), robot environment executes the action $a$ and returns next state $s'$, reward $r$ and terminate flag $tr$. The terminate flag will be one if there is self-collision or the robot loses balance.

The robot is trained in MATLAB using Robotics toolbox and tested in MuJoCo, a dynamic simulation environment. The training setting is modelled in TensorFlow code-base in Python. RMSProp optimizer is used to train both actor and critic networks with learning rate 0.0001 for both. Target networks are used for both critic and actor networks. These target networks are copied into their corresponding actor and critic network after every 1000 and 1100 steps respectively. We used a replay

buffer (R) of size 50000 to store the information of each step. Training was run for 10000 episodes with 150 steps in each episode. A mini-batch of size 64 is sampled randomly from the replay buffer for training the networks. A normally distributed decaying noise function was used for exploration. The episode was terminated in between if the robot went unstable or self-collided. Results and observations of training are presented and discussed in the subsequent section.

## 4.3 Results and Simulations

The humanoid robot shown in Fig. 1.1 was trained for reachability tasks of both the hands, taking into account stability and collision avoidance using the reward setting explained in previous section. For incorporating stability criteria in the reward function, we used Zero Moment Point [43], a dynamic stability check measure as explained in Chapter 2.

### 4.3.1 Training Results

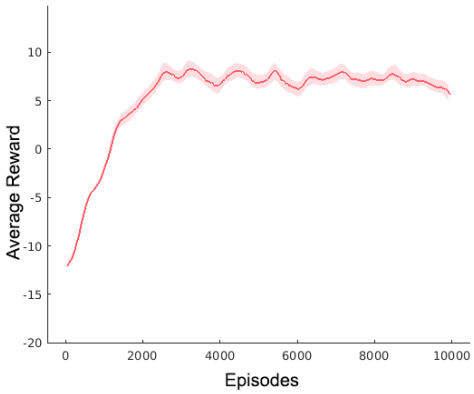



Figure 4.2: Average reward: X-axis shows episodes and Y-axis shows the value of the reward. Coloured region is 95% confidence interval.

In order to evaluate the learning process, average reward per episode and error per episode are plotted, which are shown in Figs. 4.2,4.3. Fig. 4.2 shows the average reward curve with episodes during training. The bold line shows the reward value and coloured region around shows the 95% confidence interval. It can be observed that the average reward increases gradually and saturates showing completion of the training process.

In Fig. 4.3, error curves of both the hands from their respective goals at the end of each episode are plotted. It can observed that these errors decrease gradually and saturates to a value very close to zero. This signifies the completion of the task, as error depicts distance between hands and their respective goals. In Fig. 4.3, each coloured line indicates the error corresponding to one task. As both the tasks were trained simultaneously, we can see that the both the curves of average error have similar profile.

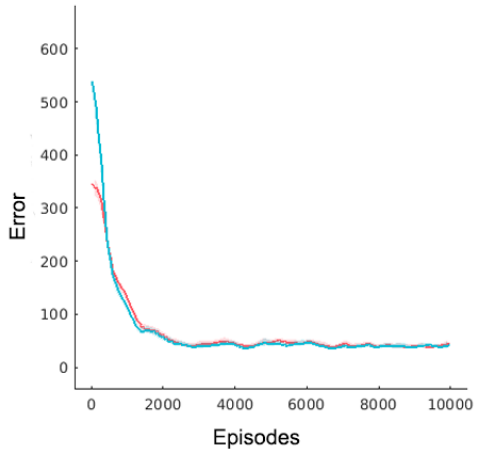



Figure 4.3: Average error graphs of left (red line) and right (blue line) hand. X-axis shows episodes and Y-axis show value of the errors.

It can be observed that both reward and error plots started to saturate after 3000 episodes which shows that the optimal solution has been attained, therefore indicating that the network has learnt the optimal IK solver.

### 4.3.2 Testing in dynamic simulator

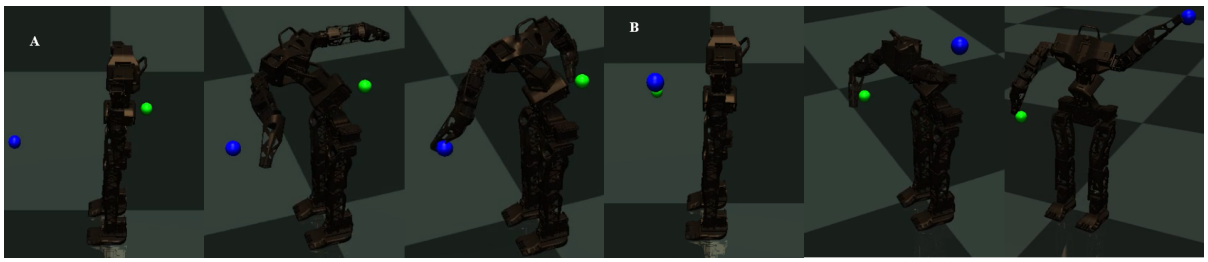



Figure 4.4: Snippets of two cases of humanoid simulation in MuJoCo, where the trained model is tested for multi-goal reachability task. In A, one goal position is the front and another in back. In B, both goal positions are in front but one is upwards whereas the other one is just in front of pelvis.

The learnt IK solver is tested by providing several random goal positions. For testing, we used a dynamic simulator, MuJoCo, instead of MATLAB, to show the transferability of the solver to a more realistic environment. Some snippets of humanoid from this testing are shown in Fig. 4.4. The snippets show two different goal position settings for both the hands, and it can be seen that the solver was able to successfully solve in both settings. We can also observe that the start, intermediate and the final positions are stable configurations. This shows the framework was able to learn the final IK solution as well as the joint trajectories needed to reach given goal positions. Thus, we can conclude that the framework has learnt a stable multi-goal IK solver for 27 DoF humanoid which gives stable configurations at all

intermediate steps.

The above learnt IK solver can be easily transferred to a real robot by using velocity level controllers. This can be done easily by providing the goal positions to the IK solver and then using the solver in the feedback loop of the velocity controllers.

## 4.4 Conclusions

In this chapter, a methodology to learn inverse kinematics of multiple open kinematic chains was proposed. The proposed methodology was based on DDPG and was able to learn IK solver that can simultaneously solve IK for multiple end effectors. A vivid description of the method and the networks used are presented along with reward function modelling.

The proposed framework was applied to learn generalized IK solver for a 27 DoF humanoid with 5 DoF articulated torso for reachability tasks of both hands and results were presented. Results show that the framework is capable of learning inverse kinematics (also joint trajectories), along with maintaining stability and self-collision avoidance. Although the proposed model has limitations like collision avoidance and accuracy, this model can be good prototype for solving inverse kinematics on highly redundant manipulators.

*Chapter 5*

# A Novel RL Framework for Multi-Goal Inverse Kinematics

## 5.1 Introduction

In this chapter, a novel RL framework called DiGrad is proposed which is based on differential policy gradients to learn multi-tasking in complex robotic systems like humanoids where different tasks may share a set of common actions (or kinematic chains) and are not completely separable. One direction for learning multiple tasks in such scenarios is to use the standard DDPG setting, considering the whole of them as a single task, taking a compound action to solve them all at the same time and designing a global reward for globally addressing these tasks which was presented in Chapter 4. However we found DDPG to be somewhat unstable for such multi-task scenarios. DiGrad addresses these problems by using differential policy gradient updates. The proposed framework shows substantial improvement over DDPG and is more robust on all the experiments conducted.

The rest of the chapter is organised as follows. Section 5.2 discusses the related works. Section 5.3 explains the mathematical background behind the proposed framework and provides the detailed algorithm. Finally, Section 5.4 and 5.5 contain the experimental results and conclusion respectively.

## 5.2 Related Works

Most of the multi-task reinforcement learning algorithms rely on transfer learning approaches. A good collection of these methods is shown in [48]. Some of the recent works based on this approach are presented in [49, 50]. Some works [51, 52] explored learning universal abstractions of state-action pairs or feature successors.

Apart from transfer learning, some works like [53, 54] investigated joint training of multiple value functions or policies. In a deep neural network setting, Distral [55] provided a framework for simultaneous training of multiple stochastic policies and a distilled master policy. Unlike our work, Distral uses multiple networks for each policy and one more network for the distilled policy. In our work, we show how we can use a single network to learn multiple deterministic policies simultaneously.

All the above mentioned methods assume multi-agent scenario whereas in this work, we concentrate on learning multiple tasks in a robotic system. Some very recent works in this scenario are [56] and [57]. These works do not talk about the actions which are shared among different tasks, thus limiting their applicability. Unlike these frameworks, we explore the case of multi-task learning in branched manipulator which have shared action-spaces.

Works on coarticulation [58, 59] and Multi-Objective RL (MORL) [60] are of similar flavour as our work but there are some significant differences as explained. In coarticulation, the agent is to devise a global policy by merging the policies associated with the controllers, that simultaneously commits to them according to their degree of significance. The action selection mechanism in the framework takes the ordered intersection of the $\epsilon$ redundant sets computed for every controller in progress. The central difference in DiGrad is that all the tasks are completed simultaneously unlike in a sequential manner. Another major difference is that the individual sub-policies are not required prior to learning the combined policy unlike in coarticulation. Besides, DiGrad is defined in context of deep neural networks and deterministic policies, whereas coarticulation is for stochastic scenarios under linear approximation of action value function. Also action spaces are same for all controllers in coarticulation, which is not the case with DiGrad.

In MORL, the objective is to find the pareto-optimal policy in Multi Objective MDP (MOMDP). Here, the action spaces of all objectives are considered to be same, whereas DiGrad can used for cases where tasks have different action dimensions. MORL based algorithms are proposed for stochastic policies unlike DiGrad, where only deterministic policies are considered. As per the knowledge of authors, deep learning methods for MDPs have not been extended to Multi Objective MDPs, whereas Digrad is proposed in the context of deep RL for MDPs. Also, DiGrad is one such framework in Deep RL, where multi-task/objective is learnt using single actor-critic network.

DiGrad is proposed for learning control tasks in complex robotic systems. A survey of the methods for learning mechanical models of robots is presented in [61]. Some of the related works in the context of robotic control are given in [62, 63]. However, these methods are based on complex optimization which requires modelling of the complete dynamics, whereas such modelling is not required for DiGrad.

## 5.3 DiGrad: Differential Policy Gradients

We propose a framework for simultaneous reinforcement learning of multiple tasks with shared actions between the tasks in continuous domain. The method is based on differential action-value updates in actor-critic based framework using the DPG theorem. The framework learns a compound policy which optimally performs all the shared tasks simultaneously. Fig. 5.1 shows the higher level description of the method. In this section, we describe the mathematical framework behind this work.

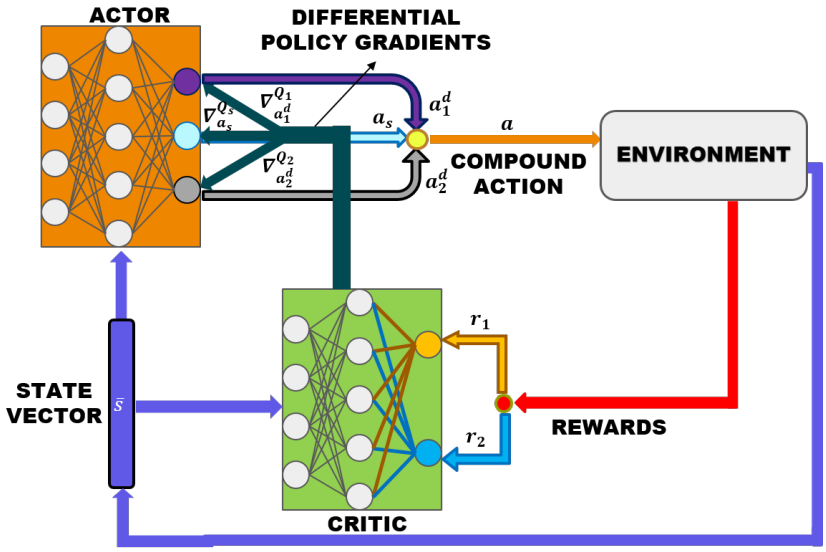



Figure 5.1: Overview of the algorithm showing differential policy gradient update. The disjoint actions $a_i^d$ and shared actions $a_s$ are combined to get a compound action as shown. Note that, the penultimate layer weights for each $Q_i$ are updated based on only their corresponding reward $r_i$. Also, $\nabla Q_s$ is the policy gradient update corresponding to shared action $a_s$ as shown in Eq. (5.9).

### 5.3.1 Environment setting

Consider $n$ tasks in a standard RL setting and their corresponding $n$ action spaces $A_1$, $A_2$, ..., $A_n$. We will assume that the state space $S$ is the same across all the tasks and an infinite horizon with discount factor $\gamma$. Let $A$ be a compound action space which performs the given set of $n$ tasks simultaneously. Let $a \in A$ denote compound actions, $a_i \in A_i$ denote actions and $s \in S$ denote states. Therefore the relation between the compound actions and all the $n$ actions is given by,

$$a = \bigcup_{i=1}^{n} a_i.$$

The actions corresponding to different tasks may or may not be fully separable. Suppose $a_s$ corresponds to the set of actions which is shared among $k$ of the $n$ tasks, then $a_s$ is given by,

$$a_s = \bigcap_{j=1}^{k} a_j \qquad \text{where,} \quad a_j \subset a. \tag{5.1}$$

The reward functions for the $i^{th}$ task depend only on the corresponding actions $a_i$. Therefore, we denote the reward function as $r_i(s, a_i)$ for each task $i$; let $Q_i(s, a_i)$ be the corresponding action value function. Let $\mu$ be the compound deterministic policy and $\mu_i$ be the task-specific deterministic policies; therefore $\mu_i \subset \mu$, where

$$\mu_i(s) = a_i \quad \text{and} \quad \mu(s) = a.$$

### 5.3.2 Proposed framework

An actor-critic based framework is proposed for multi-task learning in the given environment setting. We use a compound policy $\mu(s)$ parametrized by $\theta^{\mu}$, instead of multiple policy networks for each policy $\mu_i$.

$$\mu : S \rightarrow A$$

A simple parametrization for action-value functions would be multi-critic network setting where each $Q_i(s, a_i)$ has a separate network $\theta^{Q_i}$, which outputs an action-value corresponding to the $i^{th}$ task. Another approach for modelling action-value function is single critic setting where a single network parametrized by $\theta^{Q}$ outputs action-values for all the tasks. Here, $Q_i$ is the action value corresponding to the $i^{th}$ task. Works on multi-DDPG [56] show that a single critic network is sufficient in multi-policy learning. In this setting, the penultimate layer weights for each $Q_i$ are updated based on only the reward $r_i(s, a_i)$ obtained by performing the corresponding action $a_i$. The remaining shared network captures the correlation between the different actions $a_i$. Hence this kind of parametrization is more useful in the case of shared actions. Apart from this, the number of parameters are significantly reduced by the use of a single network. In the following subsections, we explain the critic and policy update for both the single critic and multi-critic network settings.

### 5.3.3 Critic update

Consider a single actor-critic based framework where critic $Q(s, a)$ is given by function approximators parametrized by $\theta^{Q}$ and the actor $\mu(s)$ is parametrized by $\theta^{\mu}$. Let the corresponding target networks be parametrized by $\theta^{\mu'}$, $\theta^{Q'}$. Since we have multiple critic outputs $Q_i$, we optimize $\theta^{Q}$ by minimizing the loss as given by [56]:

$$L(\theta^{Q}) = \sum_{i=1}^{n} (Q_i(s_t, a_{it}|\theta^{Q}) - y_{it})^2 \tag{5.2}$$

where the target $y_{it}$ is

$$y_{it} = r_i(s_t, a_{it}) + \gamma Q'(s_{t+1}, {\mu'}_{t+1}(s_{t+1}|\theta^{\mu'})|\theta^{Q'}). \tag{5.3}$$

If there are multiple critic networks, network parameters are optimized by minimizing the corresponding loss:

$$L(\theta^{Q_i}) = (Q_i(s_t, a_{it}|\theta^{Q}) - y_{it})^2. \tag{5.4}$$

In both these settings of critic, there is only a single actor which learns the compound policy $\mu$. The differential policy gradient update on the compound policy is explained in the next subsection.

### 5.3.4 Differential Policy Gradient

Each task has a corresponding reward, $r_i(s, \mu_i(s))$ and hence to learn all the tasks we need to maximize the expected reward for each of the task with respect to the corresponding action, $a_i$. Therefore

the performance objective to be maximized is:

$$J(\mu, \{\mu_i\}_{i=1}^n) = \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^\beta}[r_i(s, \mu_i(s))] \tag{5.5}$$

where $\beta$ is behaviour policy [38] such that $\beta(s) \neq \mu(s)$.

The update on the parametrized compound policy $\mu(s|\theta^\mu)$ is given by applying deterministic policy gradient theorem on Eq. (5.5). The resulting update is:

$$\begin{aligned}\nabla_{\theta^\mu} J &\approx \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^\beta}[\nabla_{\theta^\mu} Q_i(s, a_i|\theta^Q)|_{a_i=\mu_i(s|\theta^\mu)}] \\ &= \sum_{i=1}^{n} \mathbb{E}_{s\sim\rho^\beta}[\nabla_{a_i} Q_i(s, a_i|\theta^Q)|_{a_i=\mu_i(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i(s|\theta^\mu)].\end{aligned} \tag{5.6}$$

#### 5.3.4.1 DiGrad with shared actions

In the above environment setting, let all the $n$ tasks share a common set of actions $a_s$, i.e., $k = n$. Let $a_1$, $a_2$, ..., $a_k$ be the actions corresponding to these $k$-tasks. Therefore from Eq. (5.1),

$$a_s = \bigcap_{j=1}^{k} a_j.$$

Now the compound action $a$ can be written as:

$$\begin{aligned}a &= \{a_1 \cup a_2 ... \cup a_k\} \\ &= \{\{a_1 \setminus a_s\} \cup \{a_2 \setminus a_s\} ... \cup \{a_k \setminus a_s\} \cup a_s\} \\ &= \{a_1^d \cup a_2^d ... \cup a_k^d \cup a_s\}\end{aligned}$$

where, $a_j^d = \{a_j \setminus a_s\}$ and $\setminus$ is the set difference operator.

Here we can see that $a_s$ is the shared action set and $a_1^d, a_2^d, ...a_k^d$ are separable action sets of the $k$ tasks. Therefore we can write $a_i$ = [$a_i^d$,$a_s$]. Similarly, $\mu_i = [\mu_i^d, \mu_s]$. From here onwards, to make it succinct we drop $s \sim \rho^\beta$ from the subscript of $\mathbb{E}_{s\sim\rho^\beta}$ and simply represent it as $\mathbb{E}$.

Substituting these in Eq. (5.6):

$$\nabla_{\theta^\mu} J \approx \sum_{i=1}^{k} \mathbb{E}[\nabla_{[a_i^d, a_s]} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu), a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu}[\mu_i^d(s|\theta^\mu), \mu_s(s|\theta^\mu)]]$$

On expanding with respect to gradient operator we get,

$$= \sum_{i=1}^{k} \mathbb{E} \begin{bmatrix} \nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \\ \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \end{bmatrix}^T \begin{bmatrix} \nabla_{\theta^\mu} \mu_i^d(s|\theta^\mu) \\ \nabla_{\theta^\mu} \mu_s(s|\theta^\mu) \end{bmatrix}$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i^d(s|\theta^\mu) \\
&\qquad + \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_s(s|\theta^\mu)]
\end{aligned}
$$

$$
\begin{aligned}
&= \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i^d(s|\theta^\mu)] \\
&+ \mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_s(s|\theta^\mu)].
\end{aligned} \tag{5.7}
$$

We can see that the second term on R.H.S of Eq.(5.7) will be zero if all the action spaces are disjoint, that is, $a_s = \emptyset$. Hence, this framework can be used even when the actions between different tasks are fully separable. Since the update on the actor are the sum of gradients of different action values, we call this a differential gradient update. It is different from the standard gradient update where an actor is updated based on a single action value [27].

#### 5.3.4.2 Heuristic of direction

From Eq. (5.7), we can see that the policy gradient update for the sub-policy ($\mu_s$) of shared action $a_s$ is taken as sum of the gradients of action values corresponding to the tasks it affects, whereas for policy ($\mu_j^d$) of separable actions $a_j^d$, the gradient update is taken as the gradient of only the corresponding $Q_j$. Thus, this uneven scaling of gradients may lead to delayed convergence and sometimes biasing. In order to equally scale all the gradient updates, we take the average value of the gradients obtained from the different Q-values for the shared action $a_s$. This modification will not affect the direction of gradient, only the magnitude will be scaled. This is further supported by radial algorithm proposed in multi-objective RL [60], where similar ascent direction is used to find a solution belonging to Pareto front. Thus, the differential policy gradient update proposed in DiGrad ascends in the direction of Pareto-optimal policy. By applying this heuristic of direction, the differential gradient update can be written as:

$$
\begin{aligned}
\nabla_{\theta^\mu} J \approx &\sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i^d(s|\theta^\mu)] \\
&+ \frac{1}{k} \mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_s(s|\theta^\mu)].
\end{aligned} \tag{5.8}
$$

#### 5.3.4.3 Generalisation

Suppose there are $k$ tasks which share a set of action $a_s$ as above and $n-k$ tasks which are independent, with corresponding actions $a_{k+1}, ..., a_n$, then Eq. (5.8) can be written as:

$$\begin{aligned}\nabla_{\theta^\mu} J \approx & \sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)|_{a_i^d=\mu_i^d(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i^d(s|\theta^\mu)] \\ & + \frac{1}{k}\mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)|_{a_s=\mu_s(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_s(s|\theta^\mu)] \\ & + \sum_{i=k+1}^{n} \mathbb{E}[\nabla_{a_i} Q_i(s, a_i|\theta^Q)|_{a_i=\mu_i(s|\theta^\mu)} \nabla_{\theta^\mu} \mu_i(s|\theta^\mu)].\end{aligned} \tag{5.9}$$

From Eq. (5.9), we can observe that the policy gradient update for the policy ($\mu_s$) of the shared action set $a_s$ is the average of the gradients of the action-value functions of all the tasks it affects.

This can be easily extended to cases where there are more than one set of shared tasks. Our framework can accommodate heterogeneous dependent action spaces as compared to related multi-task RL algorithms which assume that action spaces are homogeneous or independent or both. This demonstrates the wider applicability of our framework.

### 5.3.5 Algorithm

In this subsection, we describe the algorithm to learn multiple tasks using DiGrad. The flow of the algorithm is very similar to standard DDPG but there are significant differences in terms of the critic and actor updates as shown in the previous subsections. In DiGrad, compound action $a$ is executed in the environment which returns a vector of rewards $\vec{r}_t$ corresponding to each task instead of a single reward. The replay buffer $B$ stores the current state $s_t$, compound action $a_t$, observed state after executing action is $s_{t+1}$ and the vector of rewards is $\vec{r}_t$. The entire flow of the algorithm is shown in Algorithm 6.

## 5.4 Experiments and Results

The proposed framework was tested in different settings in order to analyse the advantages of each setting. We considered four different network settings for DiGrad as follows:
(1) Single critic network with heuristics
(2) Single critic network without heuristics
(3) Multi critic network with heuristics
(4) Multi critic network without heuristics.

We compare all of them with a standard DDPG setting. We use the same set of hyper parameters in all the five settings. The critic network architecture is the same for both single and multiple critic case in all aspects except in the number of outputs. The actor network parameters are also the same for all

**Algorithm 6** Multi-task learning using DiGrad

Randomly initialise actor ($\mu(s|\theta^{\mu})$) and critic network ($Q(s, a|\theta^{Q})$) with weights $\theta^{\mu}$ and $\theta^{Q}$.
Initialize the target network with weights $\theta^{\mu'} \leftarrow \theta^{\mu}$ and $\theta^{Q'} \leftarrow \theta^{Q}$.
**for** $i$ = 1 to $\text{E}_{\text{max}}$
    Initialise random noise $N$ for exploration.
    Reset the environment to get initial state $s_1$.
    **for** $t$ = 1 to $\text{Step}_{\text{max}}$
        Get action $a_t = \mu(s_t|\theta^{\mu}) + N$.
        Execute compound action $a_t$ and get the reward vector $\vec{r}_t$, which contains rewards of all the tasks.
        Get the new state $s_{t+1}$.
        Store transition $(s_t, a_t, \vec{r}_t, s_{t+1})$ in replay buffer $B$.
        Randomly sample a mini-batch $M$ from replay buffer $B$.
        Update critic $\theta^{Q}$ according to Eqs.(5.2), (5.3), (5.4).
        Spilt sampled compound actions $a_t$ into $a_i$, $a_i^d$, $a_s$ as explained in Section 4.
        Calculate differential policy gradients with respect to their corresponding sub-actions $a_i$,$a_i^d$,$a_s$ as given in Eq.(5.9)
        Update actor policy $\theta^{\mu}$ according to the calculated differential policy gradient above.
        Update the target networks $\theta^{\mu'}$ and $\theta^{Q'}$.
    **end for**
**end for**

the cases. We show the comparison of average reward as well the mean scores of each task in all the plots. Note that the average reward curves for DDPG are not shown as the reward function settings for DDPG is different from that of DiGrad.

The proposed multi-task learning framework was tested on the humanoid environment. In this environment training involved learning reachability tasks for all the end effectors simultaneously, i.e., learning a policy on the joint space trajectories to reach any point in their workspace. For all the experiments, we define error and score for a particular task $i$ as,

$$error_i = ||G_i - E_i||, \quad score_i = -log(error_i)$$

where $G_i$ and $E_i$ are the coordinates of goal and end-effector of the $i_{th}$ chain.

In all the experiments, agents were implemented using TensorFlow Code base consisting of 2 fully connected layers. The network architecture for all the settings is explained in Table 1. The output layer has Tanh activation in actor and no activation in critic. Replay buffer size is 50000 and batch size is 64. RMSProp optimizer is used to train actor and critic networks with learning rate 0.0001 for both the networks. We used CReLu activation in all the hidden layers. While training, a normally distributed decaying noise function was used for exploration. By contrast, while testing this noise is omitted. We

| Network Architecture | | | |
|---|---|---|---|
| Settings | Actor | Critic | Dropout |
| Single critic with heuristics | 1 x(700 x 400 x $a$) | 1 x(700 x 400 x $n$) | 20% for all layers |
| Single critic without heuristics | 1 x(700 x 400 x $a$) | 1 x(700 x 400 x $n$) | 20% for all layers |
| Multi critic with heuristics | 1 x(700 x 400 x $a$) | $n$ x(700 x 400 x 1) | 20% for all layers |
| Multi critic without heuristics | 1 x(700 x 400 x $a$) | $n$ x(700 x 400 x 1) | 20% for all layers |
| DDPG | 1 x(700 x 400 x $a$) | 1 x(700 x 400 x 1) | 20% for all layers |

Table 5.1: Network architecture for different settings of DiGrad and DDPG. Note that $n$ denotes the number of tasks and $a$ denotes compound action dimension of all the tasks.

set the discount factor to be 0.999 in all the settings. In all the tasks, we use low-dimensional state description which include joint angles, joint positions and goal positions. The actor output is a set of angular velocities $\dot{q}$. Hence the policy learns a mapping from the configuration space to joint velocity space.

#### 5.4.0.1 Reward function

The reward function for DiGrad settings is modelled keeping in mind the multi-task application. As defined before, $r_i$ is the reward corresponding to the action $a_i$ of the $i_{th}$ task. We give a small positive reward to $r_i$ if task $i$ is finished. Also, if all the end effectors reach their respective goals, a positive reward is given to all the tasks. For all other cases, a negative reward proportional to the $error$ is given. In DDPG setting, there is a single reward unlike DiGrad. A positive reward is given when all the end effectors reach their goals simultaneously. Else, a negative reward is given proportional to the sum of $error$ of all the tasks, that is, sum of distances between the respective goal and its corresponding end effector.

#### 5.4.0.2 Training Results

We test our framework on the given 27 DoF humanoid robot. This experiment involved reachability tasks of both hands of the humanoid robot using the upper body (13 DoF) consisting of an articulated torso. The articulated torso is the shared chain which is affecting both the tasks. It is noteworthy that the articulated torso has five DoF whereas, the arms have four DoF each. Thus, the contribution of shared action (articulated torso) to the task is more than the non shared actions (arms). The environment for training is developed in MATLAB and the trained models were tested in a dynamic simulation environment MuJoCo.

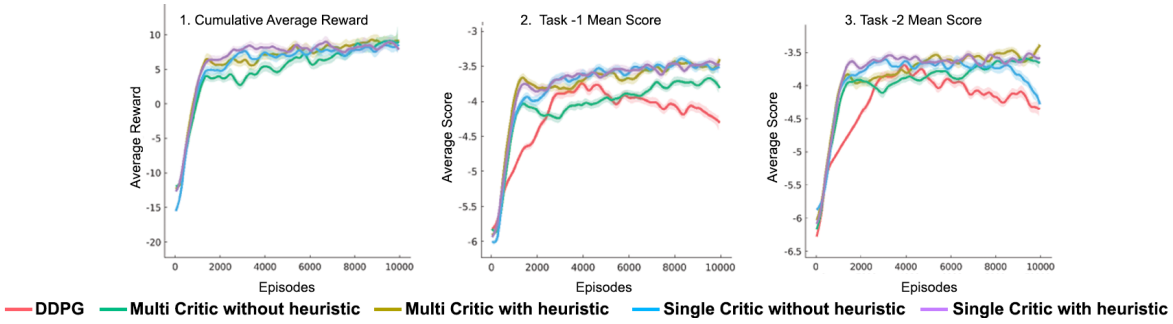



Figure 5.2: Performance curves of reachability task experiments on 8 link manipulator and humanoid. The bold line shows the average over 3 runs and the coloured areas resemble 95% confidence interval. Note that, average reward curve is not plotted for DDPG as the reward function for it is different from DiGrad frameworks.

Fig. 5.2 summarizes the results for this case. We found that DPPG is generally unstable for solving multi-tasks problems. In some runs, it may learn initially but suffers degradation later. We observe that the DiGrad algorithm yields better final results while having greater stability. From the mean scores of the tasks, we can see that the single critic frameworks converge faster and are stable throughout the experiment as compared to the multi-critic frameworks. The best setting is the single critic with heuristic, outperforming all the others in maximum cases.

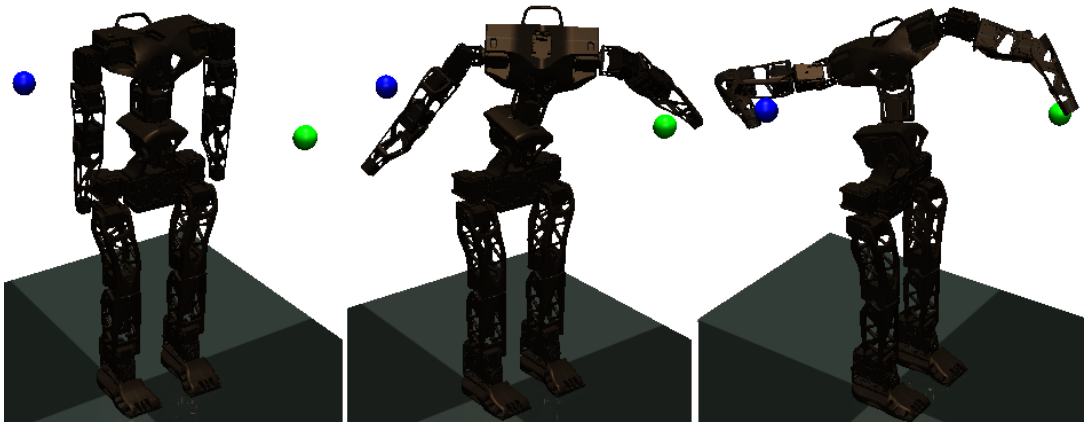


Figure 5.3: Humanoid robot multi-tasking to reach two goals simultaneously. The two goals are shown using blue and green coloured balls.

The simulations of the obtained results are shown in Fig. 5.3. In this simulation the robot starts from a stable initial position and reaches the two goals (green and blue) simultaneously. Further simulations of various control tasks learnt by DiGrad are shown here.[1].

Note that, the reward function for DDPG is kept different from the DiGrad framework. We also tried a different reward setting taking the sum of individual rewards $r_i$ as a reward signal to DDPG framework, where $r_i$ is same as defined in the DiGrad reward setting. We observed that this reward

[1]https://sites.google.com/view/digrad/video

setting led to biasing, where one of the tasks dominated others. This behaviour could have been due to the negative interference across tasks, which didn't happen in DiGrad.

## 5.5 Conclusion

In this work, we propose a deep reinforcement learning algorithm called DiGrad for multi-task learning in a single agent. This framework is based on DDPG algorithm and is derived from DPG theorem. In this framework, we have a dedicated action value for each task, whose update depends only on the action and reward corresponding to the task. We introduced a differential policy gradient update for the compound policy.

We tested the proposed framework for learning reachability tasks of humanoid. These experiments show that our framework gave better performance than DDPG, in terms of stability and robustness. These performances are achieved keeping the number of parameters comparable to DDPG framework. Also, the algorithm learns multiple tasks without any decrease in the performance of each task. Our work focuses on learning coordinated multi-actions in the field of robotics, where a single agent performs multiple tasks simultaneously. The framework was able to capture the relation among tasks where some tasks had overlapped action space.

*Chapter 6*

# Conclusions

Inverse kinematics of redundant manipulators had been a topic of research for decades, however the application of proposed methods was restricted to not more than 7 DoF manipulators. As the number of DoF increase, these methods become computationally expensive. Especially in case of humanoid robots, there is very limited applicability of these methods and for a real time control of these robots, there is need to investigate new methods that significantly brings down the complexity and time. In this thesis, we have explored Deep Reinforcement Learning based inverse kinematic solvers for learning real time IK of humanoids, besides ensuring the stability and self-collision avoidance.

In order to learn a inverse kinematic solver, a continuous policy learning algorithm, Deep Deterministic Policy Gradient was explored. In the first part of the thesis, it was shown how this framework can be adopted in case of humanoids for solving IK or the joint-space trajectories required to reach the given goal. An elaborative description of the environment modelling and reward function is provided, along with the network architectures used for the training. In the proposed methodology, current configuration of the robot and the required goal were taken as inputs and the learnt IK solver provided the solution (joint-trajectories) in an iterative manner. The obtained joint trajectories ensure the stability of the humanoid in double support phase as well as avoid self-collision. This methodology showed an average success rate of 90% and it converged within 2000 episodes in all the settings. Simulations for three different settings were shown in the results, that had high susceptibility of loosing balance, to show the effectiveness of the framework.

In the second part of thesis, a more complicated problem of solving IK for multiple kinematic chains (with shared sub chains) is addressed. The methodology proposed in the first part was extended to dual-arm reachability tasks of a humanoid robot. It was shown how the framework can be adopted to a different and highly complex scenario, without making significant modifications to the framework. A detailed explanation on the required changes in network architecture, state vector and reward function was also provided. Finally, MuJoCo simulations of the robot were shown, that successfully accomplish the task of reaching two goals simultaneously with both the hands in the humanoid robot. Even though the proposed methodology was successful in learning multi-goal IK, it was observed that sometimes DDPG was unstable and highly sensitive to the reward function. Hence in the next part of the thesis, a

novel Reinforcement Learning framework for learning such multi-goal problems with shared actions is proposed.

The proposed multi-task RL framework, DiGrad, based on the differential policy gradients was explained in the final part of the thesis. The entire mathematical formulation was shown along with the supported literature. It was shown that policy gradients can be split and a single policy network can be used to learn multiple tasks using those gradients. In case of shared actions, a heuristic inspired by multi-objective RL was proposed and it was observed that this heuristic, in-fact, had improved the results significantly. DiGrad was tested for learning dual-arm reachability tasks in the humanoid robot and the results were compared with that of DDPG. The results showed better performance than DDPG, in terms of stability and robustness.

Future work involves making the framework more versatile such that it could be extended to various other agents as well. Also, the framework can be extended to learn collision avoidance and motion planning in cluttered environments.

# Related Publications


1. S. Phaniteja, Parijat Dewangan, Pooja Guhan, Abhishek Sarkar, and K. Madhava Krishna, "**A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots**," in *2017 IEEE International Conference on Robotics and Biomimetics (ROBIO)*.

2. S. Phaniteja, Parijat Dewangan, Abhishek Sarkar, and K. Madhava Krishna, "**Learning Multi-Goal Inverse Kinematics in Humanoid Robot**," in *2018 International Symposium on Robotics (ISR)*.

3. Parijat Dewangan, S Phaniteja, K Madhava Krishna, Abhishek Sarkar, Balaraman Ravindran, "**DiGrad: Multi-Task Reinforcement Learning with Shared Actions**," in 2018 arXiv preprint arXiv:1802.10463.

# Bibliography


[1] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press Cambridge, 1998, vol. 1, no. 1.

[2] A. Colomé, "Smooth inverse kinematics algorithms for serial redundant robots," Ph.D. dissertation, Master Thesis, Institut de Robotica i Informatica Industrial (IRI), Universitat Politecnica de Catalunya (UPC), Barcelona, Spain, 2011.

[3] Y. Chua, K. P. Tee, and R. Yan, "Robust optimal inverse kinematics with self-collision avoidance for a humanoid robot," in *RO-MAN, 2013 IEEE*. IEEE, 2013, pp. 496–502.

[4] J.-P. Saut, M. Gharbi, J. Cortés, D. Sidobre, and T. Siméon, "Planning pick-and-place tasks with two-hand regrasping," in *Intelligent Robots and Systems (IROS), 2010 IEEE/RSJ International Conference on*. IEEE, 2010, pp. 4528–4533.

[5] E. Todorov, T. Erez, and Y. Tassa, "Mujoco: A physics engine for model-based control," in *Intelligent Robots and Systems (IROS), 2012 IEEE/RSJ International Conference on*. IEEE, 2012, pp. 5026–5033.

[6] E. Rohmer, S. P. Singh, and M. Freese, "V-rep: A versatile and scalable robot simulation framework," in *Intelligent Robots and Systems (IROS), 2013 IEEE/RSJ International Conference on*. IEEE, 2013, pp. 1321–1326.

[7] N. Koenig and A. Howard, "Design and use paradigms for gazebo, an open-source multi-robot simulator," in *Intelligent Robots and Systems, 2004.(IROS 2004). Proceedings. 2004 IEEE/RSJ International Conference on*, vol. 3. IEEE, 2004, pp. 2149–2154.

[8] S. Carpin, M. Lewis, J. Wang, S. Balakirsky, and C. Scrapper, "Usarsim: a robot simulator for research and education," in *Robotics and Automation, 2007 IEEE International Conference on*. IEEE, 2007, pp. 1400–1405.

[9] E. Coumans, "Bullet physics engine," *Open Source Software: http://bulletphysics. org*, vol. 1, 2010.

[10] N. Chen, C.-M. Chew, K. P. Tee, and B. S. Han, "Human-aided robotic grasping," in *RO-MAN, 2012 IEEE*. IEEE, 2012, pp. 75–80.

[11] M. Do, P. Azad, T. Asfour, and R. Dillmann, "Imitation of human motion on a humanoid robot using non-linear optimization," in *Humanoid Robots, 2008. Humanoids 2008. 8th IEEE-RAS International Conference on*. IEEE, 2008, pp. 545–552.

[12] K. P. Tee, R. Yan, Y. Chua, and Z. Huang, "Singularity-robust modular inverse kinematics for robotic gesture imitation," in *Robotics and Biomimetics (ROBIO), 2010 IEEE International Conference on*. IEEE, 2010, pp. 920–925.

[13] M. F. Møller, "A scaled conjugate gradient algorithm for fast supervised learning," *Neural networks*, vol. 6, no. 4, pp. 525–533, 1993.

[14] D. E. Whitney, "Resolved motion rate control of manipulators and human prostheses," *IEEE Transactions on man-machine systems*, vol. 10, no. 2, pp. 47–53, 1969.

[15] S. R. Buss, "Introduction to inverse kinematics with jacobian transpose, pseudoinverse and damped least squares methods," *IEEE Journal of Robotics and Automation*, vol. 17, no. 1-19, p. 16, 2004.

[16] W. A. Wolovich and H. Elliott, "A computational technique for inverse kinematics," in *Decision and Control, 1984. The 23rd IEEE Conference on*, vol. 23. IEEE, 1984, pp. 1359–1363.

[17] J. J. Moré, "The levenberg-marquardt algorithm: implementation and theory," in *Numerical analysis*. Springer, 1978, pp. 105–116.

[18] S. Chiaverini, "Singularity-robust task-priority redundancy resolution for real-time kinematic control of robot manipulators," *IEEE Transactions on Robotics and Automation*, vol. 13, no. 3, pp. 398–410, 1997.

[19] J. Hollerbach and K. Suh, "Redundancy resolution of manipulators through torque optimization," *IEEE Journal on Robotics and Automation*, vol. 3, no. 4, pp. 308–316, 1987.

[20] B. Siciliano, "Kinematic control of redundant robot manipulators: A tutorial," *Journal of intelligent and robotic systems*, vol. 3, no. 3, pp. 201–212, 1990.

[21] J. Baillieul, "Kinematic programming alternatives for redundant manipulators," in *Robotics and Automation. Proceedings. 1985 IEEE International Conference on*, vol. 2. IEEE, 1985, pp. 722–728.

[22] J.Baillieul, "A constraint oriented approach to inverse problems for kinematically redundant manipulators," in *Robotics and Automation. Proceedings. 1987 IEEE International Conference on*, vol. 4. IEEE, 1987, pp. 1827–1833.

[23] A. D'Souza, S. Vijayakumar, and S. Schaal, "Learning inverse kinematics," in *Intelligent Robots and Systems, 2001. Proceedings. 2001 IEEE/RSJ International Conference on*, vol. 1. IEEE, 2001, pp. 298–303.

[24] V. Mnih, K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski, *et al.*, "Human-level control through deep reinforcement learning," *Nature*, vol. 518, no. 7540, p. 529, 2015.

[25] S. Levine and V. Koltun, "Guided policy search," in *Proceedings of the 30th International Conference on Machine Learning (ICML-13)*, 2013, pp. 1–9.

[26] J. Schulman, S. Levine, P. Abbeel, M. Jordan, and P. Moritz, "Trust region policy optimization," in *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, 2015, pp. 1889–1897.

[27] T. P. Lillicrap, J. J. Hunt, A. Pritzel, N. Heess, T. Erez, Y. Tassa, D. Silver, and D. Wierstra, "Continuous control with deep reinforcement learning," *arXiv preprint arXiv:1509.02971*, 2015.

[28] S. Gu, E. Holly, T. Lillicrap, and S. Levine, "Deep reinforcement learning for robotic manipulation with asynchronous off-policy updates," in *Robotics and Automation (ICRA), 2017 IEEE International Conference on*. IEEE, 2017, pp. 3389–3396.

[29] C.-L. Shih and W. A. Gruver, "Control of a biped robot in the double-support phase," *IEEE transactions on systems, man, and cybernetics*, vol. 22, no. 4, pp. 729–735, 1992.

[30] M. Dekker, "Zero-moment point method for stable biped walking," *Eindhoven University of Technology*, 2009.

[31] M. Vukobratović and J. Stepanenko, "On the stability of anthropomorphic systems," *Mathematical biosciences*, vol. 15, no. 1-2, pp. 1–37, 1972.

[32] C. Dube, M. Tsoeu, and J. Tapson, "A model of the humanoid body for self collision detection based on elliptical capsules," in *Robotics and Biomimetics (ROBIO), 2011 IEEE International Conference on*. IEEE, 2011, pp. 2397–2402.

[33] https://www.cs.cmu.edu/~katef/DeepRLControlCourse/lectures/lecture2_mdps.pdf.

[34] https://en.wikipedia.org/wiki/Reinforcement_learning.

[35] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. Riedmiller, "Playing atari with deep reinforcement learning," *arXiv preprint arXiv:1312.5602*, 2013.

[36] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *Proceedings of the 31st International Conference on Machine Learning (ICML-14)*, 2014, pp. 387–395.

[37] M. Barto, "J. 4 supervised actor-critic reinforcement learning," *Handbook of learning and approximate dynamic programming*, vol. 2, p. 359, 2004.

[38] C. J. Watkins and P. Dayan, "Q-learning," *Machine learning*, vol. 8, no. 3-4, pp. 279–292, 1992.

[39] M. VACEK, J. ŽILKOVÁ, and M. PÁSTOR, "Regulation of dynamixel actuators in robot manipulator movement," *Acta Electrotechnica et Informatica*, vol. 14, no. 3, pp. 32–35, 2014.

[40] M. Lapeyre, P. Rouanet, J. Grizou, S. Nguyen, F. Depraetre, A. Le Falher, and P.-Y. Oudeyer, "Poppy project: Open-source fabrication of 3d printed humanoid robot for science, education and art," in *Digital Intelligence 2014*, 2014, p. 6.

[41] https://en.wikipedia.org/wiki/Denavit%E2%80%93Hartenberg_parameters.

[42] W. Vaughan Jr and R. Herrnstein, "Stability," *Melioration, and Natural Selection*, vol. 1, pp. 185–215, 1987.

[43] S. Kajita, H. Hirukawa, K. Harada, and K. Yokoi, *Introduction to humanoid robotics*. Springer, 2014, vol. 101.

[44] R. Featherstone and D. Orin, "Robot dynamics: equations and algorithms," in *Robotics and Automation, 2000. Proceedings. ICRA'00. IEEE International Conference on*, vol. 1. IEEE, 2000, pp. 826–834.

[45] B. Siciliano and O. Khatib, *Springer handbook of robotics*. Springer Science & Business Media, 2008.

[46] R. Featherstone, "Plucker basis vectors," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1892–1897.

[47] W. Shang, K. Sohn, D. Almeida, and H. Lee, "Understanding and improving convolutional neural networks via concatenated rectified linear units," in *International Conference on Machine Learning*, 2016, pp. 2217–2225.

[48] A. Lazaric, "Transfer in reinforcement learning: a framework and a survey," in *Reinforcement Learning*. Springer, 2012, pp. 143–173.

[49] A. A. Rusu, S. G. Colmenarejo, C. Gulcehre, G. Desjardins, J. Kirkpatrick, R. Pascanu, V. Mnih, K. Kavukcuoglu, and R. Hadsell, "Policy distillation," *arXiv preprint arXiv:1511.06295*, 2015.

[50] H. Yin and S. J. Pan, "Knowledge transfer for deep reinforcement learning with hierarchical experience replay." in *AAAI*, 2017, pp. 1640–1646.

[51] D. Borsa, T. Graepel, and J. Shawe-Taylor, "Learning shared representations in multi-task reinforcement learning," *arXiv preprint arXiv:1603.02041*, 2016.

[52] J. Zhang, J. T. Springenberg, J. Boedecker, and W. Burgard, "Deep reinforcement learning with successor features for navigation across similar environments," *arXiv preprint arXiv:1612.05533*, 2016.

[53] A. Lazaric and M. Ghavamzadeh, "Bayesian multi-task reinforcement learning," in *ICML-27th International Conference on Machine Learning*. Omnipress, 2010, pp. 599–606.

[54] C. Dimitrakakis and C. A. Rothkopf, "Bayesian multitask inverse reinforcement learning," in *European Workshop on Reinforcement Learning*. Springer, 2011, pp. 273–284.

[55] Y. Teh, V. Bapst, W. M. Czarnecki, J. Quan, J. Kirkpatrick, R. Hadsell, N. Heess, and R. Pascanu, "Distral: Robust multitask reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 4499–4509.

[56] Z. Yang, K. Merrick, H. Abbass, and L. Jin, "Multi-task deep reinforcement learning for continuous action control," in *Proceedings of the Twenty-Sixth International Joint Conference on Artificial Intelligence, IJCAI-17*, 2017, pp. 3301–3307.

[57] T. D. Kulkarni, K. Narasimhan, A. Saeedi, and J. Tenenbaum, "Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation," in *Advances in neural information processing systems*, 2016, pp. 3675–3683.

[58] K. Rohanimanesh, R. Platt, S. Mahadevan, and R. Grupen, "Coarticulation in markov decision processes," in *Advances in Neural Information Processing Systems*, 2005, pp. 1137–1144.

[59] K. Rohanimanesh and S. Mahadevan, "Coarticulation: An approach for generating concurrent plans in markov decision processes," in *Proceedings of the 22nd international conference on Machine learning*. ACM, 2005, pp. 720–727.

[60] S. Parisi, M. Pirotta, N. Smacchia, L. Bascetta, and M. Restelli, "Policy gradient approaches for multi-objective sequential decision making," in *Neural networks (ijcnn), 2014 international joint conference on*. IEEE, 2014, pp. 2323–2330.

[61] O. Sigaud, C. Salaün, and V. Padois, "On-line regression algorithms for learning mechanical models of robots: a survey," *Robotics and Autonomous Systems*, vol. 59, no. 12, pp. 1115–1129, 2011.

[62] J. Salini, V. Padois, and P. Bidaud, "Synthesis of complex humanoid whole-body behavior: a focus on sequencing and tasks transitions," in *Robotics and Automation (ICRA), 2011 IEEE International Conference on*. IEEE, 2011, pp. 1283–1290.

[63] R. Lober, V. Padois, and O. Sigaud, "Multiple task optimization using dynamical movement primitives for whole-body reactive control," in *Humanoid Robots (Humanoids), 2014 14th IEEE-RAS International Conference on*. IEEE, 2014, pp. 193–198.