Learning Dual Arm Coordinated Reachability Tasks in a Humanoid Robot with Articulated Torso

Phaniteja S[1*], Parijat Dewangan[1*], Pooja Guhan[1], Madhava Krishna K[1], Abhishek Sarkar[1]

***Abstract*— Performing dual arm coordinated (reachability) tasks in humanoid robots require complex planning strategies and this complexity increases further, in case of humanoids with articulated torso. These complex strategies may not be suitable for online motion planning. This paper proposes a faster way to accomplish dual arm coordinated tasks using methodology based on Reinforcement Learning. The contribution of this paper is twofold. Firstly, we propose DiGrad (Differential Gradients), a new RL framework for multi-task learning in manipulators. Secondly, we show how this framework can be adopted to learn dual arm coordination in a 27 degrees of freedom (DOF) humanoid robot with articulated spine. The proposed framework and methodology are evaluated in various environments and simulation results are presented. A comparative study of DiGrad with its parent algorithm in different settings is also presented.**

I. Introduction

Humans use both arms in a coordinated manner to accomplish many day-to-day tasks. In a human-like robot scenario, using both arms of a robot increases load capacity without any changes in the system, making it an interesting and ongoing research topic. Apart from this, dual-arm manipulation is useful in many industrial scenarios [1], [2] where coordinated tasks need to be performed for instance, lifting heavy boxes, tightening a nut and bolt, cutting etc. In particular, dual-arm coordinated tasks in case of humanoid robots provides uniform distribution of load (makes balancing easy), besides, increasing load handling capacity. However, before the dual-arm manipulation problem is addressed, dual-arm reachability and grasping problem has to be solved.

The reachability and grasping problem usually consists of finding the inverse kinematics solution that can give a valid grasp. Some works [3], [4] store the entire reachability space and search it for the valid grasps. The main drawback of this method is that a large memory is needed to store the reachability space and this increases drastically with the increase in degrees of freedom. Another way is to find a configuration by calculating inverse kinematics on the go and then validate the grasp solutions. Hence this method involves searching [5]–[9] for a feasible inverse kinematic solution that can provide valid grasp solutions. However, when criteria like stability is also included (bipedal robots), the main problem lies in finding a valid and stable inverse kinematics solution which usually takes a long time and hence is inappropriate for online motion planning. The extension of these methodologies to dual arm scenarios are provided in [1], [10]–[12]. One can see that the humanoids [10], [13] used in these works have torso shaped as box with limited DOF and two arms having large DOF. The main advantage of this kind of structure is that there is limited dependency on the torso while performing dual arm coordinated tasks.

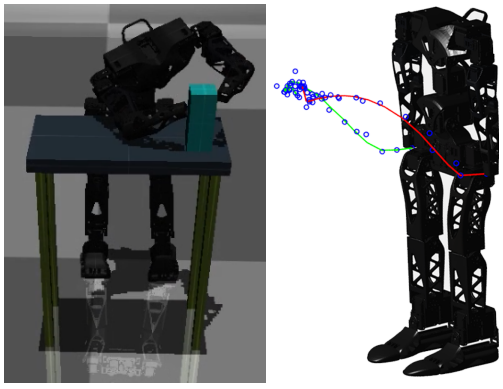


Fig. 1: Left: Humanoid with articulated torso grabbing an object. Right: Trajectory followed by humanoid to grab the object.

Humanoids with articulated torso like Acroban [14], Poppy [15] etc. can perform more complex manipulation tasks compared to the robots without articulated spine. This comes at the expense of more costly computations and complicated mathematical formulations. Dual arm coordination in such robots is highly dependent on spine and this makes it difficult to find a valid IK solution. Especially, calculating valid IK configurations in such structures is not a well solved problem in a classical methodology. Learning based approaches [16] provide a very good alternative for performing complex manipulation tasks in the above mentioned robots. Specifically, Reinforcement Learning (RL) [17] provides a very good framework and some intitial works used it for learning motor control [18] and to teach a biped how to walk [19].

Among the recent advancements in the field of Deep Reinforcement Learning (Deep RL), Deep Deterministic Policy Gradients (DDPG) [20] provided an efficient framework for learning tasks in continuous domain. In [21], the authors presented a modified version of DDPG for teaching a 7 DOF manipulator to open a door. In our recent work [22], we show how DDPG can be used to learn stable Inverse Kinematic solutions of a humanoid robot. In the case of humanoid robots with articulated torso, finding valid IK configurations

*These authors have contributed equally.

[1]All the authors are with Robotics Research Center, International Institute of Information Technology, Hyderabad, India {phaniteja.sp, parijat10, poojaguhan97}@gmail.com, {mkrishna, abhishek.sarkar}@iiit.ac.in

for dual arm coordinated tasks is not a well solved problem. This paper proposes a methodology based on Deep RL to address this problem.

The contribution of this paper is twofold. Firstly we propose a model-free deep RL based framework, DiGrad, for learning continuous control tasks of multiple kinematic chains having a shared kinematic chain between them. This new framework, based on DDPG, is proposed in the context of robotics and comparative results are presented between DiGrad and DDPG. Results show that DiGrad is stable and performs better compared to DDPG in the same scenarios. Secondly, we apply the proposed framework to learn the dual arm coordinated tasks in a 27 DoF humanoid robot with articulated torso [23] (Fig. 1) in cluttered environments. We show that DiGrad successfully learns complex 3D coordinated tasks in three different settings with varying levels of difficulty. In all these settings, along with finding the final IK configuration, we show that the framework is able to learn the joint trajectories that are needed to reach the final configuration. All these results are shown in MuJoCo [24] simulation environment. Also, once learnt, the proposed methodology involves only the forward pass in neural networks to safely reach the final configuration and hence, it is appropriate for online motion planning.

II. DiGrad

In any standard RL environment [17], there is an agent and an environment. The agent in state s performs an action a on the environment and the environment gives out the updated state s' and the reward r back to the agent. The main goal of the RL is to maximize the cumulative reward in an episode. An episode is described as a sequence of states, actions and rewards, which ends with terminal state. We suggest readers to go through [17] for a detailed explanation of RL and actor-critic algorithms. Development of elegant Deep RL algorithms [20], [25] in recent times provided a framework for learning continuous control of manipulators [21]. However, DDPG performs poorly when complex tasks using multiple kinematic chains are involved. Therefore, in this section we present an algorithm, DiGrad, based on Differential Policy Gradients to learn such complex tasks.

A. Differential Policy Gradient Theorem

Consider k tasks in a standard reinforcement learning environment that are needed to be learnt together and have the same state space. Let r_i be reward corresponding to i^{th} task, $i = 1, 2, .., k$. Let the critic network Q be parametrized by θ^Q and actor network μ by θ^μ. The critic network Q outputs multiple action-values each corresponding to a task. DiGrad aims to learn the compound policy μ by splitting it into multiple sub-policies μ_i and taking a combined update using these sub-policies. Before explaining the differential policy gradient theorem for compound policy, we define the differential policy gradients.

Definition 1. *Suppose Q_i is the action-value function corresponding to i^{th} task obtained by performing action a_i in state s. Let μ_i be the sub policy corresponding to the action a_i and μ be the compound policy. Then, $\nabla_{a_i} Q_i(s, a_i|\theta^Q)\nabla_{\theta^\mu}\mu_i(s|\theta^\mu)$ is called the Differential Policy Gradient for i = 1, 2, .., k.*

Theorem 1. *The policy gradient for the compound policy μ in terms of differential policy gradients is given by,*

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

where $a_i = \mu_i(s|\theta^\mu)$ and $\mathbb{E}$ refers to expected value.

The proof for the above theorem is provided in the supplementary material[1](Theorem 1). There can be cases where a set of actions affect more than one tasks. We call this set of actions as shared actions a_s. For example, consider the case of the 27 DoF humanoid as shown in Fig. 1. It has a multi-chain architecture in upper body where a 5 DoF spine is shared between two chains (two arms). Thus, the spine contributes to reachability tasks of both hands. Therefore, we can say that the spine acts as a shared action for tasks that require coordination of both arms. In such cases where a set of actions a_s are shared between all the tasks, the policy gradient update mentioned in (1) needs to be modified. The following theorem and corollary, whose proofs are given in the supplementary material[1](Theorem 2 and Corollary 1 respectively), illustrate this modification.

Theorem 2. *Suppose a set of actions a_s are shared between all k tasks, then the updated policy gradient for compound policy μ is given by,*

$$\begin{aligned}\nabla_{\theta^\mu} J \approx &\sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)\nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu)] \\ &+\mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)\nabla_{\theta^\mu}\mu_s(s|\theta^\mu)]\end{aligned} \tag{2}$$

where $a_i^d = \{a_i - a_s\}$, $\mu_i^d = \{\mu_i - \mu_s\}$ and '$-$' represents set difference.

Corollary 1. *(Heuristic of Direction) Suppose a set of actions a_s are shared between all k tasks, then the updated policy gradient for compound policy μ in direction of Pareto front is given by,*

$$\begin{aligned}\nabla_{\theta^\mu} J \approx &\sum_{i=1}^{k} \mathbb{E}[\nabla_{a_i^d} Q_i(s, a_i|\theta^Q)\nabla_{\theta^\mu}\mu_i^d(s|\theta^\mu)] \\ &+\frac{1}{k}\mathbb{E}[\sum_{i=1}^{k} \nabla_{a_s} Q_i(s, a_i|\theta^Q)\nabla_{\theta^\mu}\mu_s(s|\theta^\mu)]\end{aligned} \tag{3}$$

where $a_i^d = \{a_i - a_s\}$, $\mu_i^d = \{\mu_i - \mu_s\}$ and '$-$' represents set difference.

B. Algorithm - DiGrad

Let's denote the reward corresponding to the i^{th} task as $r_i(s, a_i)$ and let γ be the discount factor. The RL agent

[1]Supplementary material can be found in $robotics.iiit.ac.in/people/phani.teja/Humanoids\_SM.pdf$

performs a compound action a on the environment in state s and moves to new state which we denote by s'. Now, critic network $Q(s, a|\theta^Q)$ is updated by minimising the following loss function:

$$L(\theta^Q) = \sum_{i=1}^{n} (Q_i(s, a_i|\theta^Q) - y_i)^2 \tag{4}$$

where y_i is the target given by:

$$y_i = r_i(s, a_i) + \gamma Q_i(s', \mu(s'|\theta^{\mu'})|\theta^{Q'})$$

and Q_i represents the Q-value corresponding to i^{th} task.

Update on actor network with learning rate η is given by:

$$\theta^\mu = \theta^\mu + \eta \nabla_{\theta^\mu} J \tag{5}$$

where $\nabla_{\theta^\mu} J$ is given by either (1) or (3).

In order to stabilize learning process, target networks $\theta^{Q'}, \theta^{\mu'}$ are used for both critic and actor as in DDPG. Target networks are slowly updated by using $\tau << 1$ as follows:

$$\begin{aligned} \theta^{Q'} &= \tau\theta^{Q'} + (1-\tau)\theta^Q \\ \theta^{\mu'} &= \tau\theta^{\mu'} + (1-\tau)\theta^\mu \end{aligned} \tag{6}$$

We use experience replay which addresses the issue of data being dependent as most of the optimization algorithms need samples which are sampled from identically independent distributions (i.i.ds). Hence, we use a replay buffer R which stores the data of every step. For training, we randomly sample from the replay buffer which ensures that samples drawn are from identically independent distributions.

Algorithm 1 DiGrad

1: Randomly initialise actor ($\mu(s|\theta^\mu)$) and critic network ($Q(s, a|\theta^Q)$) with weights θ^μ and θ^Q.
2: Initialize the target network with weights $\theta^{\mu'} \leftarrow \theta^\mu$ and $\theta^{Q'} \leftarrow \theta^Q$.
3: **for** i = 1 to E_{max}
4: Initialise random noise N for exploration.
5: Reset the environment to get initial state s_1.
6: **for** t = 1 to $Step_{max}$
7: Get action $a_t = \mu(s_t|\theta^\mu) + N$.
8: Execute compound action a_t and get the reward vector $\vec{r}_t$, which contains rewards of all the tasks.
9: Get the new state s_{t+1}.
10: Store transition $(s_t, a_t, \vec{r}_t, s_{t+1})$ in replay buffer B.
11: Randomly sample a mini-batch M from replay buffer B.
12: Update critic θ^Q according to Eq. (4).
13: Spilt sampled compound actions a_t into a_i, a_i^d and a_s.
14: Calculate differential policy gradients with respect to their corresponding sub-actions a_i,a_i^d,a_s as given in Eq. (1) or (3).
15: Update actor policy θ^μ according to the calculated differential policy gradient above.
16: Update the target networks $\theta^{\mu'}$ and $\theta^{Q'}$.
17: **end for**
18: **end for**

Complete algorithm and implementation details are provided in Algorithm 1. Our framework and training algorithm is very similar to DDPG except in following points. The major difference lies in the updates of actor as well as critic networks. In DiGrad, the actor network is updated according to the differential policy gradient theorem and critic update follows the modified loss function, shown in Eq. (4). Another difference is the way in which the actions are treated in DiGrad. In DiGrad, each action a_i has its own reward r_i unlike DDPG where a single global reward is used for the compound action. Thus, critic network in DiGrad approximates Q-value for all the tasks separately, unlike one Q-value for all the tasks in DDPG. Also, DDPG does not have the concept of a compound action. Lastly, the flow of gradients in the actor network is not uniform in DiGrad (and hence **Di**fferential Policy **Grad**ients) and differential policy gradient theorem presented in the previous subsection complies with this argument. In other words, the compound action a is just concatenation of all actions a_i, and individual (or different) gradients for each action a_i (or a_s) with respect to their corresponding Q_i flows into the actor network.

III. Learning Dual Arm coordinated Reachability Tasks

Learning dual arm coordination in humanoids with articulated torso using DiGrad is discussed in this section. Before discussing training process, we present the description of state vector and reward function modelling, as they are key to any RL algorithm. We also present the network architectures used for training and their corresponding hyper-parameters. We start off by discussing the environment used for training and then proceed to descriptions of above mentioned things in later subsections.

A. Environments

DiGrad was used to learn dual arm reachability tasks, where the humanoid robot learns to grab an object using both the hands, taking into consideration collision avoidance and stability criteria. The framework was tested in 4 different environments with varying difficulty levels. In each of these, the robot is trained to grab an object(cube/book) which is randomly sampled within its configuration space. The description is as follows:

1) A simple 3D environment with no obstacles: In this setting, the robot is trained to grab a cube in an empty surrounding with both hands (Fig. 4A).

2) 3D environment with randomly placed obstacles: This setting is similar to the above one, except for three sphere shaped obstacles which are included in the environment. Positions and sizes of these spheres are sampled randomly in each episode (Fig. 4B).

3) Cuboid on a Table: In this setting, environment consists of a table and objects to be grabbed are sampled at random positions on the table in each episode (Fig. 4C).

4) Book on a Shelf: This environment setting is similar to previous environment but even more complex, since valid configuration space of robot is highly restricted, due to a large increase in probability of collision. Here, the robot learns to grab a book placed on a shelf (Fig. 4D).

All the above mentioned environments are developed and trained in MuJoCo. MuJoCo provides accurate collision and position data, which is used to model our state and reward function. The training is carried out by executing compound actions provided by the actor, in MuJoCo and observing the obtained states and rewards. Collision is checked using internal functions in MuJoCo. The stability of the robot is checked using the relative position of center of mass (COM) of the robot with the ground level. Episode is terminated whenever stability check fails.

B. State and Action vectors

For all the environments mentioned above, state vector s consists of joint angles of upper body of humanoid, 3D coordinates of end effectors of both hands, 3D coordinates of goal positions and flags pertaining to stability, collision and task completion. In addition to these, state vector may have extra information, depending upon obstacle settings.

In this paper, we deal with two types of obstacle settings. Firstly, for obstacles whose position and size vary randomly (Environment 2), we include position and size of obstacles into the state vector s. Here, the number of obstacles is kept constant. Position and size of obstacles are randomly sampled in each episode and the robot is trained to reach the goal avoiding collisions with the obstacle along with maintaining stability. Secondly, we have considered the environment to be constant throughout where the obstacles are large static objects like shelf or table (Environment 3, 4). In this setting, the obstacle data need not be included in the state vector as it is constant.

Compound action vector a is concatenation of three action vectors a_r, a_l and a_{sp}, where a_r are the angular velocities of right arm joints, a_l are the angular velocities of left arm joints and a_{sp} are the angular velocities of spine joints (shared actions). From Theorem 2, it can be inferred that $a_i^d = a_i - a_s$ and here we have $a_1^d = a_r$, $a_2^d = a_l$ and $a_s = a_{sp}$. Therefore we consider two actions $a_1 = a_r \cup a_{sp}$ and $a_2 = a_l \cup a_{sp}$ with the corresponding reward functions r_1 and r_2. While training as well as testing we use the compound action a $(= a_1 \cup a_2)$, to provide angular velocities to all the active joints (upper body) of robot.

C. Reward Function Modelling

In order to apply DiGrad for cluttered environments, we have to take into account collision avoidance along with goal reachability. Further, we should ensure that both hands reach the goal positions simultaneously. Based on all these criteria we modelled the reward function as follows:

$$r_i = -\alpha dist_i + \begin{cases} -n_1 \ if \ (cols) \\ -n_2 \ if \ (instb) \\ +m_1 \ if \ (gb_i) \\ +m_2 \ if \ (goal_i) \end{cases} \tag{7}$$

where $cols$, $instb$, gb_i, $goal_i$ are flags referring to collision, instability, goal boundary and goal respectively and n_1, n_2, m_1, m_2 are positive constants.

In order to ensure that both hands reach the end goal position simultaneously, as needed for the coordinated tasks, we have taken the above reward function for each arm of humanoid and a very large positive reward κ is given when both hands reach their goals simultaneously, encouraging the robot to learn coordinated goal reachability. Hence,

$$r_1 = \kappa, r_2 = \kappa \quad if \ (goal_1 \quad and \quad goal_2) \tag{8}$$

With this definition of reward function, we move on to next subsection where network architectures and hyper-parameters used for training are discussed.

D. Network Architecture and Training

1) Network Architecture: In all the experiments, agents are implemented in a TensorFlow codebase. Both actor and critic networks consist of two fully connected hidden layers. Hidden layers consists of 700 and 400 hidden units with CReLu activation and a drop-out of 0.8. Batch normalization is used in actor network across all layers, but in critic network, it is used only in first layer. In critic network, L2 regularization of 0.01 is also used. The output layer has 13 outputs in actor with Tanh activation with and 2 outputs in critic with no activation. For training, critic takes state s and action vectors a as inputs and outputs 2 Q-values. Actor network takes state vector as input and gives out required angular velocities, a, for the 13 joints.

2) Training: For training, a learning rate of 0.0001 is used for both actor and critic. Replay buffer size is set as 45000 for all settings and a batch size is set to 64. A discount factor to be 0.999 is used. Training for each environment is run for at least 0.6 million steps and at most 1.5 million steps depending on the setting. For the environment without any obstacles, the training was run for 4000 episodes while for the remaining environments it was run for 10000 episodes. In all cases, each episode runs for 150 steps. Episode is terminated whenever the robot loses its balance or when goals are reached. In order to explore the workspace, an exploration noise N needs to be added to the action during training. In our case, we have taken a decaying random normal noise as N. Once training is complete, obtained solutions are tested without added noise. The solutions obtained are collision free and stable, but have oscillations due to the noise added during training. These oscillations are removed in post processing which is explained in next subsection.

E. Obtaining Joint Trajectories

Once the actor network learns the correlation between state and actions, final configuration for grabbing the required object is obtained in an iterative manner (in less than 150 iterations). Also, the robot do not collide or lose stability in any one of these iterations, providing a set of stable collision free way points. One can easily obtain trajectories (13 positional trajectories for 13 joints), by interpolating these

way points. However, linear interpolation will not provide smooth trajectory profiles because of the oscillations induced during training. Therefore, cubic smoothening splines [26] (piecewise cubic functions that are continuous and have continuous first and second order derivatives) are used in order to reduce these oscillations.

For most of the cases, a smoothening factor of 0.2 yielded smooth and collision free trajectories. In few cases, a constant smoothening factor led to collision. In order to handle such cases, a binary search is performed to find smoothening factor for each joint that ensures collision free trajectory. Joint trajectories thus obtained are followed using angular velocity controllers and results are shown in a dynamic simulation environment, MuJoCo.

Time Complexity: Since generation of way points involve only forward passes in a neural network, the major contributing factor is the time taken for calculating forward kinematics (FK) during these passes. Fastest known method calculates FK in $O(log\ n)$ [27], where n is number of joints. Time for post processing depends on the time taken to fit smoothening splines through the way points generated. Best known complexity for this, with m way points is $O(m)$ [28]. Hence, overall computation complexity of the proposed methodology is $O(m\ log\ n)$ and is appropriate for online motion planning.

IV. Experiments and Results

Our approach being model free has the advantage that it does not require a model of the environment since it learns the policy through its direct interactions with the real environment. It is extremely difficult to learn the detailed dynamics of complex systems like humanoids and hence, the simulated world used in model based algorithms may not be an exact replication of the real world. The approach that we have presented here is therefore, simpler to implement and updates will be faster as well as computationally cheaper than using a model based strategy. Several experiments are conducted to evaluate DiGrad and its robustness. After that, results of the proposed methodology for learning dual arm coordinated reachability in the aforementioned cluttered environments are presented.

A. Digrad

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

Heuristic mentioned above refers to the heuristic of direction presented in Corollary 1. We compare all of them with a standard DDPG setting. We use the same set of hyper parameters (Section III C) 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 the cases. More details about network architecture can be found in previous section. 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. We show the comparison of average reward as well the mean scores of each task in all the plots.

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.

We test our framework on a 27 DOF humanoid robot [23] shown in Fig. 1. This experiment involved reachability tasks of the 2 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 5 DOF whereas, the arms have 4 DOF each. Thus, the contribution of shared action (articulate torso) to the task is more than the non shared actions (arms).

Fig. 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 mean scores of tasks, we can see that single critic frameworks converge faster and are stable throughout the experiment as compared to the multi-critic frameworks. The best setting is again the single critic with heuristic, outperforming all the others in all the cases. Due to space constraints, further evaluations of DiGrad are presented in video. For learning dual arm coordinated reachability tasks, we used single actor - single critic architecture along with heuristic of direction. Results and simulations for the same are presented in next subsection.

B. Dual Arm coordinated Reachability tasks

1) Training Results: The humanoid was trained using DiGrad to learn dual arm coordinated reachability in four different environmental setting mentioned in section III. The training performance curves (average score vs episodes) for all the environments are shown in Fig. 3. From these curves, we can infer that the learning saturated at 4000 episodes for all the environments and the robot was able to reach the goal avoiding the obstacle in their respective environment.

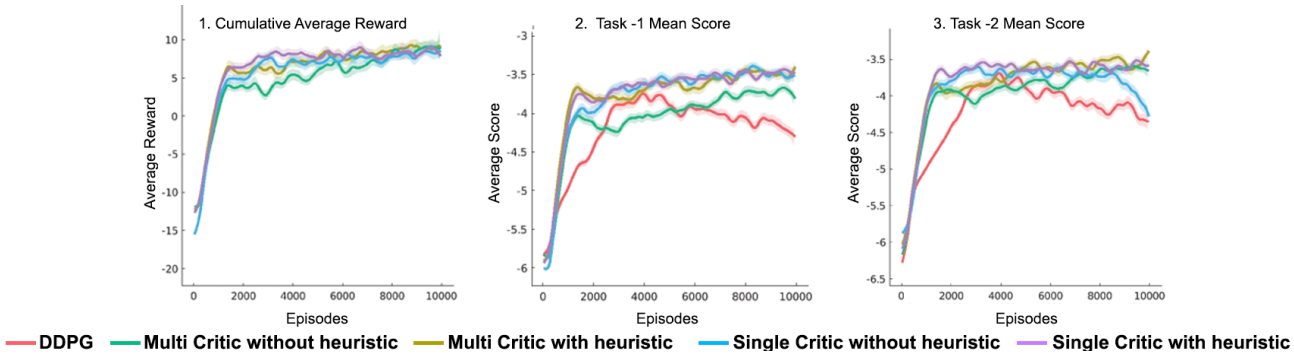
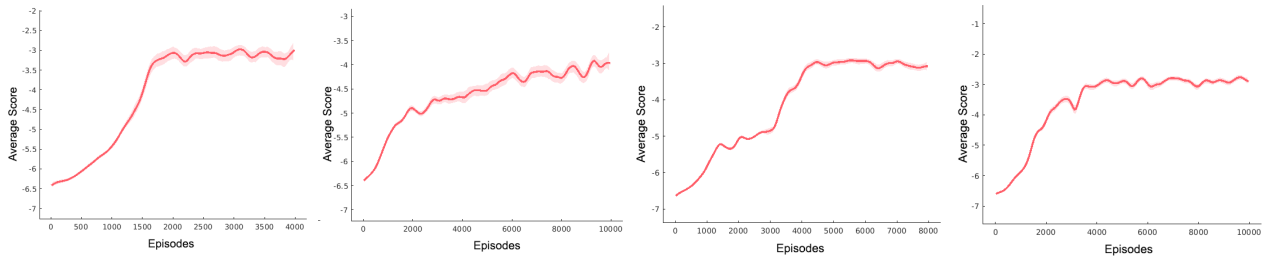


Fig. 2: Performance curves of reachability task experiments on humanoid with articulated spine. The bold line shows the average over 3 runs and the coloured areas show average standard deviation over the tasks. Note that, average reward curve is not plotted for DDPG as the reward function for it is different from DiGrad frameworks.



(a) No obstacles (b) 3 sphere-shaped obstacles (c) Table (d) Shelf

Fig. 3: Figure shows the performance curves of the training in different environment settings. The network architecture used here is single critic with heuristic. In all the graphs, the x-axis shows the number of training episodes. The bold line represents the mean value of the score and the coloured area around it shows the standard deviation of the score.

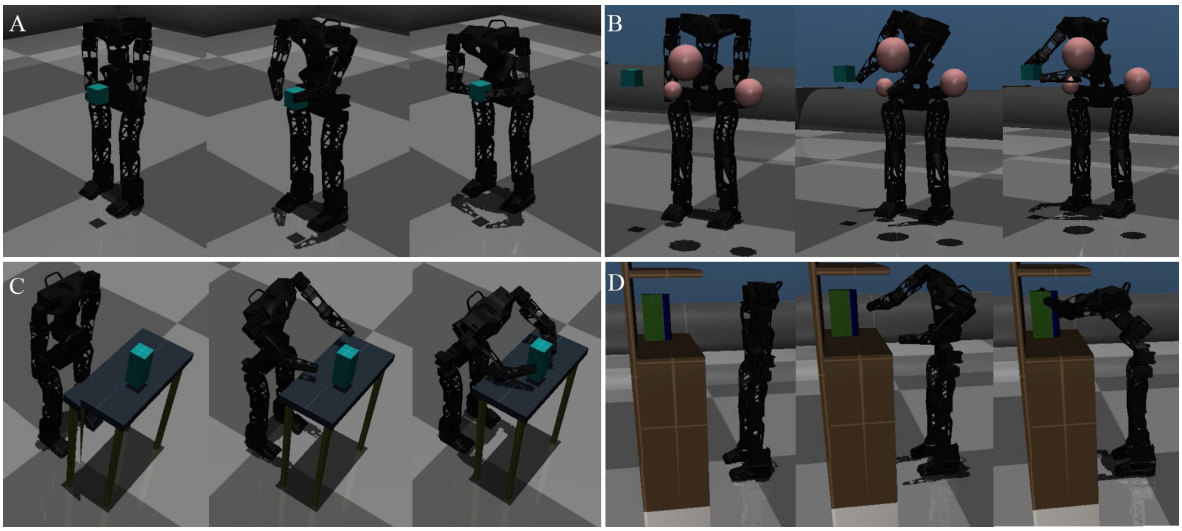


Fig. 4: Humanoid robot performing reachability tasks in different environments (A) No obstacles (B) Random Obstacles around the target position (C) Grabbing block from a table (D) Reaching out to a book located in a shelf

2) Trajectory Generation and Simulation Results: The joint trajectories are generated by taking way points obtained using DiGrad and then fitting smoothening splines. Fig. 1 shows these way points (blue dots) and corresponding trajectories generated (red and green) in Cartesian space. Although, smoothing has been performed on joint trajectories of all 13 joints, it can be observed that trajectories in coordinate space were also smoothed, showing correlation between coordinate space and joint space. In our framework, actor network learns this correlation by the end of training.

Fig. 4 shows the test results where the robot is trying to grab an object with both the hands in all the environments explained above. Each of these environments differ in the way collision avoidance is learnt by the actor. In case

of dynamic obstacle environment (Fig. 4B), the obstacle details are included in the state vector. Thus, the DiGrad network computes the trajectories keeping in consideration the obstacle position and size. In the case of Table and Shelf environments, the network memorize the entire workspace as the obstacles are constant. The trajectory generation in these cases needs to be precise as the obstacles are closely clustered near the goal object. It can be observed from all cases in Fig. 4 that the articulated torso plays an important role in performing reachability task as well as to main stability.

V. Conclusions

In this paper, the problem of learning dual arm coordinated tasks in humanoids with articulated torso was addressed. A novel RL framework, DiGrad was proposed in the context of manipulators with branched chains for learning collision free configurations, required to accomplish the given task. The way points (or intermediate configurations) to reach the final configuration were calculated using DiGrad and then joint trajectories were obtained by fitting smoothening splines through them. This methodology was shown to have a theoretical complexity of $O(m\ logn)$ and hence is suitable for online planning (n joints, m way points). DiGrad was tested with different settings and a comparative analysis with DDPG was presented. Finally, the proposed methodology was implemented in four different environment settings and simulation results were shown. However, few limitations to our approach remain. Most notably, all results presented were in the presence of static obstacles. The proposed method also has limitations when it comes to precision. Therefore, extending the proposed methodology for dynamic obstacle scenario with good degree of precision provides a good scope for future.

References

[1] C. Smith, Y. Karayiannidis, L. Nalpantidis, X. Gratal, P. Qi, D. V. Dimarogonas, and D. Kragic, "Dual arm manipulation—a survey," *Robotics and Autonomous systems*, vol. 60, no. 10, pp. 1340–1353, 2012.

[2] D.-H. Kim, S.-J. Lim, D.-H. Lee, J. Y. Lee, and C.-S. Han, "A rrt-based motion planning of dual-arm robot for (dis) assembly tasks," in *Robotics (ISR), 2013 44th International Symposium on*. IEEE, 2013, pp. 1–6.

[3] L. Guilamo, J. Kuffner, K. Nishiwaki, and S. Kagami, "Efficient prioritized inverse kinematic solutions for redundant manipulators," in *Intelligent Robots and Systems, 2005.(IROS 2005). 2005 IEEE/RSJ International Conference on*. IEEE, 2005, pp. 3921–3926.

[4] R. Diankov, N. Ratliff, D. Ferguson, S. Srinivasa, and J. Kuffner, "Bispace planning: Concurrent multi-space exploration," *Proceedings of Robotics: Science and Systems IV*, vol. 63, 2008.

[5] D. Bertram, J. Kuffner, R. Dillmann, and T. Asfour, "An integrated approach to inverse kinematics and path planning for redundant manipulators," in *Robotics and Automation, 2006. ICRA 2006. Proceedings 2006 IEEE International Conference on*. IEEE, 2006, pp. 1874–1879.

[6] M. Kallmann, A. Aubel, T. Abaci, and D. Thalmann, "Planning collision-free reaching motions for interactive object manipulation and grasping," in *ACM SIGGRAPH 2008 classes*. ACM, 2008, p. 58.

[7] S. Karaman and E. Frazzoli, "Incremental sampling-based algorithms for optimal motion planning," *Robotics Science and Systems VI*, vol. 104, p. 2, 2010.

[8] S. Dalibard, A. El Khoury, F. Lamiraux, A. Nakhaei, M. Taïx, and J.-P. Laumond, "Dynamic walking and whole-body motion planning for humanoid robots: an integrated approach," *The International Journal of Robotics Research*, vol. 32, no. 9-10, pp. 1089–1103, 2013.

[9] S. M. LaValle and J. J. Kuffner Jr, "Randomized kinodynamic planning," *The international journal of robotics research*, vol. 20, no. 5, pp. 378–400, 2001.

[10] N. Vahrenkamp, T. Asfour, and R. Dillmann, "Simultaneous grasp and motion planning: Humanoid robot armar-iii," *IEEE Robotics & Automation Magazine*, vol. 19, no. 2, pp. 43–57, 2012.

[11] N. Vahrenkamp, D. Berenson, T. Asfour, J. Kuffner, and R. Dillmann, "Humanoid motion planning for dual-arm manipulation and regrasping tasks," in *Intelligent Robots and Systems, 2009. IROS 2009. IEEE/RSJ International Conference on*. IEEE, 2009, pp. 2464–2470.

[12] M. Gharbi, J. Cortés, and T. Siméon, "A sampling-based path planner for dual-arm manipulation," in *Advanced Intelligent Mechatronics, 2008. AIM 2008. IEEE/ASME International Conference on*. IEEE, 2008, pp. 383–388.

[13] T. Asfour, D. N. Ly, K. Regenstein, and R. Dillmann, "Coordinated task execution for humanoid robots," in *The 9th International Symposium on Experimental Robotics (ISER 04*. Citeseer, 2004.

[14] O. Ly and P.-Y. Oudeyer, "Acroban the humanoid: playful and compliant physical child-robot interaction," in *ACM SIGGRAPH 2010 Emerging Technologies*. ACM, 2010, p. 4.

[15] 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.

[16] R. F. Reinhart and J. J. Steil, "Reaching movement generation with a recurrent neural network based on learning inverse kinematics for the humanoid robot icub," in *Humanoid Robots, 2009. Humanoids 2009. 9th IEEE-RAS International Conference on*. IEEE, 2009, pp. 323–330.

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

[18] J. Peters, S. Vijayakumar, and S. Schaal, "Reinforcement learning for humanoid robotics," in *Proceedings of the third IEEE-RAS international conference on humanoid robots*, 2003, pp. 1–20.

[19] J. Morimoto, G. Cheng, C. G. Atkeson, and G. Zeglin, "A simple reinforcement learning algorithm for biped walking," in *Robotics and Automation, 2004. Proceedings. ICRA'04. 2004 IEEE International Conference on*, vol. 3. IEEE, 2004, pp. 3030–3035.

[20] 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.

[21] 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.

[22] S. Phaniteja, P. Dewangan, P. Guhan, A. Sarkar, and K. M. Krishna, "A deep reinforcement learning approach for dynamically stable inverse kinematics of humanoid robots," in *Robotics and Biomimetics (ROBIO), 2017 IEEE International Conference on*. IEEE, 2017, pp. 1818–1823.

[23] D. Goel, S. P. Teja, P. Dewangan, S. V. Shah, A. Sarkar, and K. M. Krishna, "Design and development of a humanoid with articulated torso," in *Robotics and Automation for Humanitarian Applications (RAHA), 2016 International Conference on*. IEEE, 2016, pp. 1–5.

[24] 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.

[25] D. Silver, G. Lever, N. Heess, T. Degris, D. Wierstra, and M. Riedmiller, "Deterministic policy gradient algorithms," in *ICML*, 2014.

[26] C. De Boor, C. De Boor, E.-U. Mathématicien, C. De Boor, and C. De Boor, *A practical guide to splines*. Springer-Verlag New York, 1978, vol. 27.

[27] R. Wakatabe, Y. Kuniyoshi, and G. Cheng, "O (logn) algorithm for forward kinematics under asynchronous sensory input," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*, May 2017, pp. 2502–2507.

[28] M. Hutchinson, "Algorithm 642: A fast procedure for calculating minimum cross-validation cubic smoothing splines," *ACM Transactions on Mathematical Software (TOMS)*, vol. 12, no. 2, pp. 150–153, 1986.