A Deep Reinforcement Learning Approach for Dynamically Stable Inverse Kinematics of Humanoid Robots

by

Phaniteja S, Parijat Dewangan, Pooja Guhan, Abhishek Sarkar, Madhava Krishna

in

IEEE International Conference on Robotics and Biometrics 2017

Report No: IIIT/TR/2017/-1

Centre for Robotics
International Institute of Information Technology
Hyderabad - 500 032, INDIA
December 2017

A Deep Reinforcement Learning Approach for Dynamically Stable Inverse Kinematics of Humanoid Robots

Phaniteja S*
Robotics Research Center
IIIT Hyderabad
Hyderabad, India
phaniteja.sp@gmail.com

Parijat Dewangan*
Robotics Research Center
IIIT Hyderabad
Hyderabad, India
parijat.dewangan@research.iiit.ac.in

Pooja Guhan
Robotics Research Center
IIIT Hyderabad
Hyderabad, India
pooja.guhan@students.iiit.ac.in

Abhishek Sarkar
Robotics Research Center
IIIT Hyderabad
Hyderabad, India
abhishek.sarkar@iiit.ac.in

K Madhava Krishna
Robotics Research Center
IIIT Hyderabad
Hyderabad, India
mkrishna@iiit.ac.in

***Abstract*—Real time calculation of inverse kinematics (IK) with dynamically stable configuration is of high necessity in humanoid robots as they are highly susceptible to lose balance. This paper 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 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.**

***Index Terms*—inverse kinematics, deep reinforcement learning, humanoid, stability.**

I. 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

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

The inverse kinematics problem [1], [2] is to find a mapping from the end-effector coordinates to actuator space which can be represented as

$$\boldsymbol{\theta} = f^{-1}(\mathbf{x}) \tag{2}$$

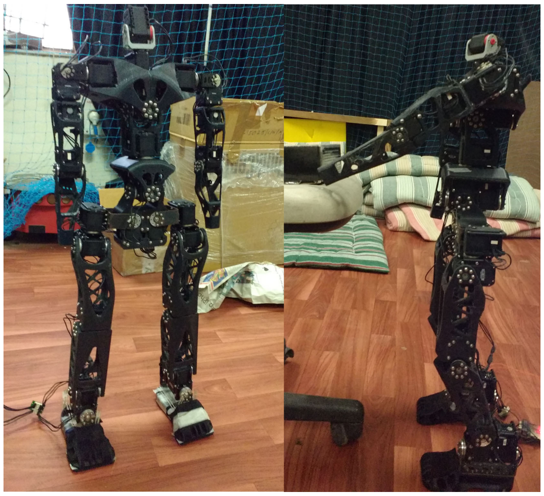


Fig. 1: Humanoid Robot with articulated torso

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.

Inverse kinematics for humanoid robots are important for applications like pick and place [3], physics engines [4]–[8] and human-robot interactions like tele-operating a robot to grasp objects [9], or execute a series of coordinated gestures [10], [11]. Inverse kinematic approaches can be broadly divided into two categories, namely closed-from analytical methods and numerical methods. Some examples of numerical methods are BFGS [12], pseudo-Jacobian inverse [1], [13],

*Equal contribution

[14], Jacobian transpose [1], [14], [15], Damped Least Square method (DLS) [14], [16], and Cyclic Coordinate Descent (CCD). Unlike closed-form analytical methods, the convergence time of numerical methods may vary and the results are not repeatable. On the top of that, computing inverse kinematics under constraints of stability and self-collision avoidance cannot be done efficiently in real time.

Recent advancements in RL [17] like Deep Q-learning (DQN) [18], Deterministic Policy gradients (DPG) [19], Guided Policy Search [20], Trust region policy optimization [21] and DDPG [22], [23] provide us many frameworks for not only learning the complex problem of IK, but also to optimize the required criteria. Among these methods, DDPG learns an efficient policy when the action space is continuous which is the case with inverse kinematics. 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.

In this paper we demonstrate how deep RL can be used to learn generalized solutions for inverse kinematics. 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 [24], [25].

The rest of the paper is organized as follows. In Section II, kinematic model of the robot is explained followed by the calculation of zero moment point (ZMP) and a brief description of general IK solvers. Section III explains DDPG algorithm and the proposed methodology to learn the stable IK solver. It also gives a brief description about the reward function used and the network architecture. Following that we show the results of training in Section IV along with numerical simulations. Finally conclusions and future work are discussed in Section V.

II. Kinematic modelling and Inverse Kinematics

The humanoid model used for study is shown in Fig. 1. The robot is small, with total height of 84 cm, total weight below 5 Kg and 27 DoF. The main highlight is its vertebral column (5 DoF articulated torso), which makes the biped more close to human. In this work, we have tried to explore the usage of articulated torso for performing reachability hand tasks.

A. Kinematic model of the robot

The kinematic model of the robot is represented using the D-H convention with the base frame at the right leg sole and the first joint angle starting from right ankle. Starting from the base, a coordinate frame is defined at each joint and at the end of each end-effector (hands and left leg). The complete kinematic model with axes numbering is shown in Fig. 2.

In Fig. 2, 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.

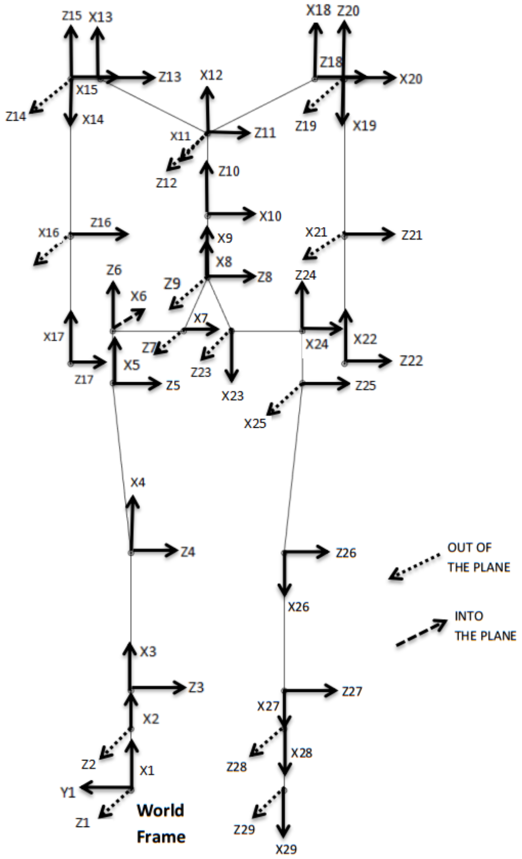


Fig. 2: Kinematic model of the robot

B. Stability and Calculation of ZMP

ZMP [26], [25] is one of the widely used dynamic stability measures which was proposed by Vukobratović and Stepanenko in 1972. In bipeds, a configuration is stable if ZMP lies inside the convex hull of the feet, which defines the support polygon [27].

In double support phase when the robot is stationary, ZMP is equal to the center of mass (CoM) projection in the support polygon. Whereas when the robot is not stationary, ZMP might deviate from the CoM projection. In order to have an accurate ZMP point we need to include momentum and angular momentum into the calculation. Hence the ZMP should be calculated as [28]

$$\begin{aligned} p_x &= \frac{Mgx - \dot{L_y}}{Mg + \dot{P_z}} \\ p_y &= \frac{Mgy + \dot{L_x}}{Mg + \dot{P_z}} \end{aligned} \tag{3}$$

where x, y are the x and y coordinates of 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 w.r.t base frame.

C. 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. 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 [29]. 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.

III. Deep Deterministic Policy Gradient (DDPG) for Inverse Kinematics

Reinforcement learning [17] can be used to train an agent which learns from the environment directly without the use of any external data. An RL agent takes an action (a) depending on its state (s) and observes the reward (r) given out by the environment. This process is repeated. The aim of the RL agent is to maximize the cumulative reward in any task. Hence the underlying reward function and its modelling plays a crucial role in RL.

A RL agent can learn different types of policies pertaining to the given task. A policy function represents the agent's behaviour needed to complete a given task. It is a mapping from state to action. Policy learning can be subdivided into two categories: 1) Stochastic 2) Deterministic. Stochastic policy learns the conditional probability of taking an action a, given that it is in a state s, $\pi(a/s) = P[A_t = a/S_t = s]$. A stochastic policy is useful only when number of actions are discrete and countable at any given state. In a continuous state-action space, this leads to a large number of possibilities and hence large memory and search time. A deterministic policy on the other hand learns the action as a function of state, $a = \pi(s)$. This function is non-linear in complex systems like humanoids and neural-networks serves as good model to learn such kind of functions. DDPG provides us with a very good frame work to train the neural-networks for learning highly non-linear functions.

A. Deep Deterministic Policy Gradient

DDPG uses the underlying idea of DQN in the continuous state-action space. It is an Actor-Critic Policy learning [30] 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 $Q(s, a, w)$ is learnt using Bellman equation [31] as in Q-learning [32], and the actor updates the policy in the direction that improves Q, i.e., critic provides the loss function for actor. In order to avoid the divergence of neural networks in Q-learning, target networks are used which track the original networks slowly.

Suppose $Q(s, a, w)$, $\mu(s, \theta)$ represent critic and actor networks respectively and $Q'(s', a, w')$, $\mu'(s', \theta')$ represent their target networks, the loss function for critic network can be given as

$$L_c = (r + \gamma Q(s', \mu(s', \theta'), w') - Q(s, a, w))^2 \tag{4}$$

where r is the reward and γ is discount factor.

The loss function for the actor is given as

$$L_a = \nabla_a Q(s, a, w) \nabla_\theta \mu(s, \theta) \tag{5}$$

The target networks are updated as follows with $\tau << 1$

$$w' = \tau w + (1 - \tau) w' \qquad \theta' = \tau\theta + (1 - \tau)\theta' \tag{6}$$

As it can be observed from Eq. 4, 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.

B. 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 angular 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. A 2 layered network consisting of fully connected layers with 400, 300 hidden units is used for both Actor and Critic. $cRelu$ [33] is taken as activation function and τ is taken as 0.001. Batch normalization is used in the network to avoid over-fitting and handle the scale variance problems.

C. 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. 7.

$$r = \begin{cases} -\alpha dist - \beta\sqrt{\sum_i (\Delta q_i)^2} & if\, stable\, and\, collision free \\ -\kappa & if\, unstable \end{cases} \tag{7}$$

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

Algorithm 1 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 $r, done$
```

D. Environment modelling and Training

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 1 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 τ as in Eq. 6. We used a Replay buffer of size 5×10^5. The pseudo code for training is given in Algorithm 2. 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. 4 and 5 respectively. The training results and their evaluations are shown in the subsequent sections.

Algorithm 2 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
```

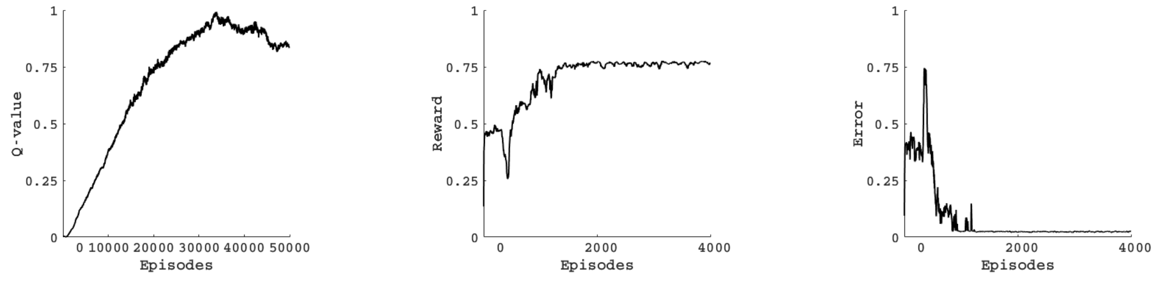
IV. Results and Simulations

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.

A. Training results

Figs. 3a and 3b show the normalized Q-value and reward of training. In Fig. 3a, 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. 3c. 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. 3b.

The trained model is tested for reachability tasks by giving random start configurations and goal positions. The accuracies



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

Fig. 3: Results of Training on 7.5 million steps

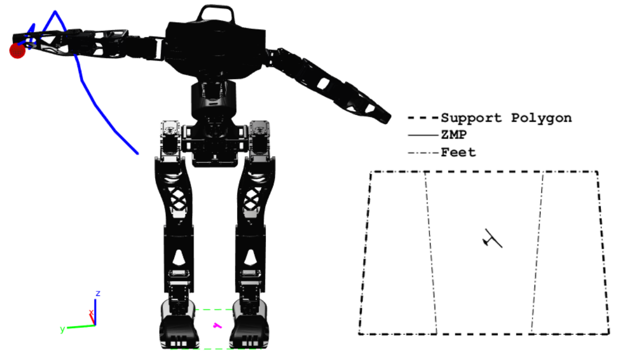
TABLE I: Performance of the IK Solver

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%

of the network obtained by using 100 random samples over 3 different seeds at different points of learning are documented in Table. I. 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.

B. Simulated Experiments

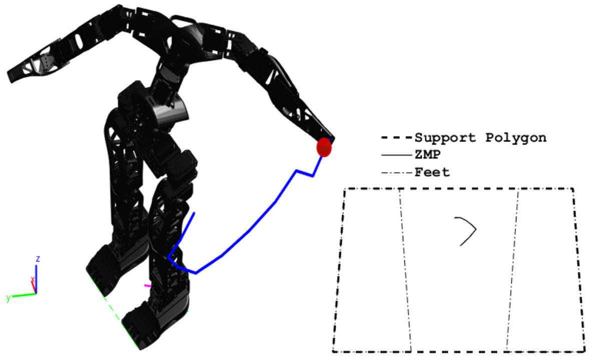
The trained IK solver is tested in the dynamic simulator of MSC Adams environment. The joint trajectories generated by the solver are given as input to the simulator for testing the solution. A set of three experiments which have high probability of losing balance are chosen in order to demonstrate the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



(a) (b)

Fig. 4: Trajectory and ZMP plot for Task 1

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. 4a. In the second task, as shown in Fig. 5a, it has to reach a 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. 6a.



(a) (b)

Fig. 5: Trajectory and ZMP plot for Task 2

Figs. 4a, 5a and 6a show the end effector trajectories along with the final posture of the robot. The corresponding ZMP plots for tasks are shown in Figs. 4b, 5b and 6b. 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. 4, 5 and 6.

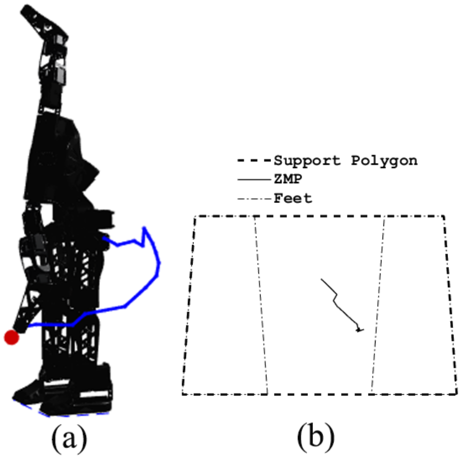


Fig. 6: Trajectory and ZMP plot for Task 3

V. Conclusions and Future work

This paper 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.

Future work includes increasing the accuracy of the trained model and learning more complex tasks which involves movement of legs.

References

[1] 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.

[2] 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.

[3] 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.

[4] 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.

[5] 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.

[6] 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.

[7] 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.

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

[9] 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.

[10] 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.

[11] 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.

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

[13] 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.

[14] 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.

[15] 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.

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

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

[18] H. Van Hasselt, A. Guez, and D. Silver, "Deep reinforcement learning with double q-learning." in *AAAI*, 2016, pp. 2094–2100.

[19] 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.

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

[21] 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.

[22] 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.

[23] 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.

[24] 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.

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

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

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

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

[29] 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.

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

[31] S. Peng, "A generalized dynamic programming principle and hamilton-jacobi-bellman equation," *Stochastics: An International Journal of Probability and Stochastic Processes*, vol. 38, no. 2, pp. 119–134, 1992.

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

[33] 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.