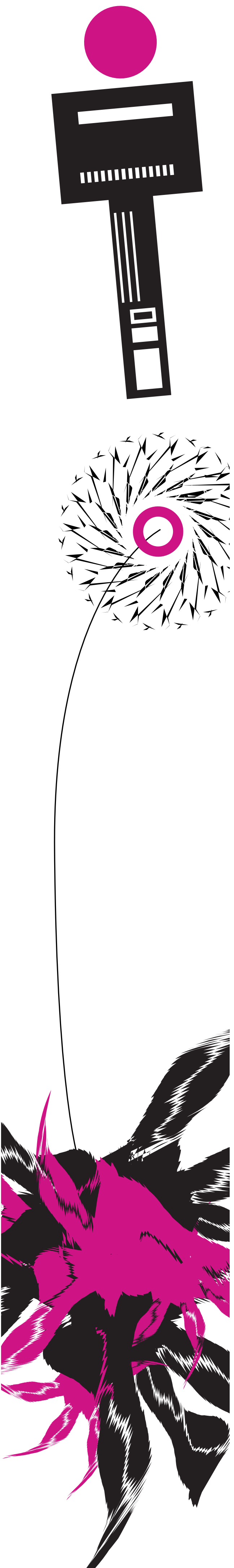


DYNAMIC ENERGY BUDGETING AND BEHAVIOURAL SHAPING ALONG COMPLEX TASK EXECUTION IN ROBOTICS USING REINFORCEMENT LEARNING

S.S. (Siddharth) Chaturvedi

MSC ASSIGNMENT

July 2021

049RaM2021
Robotics and Mechatronics
EEMathCS
University of Twente
P.O. Box 217
7500 AE Enschede
The Netherlands

Summary

As robots become more common in our day to day lives it becomes increasingly significant to design the software that controls them in ways that minimize the risk of human injury when physical human-robot interaction (pHRI) occurs. At the same time care must be taken that a safety centric software design does not hinder the robot's autonomy as it is essential to perform a task in an unstructured and dynamic environment.

Reinforcement learning (RL) is often used to achieve autonomy in robotics. One of the ways to achieve safe pHRI is by limiting the amount of energy that flows from the robot to the environment. Such limits can be easily included in the objective function of an RL framework. However, as RL uses black-box optimization algorithms to tune the robot's controller, limiting the energy used cannot be guaranteed. In order to limit the energy energy used in a deterministic way, *virtual energy tank* architecture is often used. However, these tanks are initialized with complete task energy which is often greater than the safety energy limit.

To this effect a new framework called the *Neural Energy Budget* (NEB) is proposed. NEB combines the standard RL framework and the virtual energy tank architecture such that they cancel out each other's shortcoming. The virtual energy tank guarantees limited amount of energy usage by the robot's actuators and a potential-based reward function from the RL-framework refills the energy in the tank if the robot performs well in the task. This dynamic refilling not only enables maintaining the energy in the tank below the safety limit but also provides an opportunity for the robot to escape the tank singularity condition by performing the task proficiently.

The proposed framework endows the robot's actuators with a new property of *Intelligent Selective Passivity* (ISP). Under which robot's actuators act passively with respect to the energy stored in the virtual tank when the robot performs poorly in the task and they act non-passively otherwise. Thus, this report also introduces and defines the concept of ISP. Further, application of the NEB framework is highlighted using three experiments. First experiment provides a proof of concept for the framework and introduces the ISP property. The second experiment shows how inclusion of virtual energy tank approach in an RL loop makes the robot's controller energy aware. Final experiment shows how the NEB framework can lead to memory efficient learning in an *infinite horizon* task.

Finally, the report shows how the proposed framework is a special case of the more general framework of *Homeostatic Reinforcement learning* (HRL). Following that, the main limitations of the framework are highlighted and an approach that can overcome the limitations in the future is presented.

Acknowledgments

I have been immensely supported and helped by many people during the course of my thesis. Words in this note may not be able to express the amount of gratitude I owe towards them.

Firstly, the guidance of my supervisor Dr. Federico Califano has been fundamental in not only chiseling the research idea presented in the work but also has been a source of optimism and confidence for me while working on the research. My time with him has sharpened me as a researcher. I am grateful for his guidance and input to the work.

I would like to thank prof. Dr. Stefano Stramigioli not only for heading the committee but also for his work in the field of energy based robotics. The presented thesis is a direct extension to his work in the field.

I would also like to thank Dr. Decebal Mocanu for graciously accepting to supervise me and be a part of the examination committee.

To my sister, mother and father, thank you for the unconditional love, support, inspiration, faith and care. They have worked really hard to bestow me with determination and freedom that I need to fulfill my aspirations.

I would like to thank all my friends and family for making this journey enjoyable and memorable.

My heart goes out to all the victims of the pandemic.

This work is in memory of my beloved Ba and Baba.

Siddharth
Enschede

Contents

List of Figures

List of Tables

1 Introduction

1.1 Context

In order to deliver the promise made by robotics regarding making every one's life easier it is essential for robots to leave the controlled environment of factories and slowly make their way into our lives. While, this process has already begun the key consequence of the process has been that physical human-robot interactions (pHRI) have increased a lot. Robots are also expected to continuously keep adapting according to the ever changing and unstructured environments in which they operate. These adaptations need to occur out of the simulations, in the real world, where pHRI is present. Thus, it has become more important than ever to research and come up with solutions which not only guarantee safety of humans in close proximity to the robots in an inference mode i.e. when robot has learned to perform the task but also in a learning mode i.e. when robot is still adapting its behaviour.

There exist multiple criteria that define safety in pHRI such as Head Injury Criterion (HIC), Maximum Power Index (MPI) and Maximum Mean Strain Injury Criterion (MMSIC). These criteria are achieved by limiting various physical quantities which govern the pHRI such as force, compression, velocity, energy and energy density (Tadele et al., 2014). Some safety criteria such as MPI directly depend on energy transfer while others can directly or indirectly be linked to a limit on energy transfer because other physical quantities ultimately also depend on energy transferred between the robot and its environment (Folkertsma et al., 2018). In terms of energy transfer, a safety criteria may dictate a hard limit on the amount of mechanical energy that can be transmitted by the robot to its environment per unit time. In this work it is considered that a robot executing a complex task contains two levels of control; a higher level, which decides what should the robot do next given the sensor readings; and a lower level, which maps the commands issued by the higher level to actuators of robot. A safety procedure can be implemented at both the levels.

At the higher level, the role of the controller is to make decisions about what action to take next based on which action will maximise an objective function. The objective function numerically defines which behaviours are desirable and how well the robot is performing in the task. Implementing the safety metric then simply becomes, adding the safety objective to the objective function, thus making it multi-objective. The higher level controller seeks the action that maximizes all the objectives of a multi objective function together. It is constructed by formulating the optimization problem as a dynamic-programming problem and employing optimal control (Lewis et al., 2012) techniques or reinforcement learning (RL) framework to solve it. Classical optimal control solutions are offline and require complete knowledge of the system dynamics. Therefore, they are not able to cope with uncertainties and changes in dynamics (Kiumarsi et al., 2017). The advantage of RL framework lies in the fact that certain methodologies of the framework allow a model free approach to formulate the controller and it is relatively easy to construct an objective function in the RL paradigm.

One of the drawbacks of RL framework is the fact that usually a "black-box" optimization is used in order to tune the higher level controller. Thus, safety metric may not be followed in a deterministic fashion i.e. given a situation there may be a command output from the higher level controller which violates the safety metric in the objective function. Another drawback is that the success of RL framework relies on enough and random exploration of state-action space, hence while learning the robot may explore and execute a state-action pair which is unsafe. Hence, if this exploration is not occurring in a simulation but on an actual robot system then the environment around it may become unsafe for pHRI. Unlimited discharge of energy may also lead to destruction of expensive equipment including the robot itself.

The *virtual energy budgeting* technique can provide a simple and elegant solution to the non-deterministic energy consumption problem that may occur in RL framework. At the lower level controller, this technique is frequently used to ensure the passivity with respect to energy supplied by the actuators to the robotic system (Folkertsma et al., 2018). In this technique, a fixed energy budget is assigned to a virtual energy tank in the beginning of a task and any energy used by the actuators is borrowed from this tank. Once the tank is empty, no more energy is allowed to be used by the actuators. This is done by decoupling the controller output from the actuators i.e. assigning either the force component or the velocity component of the actuators to 0. Hence, never more energy than what was present in the tank initially is used. This has been used by (Schindlbeck and Haddadin, 2015; Folkertsma et al., 2018; Raiola et al., 2018) to ensure passivity of robot's actuators. It must be noted that a system which is passive with respect to its total energy cannot produce energy by itself, its total energy is equal to the sum of initially stored energy and energy acquired or lost in interaction with its environment.

The energy tank method guarantees limited energy consumption of the system. However, guaranteeing safety is still challenging because usually these tanks are initialized with the complete task energy (Schindlbeck and Haddadin, 2015). Task energy is the total energy used by the robot to complete the task which is often much more than a safe energy limit if the task is complex and long. Thus, in such cases passivity with respect to energy in the tank may be achieved but safety criteria is still violated because the robot is capable of emitting more mechanical energy. Zult (2020) introduced an *Energy Freezing* technique to achieve safety with passivity by *freezing* the energy in the tank if an unsafe amount of energy is released and then gradually *melting* it i.e. making it available to be used again. However no distinction was made regarding the application of the used energy.

Another drawback of using task energy for tank initialization is that, entire task needs to be simulated beforehand in a simulation environment in order to determine the task energy as done in (Brodskiy, 2013). Such simulations need information about the robot's environment including quantities like friction coefficients, drag coefficients, wind speeds etc. Not only are these simulations computationally expensive but are also susceptible to integration errors. Precise estimation of task energy is needed because if the tank energy runs out before the completion of the task, the controller output will be decoupled from the actuators leaving the task incomplete. Further, even after precise estimation of task energy the environment of the robot may keep changing and so will the energy needed to do the task. It is to be noted that the decoupling of controller from actuator when energy in the tank runs out is important to guarantee limited energy consumption but this also limits the autonomy of the robot (Raiola et al., 2018)

1.2 Motivation

A clear methodology which limits the amount of energy in the tank below a defined safety metric at the same time avoids premature energy tank depletion by dynamically refilling it when needed, is missing. The methodology should also guarantee safety in inference and learning mode. In order to remedy the premature energy depletion problem small energy packets can be added to the system in controlled fashion by an external supervisor. This external supervisor should know when and how much energy should be added in the tank as too much and too frequent energy addition may lead to violation of safety criteria and too less energy will result in premature decoupling. Further, the external supervisor should not rely on computationally expensive task simulations.

In nature, in a very general sense we can observe that in order to safeguard the internal energy homeostasis, animals need to acquire enough amount of energy from their environment. The energy that they get is directly proportional to the food they consume. An animal can acquire adequate amount of food when it performs actions which lead to the food. These actions along with any other actions an animal may choose to perform in-turn utilize energy stored in the animal's body from previous meals. If the animal performs actions which lead to lesser than required or no amount of food for a prolonged amount of time then it will loose all the stored amount of energy and endanger its homeostasis which may ultimately lead to loss of life.

Thus, firstly the animal uses the stored internal energy to perform actions in its environment and then the environment rewards these actions with food corresponding to how efficient and good are the actions. This food is further converted to energy and added back to the internal energy completing the food-energy cycle. This cycle can be formalized using the "Dynamic Energy Budgeting" theory (Kooijman and Kooijman, 2010) and it provides a natural control system which not only prevents the animal from releasing infinite energy in the environment but also motivates the animals to take energy efficient actions which preserve its homeostasis. More importantly this cycle replenishes the depleted internal energy of the animal in proportional to how efficient the animal is in gaining food from the environment.

1.3 Research Statement

An animal striving to gain food in nature can be compared with a robot trying to excel at a particular task in an RL framework. Thus, the information about how well and efficient the animal is in collecting food corresponds to a possible reward function defining how well the robot is performing at the task in the RL framework. The main aim of the report is to exploit this correlation and propose a novel framework which can mimic the food-energy cycle observed in nature and replenish the energy in the energy tank based on how well the robot performs in the particular task while maintaining its level below a safety limit. The proposed framework has been named as the *Neural Energy Budgeting* (NEB) framework.

1.3.1 Objective

The NEB framework accomplishes the following objectives when used with a robot learning a complex task using reinforcement learning:

- *Universal Safety :* Throughout the task execution the energy in the tank is maintained below a safety limit in a deterministic way. This is achieved not only in inference mode but also in learning mode.
- *Flexible Energy Update :* In the proposed framework energy used to refill the tank is directly proportional to a possible reward function for the task. After certain assumptions and design choices which will be detailed later, using the reward function to refill the tank relieves us from estimating the task energy using simulations to initialize and refill the energy tank.
- *Intelligent Selective Passivity (ISP):* Refilling the tank using the reward function endows the system with a novel property that we name as *intelligent selective passivity*. Under this property the robot's actuators act passively when the robot fails to perform well in the task and act in a non-passive way when the robot performs the task well. This property provides a way for the robot to avoid tank singularity by continuously performing well at the task.
- *Complete Energy Awareness:* Embedding the energy tank approach within the reinforcement learning loop provides the robot's controller with additional internal states in the form of energy stored in the energy tank, energy leaving the tank and energy entering the tank. The robot's controller can now base certain complex decisions on these new states such as the question of affordability of an action.
- *Memory Efficient Learning:* In RL, in order to facilitate learning the trials in which robot attempts to do the task must be stored in a memory buffer and replayed later. Decoupling the controller from the actuators when energy runs out during the learning phase

provides a natural way to early-stop and restart the learning trial in an infinite horizon problem. As the energy entering the tank is directly proportional to the reward received by the robot the trials in which the robot fails to perform well end earlier than the trials in which robot performs well. This leads to more efficient learning when the memory in the buffer is limited.

1.4 Report Structure

The report ahead is structured as follows. Chapter 2 provides a background on reinforcement learning, passivity, virtual energy tanks and safety criteria in robotics. In Chapter 3 the proposed NEB framework is described and it is proved that the system features the novel property of ISP, which is further defined and discussed. A proof of concept for the first three objectives is provided in Chapter 4. Chapter 4 also comments about the relevance and meaning of hyper-parameters that govern the framework. Chapter 5 and Chapter 6 describe experiments which demonstrate, how the latter two objectives are accomplished by the framework. In Chapter 7 limitations and future work of the proposal are discussed. Finally the report is concluded in chapter 8

2 Background

2.1 Reinforcement Learning

In robotics, *higher-level controller* is a term often used to refer to the decision process responsible to generate the robot's next action based on the current state of the robot and its environment. Reinforcement Learning (RL) is a technique which can be used to tune the parameters of the higher-level controller such that the performance of the robot in a given task is optimal. In brief, the controller is tuned by trying out an initial set of controller parameters to do the task, collecting feedback and experience while doing the task and using them to gradually improve the parameter values over multiple trials.

The inspiration for this approach can be drawn from the fact that animals in nature tryout a series of actions in a particular situation, observe the reward value (like food, water, sex etc) they receive and if the reward value is satisfactory then the series of actions are repeated in similar situations, else, the series of actions are tweaked such that more reward can be gained in the future. This approach of learning resonates with the way in which cortico-basal ganglia system adapts behaviors according to the external environment by exploiting experienced environmental states and history of rewards (Sutton and Barto, 2011; Rangel et al., 2008). In this section, firstly the RL framework will be described in brief. Then an overview of the learning process used in the framework will be discussed. For a more detailed description refer to Bertsekas et al. (2000); Sutton et al. (1999).

2.1.1 The RL framework

An RL framework consists of an agent and its environment. The agent represents the higher level controller and maps a state vector $S_t \in R^n$ to action vector $A_t \in R^m$ at time t such that task performance of the agent is optimal for all S_t it visits. S_t is a vector of n scalar variables representing the information which is necessary and available to the agent at all t to decide the next action. For instance in robotics S_t can be a vector containing readings from all the sensors, distance from target, speed of actuators etc. A_t is the action vector representing all the m independent actions that the agent can take in the environment at t. In a torque controlled robot, A_t can be a vector of all torque commands issued by the controller to the m actuators.

A reward or score function can numerically define how good is it for an agent to be in S_t or to transition from S_t to S_{t+1}. Reward function R is constructed empirically and depending on its construction it can either map a subset of S_t or a subset of S_t and S_{t+1} to a scalar called reward value R_t. Conventionally, higher value of R_t corresponds to better task performance.

$$R(S_t) = R_t \text{ or } R(S_t, S_{t+1}) = R_t \tag{2.1}$$

The aim of RL framework is to optimize the agent by tuning its parameters such that the sum of R_t received by the agent in a trial is maximized.

During the learning mode, the environment provides the agent with R_t and S_t and in exchange it collects A_t from the agent. The agent uses R_t to optimize itself over multiple trials. Once the agent has been optimized then it behaves like a state-feedback controller in an inference mode wherein it simply maps S_t to A_t using the learned function mapping. The environment in RL framework uses A_t and other external factors to update S_t to S_{t+1}. A major assumption in RL framework is that the environment updates S_t according to a *Markov Decision Process* (MDP). An MDP is a discrete time process in which the next state of the system depends only on the current state and subsequent actions taken by the agents in the system (Puterman, 1990). In a simulation, it may be computationally expensive and infeasible to model an unstructured and dynamic environment hence in such cases the state updates may be stochastic. Mathematically in an MDP, the probability of state S_{t+1} at time $t+1$ can be stated as:

$$P[S_{t+1}|S_t, A_t] = P[S_{t+1}|S_1, .., , S_t, A_1, ..., A_t] \tag{2.2}$$

Where $P[S_{t+1}|S_t, A_t]$ is the probability of agent being in state S_{t+1} after taking action A_t when in S_t. The causality of the framework is highlighted in Figure 2.1

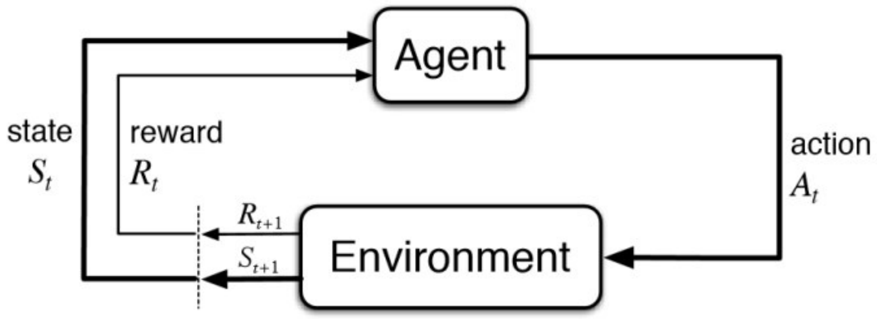


Figure 2.1: RL framework causality

2.1.2 The Learning process

At its core the agent comprises of 2 functions; the *policy* and the *state-value function* or the *action-value function*. Policy π is the function responsible for mapping S_t to A_t. It can either be modelled as a deterministic function ($A_t = \pi(S_t)$) or a probability distribution ($A_t \sim \pi(A_t|S_t)$) depending on the task. A deterministic policy directly maps S_t to A_t and a stochastic or probabilistic policy outputs a probability distribution over the action-space from which A_t is sampled. The aim of the learning process is to find the policy for which total reward received by the agent for a trial is maximum

When an agent is in a particular state S_t at t the *return-value* ($G_t \in R$) of the state refers to the total discounted reward the agent will receive at the end of the trial ($t = T$) starting from S_t. In the optimal control paradigm, this is sometimes called the *cost-to-go* (Lygeros, 2004).

$$G_t = R_t + \gamma R_{t+1} + \gamma^2 R_{t+2} \ldots + \gamma^{T-t} R_T \tag{2.3}$$

Where γ is a positive constant less than one representing the discounting factor. The significance of γ is to represent the uncertainty of reward values in the future and the fact that reward at present is more important than reward in the future. Higher value of γ represents less uncertainty of the future.

The agent uses either a state-value function or an action-value function to estimate the return value. The state-value function $v_\pi(S_t)$ maps S_t to an expectation of the return value ($E_\pi[G_t|S_t]$) if the agent is in a state S_t and is following a policy π, $v_\pi(S_t)$ is expressed as:

$$v_\pi(S_t) = E_\pi[G_t|S_t] \tag{2.4}$$

Using Equation 2.3 the state-value function can be decomposed into immediate reward value R_t plus discounted state-value function of the next state as.

$$v_\pi(S_t) = E_\pi[R_t + \gamma v_\pi(S_{t+1})|S_t] \tag{2.5}$$

On the other hand an action-value function $q_\pi(S_t, A_t)$ maps a state-action pair to an expectation of return value ($E_\pi[G_t|S_t, A_t]$) if the agent chooses action A_t at state S_t while following policy π.

$$q_\pi(S_t, A_t) = E_\pi[G_t|S_t, A_t] \tag{2.6}$$

Similar to value-state function, the action-value can also be decomposed into the immediate reward value and discounted action-value function of the next state and action pair.

$$q_\pi(S_t, A_t) = E_\pi[R_t + \gamma q_\pi(S_{t+1}, A_{t+1})|S_t, A_t] \tag{2.7}$$

Where A_{t+1} is the action taken in the next state according to π as well. It is to be noted that both v_π and q_π depend on π as the next state and reward value received by the agent also depend on π. The optimal value function $v_*(S_t)$ or $q_*(S_t, A_t)$ represent the maximum value function over all policies.

$$\begin{aligned} v_*(S_t) &= \max_\pi(v_\pi(S_t)) \\ q_*(S_t, A_t) &= \max_\pi(q_\pi(S_t, A_t)) \end{aligned} \tag{2.8}$$

The structure of value functions in Equation 2.5 and Equation 2.7 allow to solve the problem of finding the optimal value function by formulating it as *Bellman optimality equation* (Barron and Ishii, 1989). The optimal policy π^* refers to the policy corresponding to the optimal value functions.

$$\pi^* = \underset{\pi}{\operatorname{argmax}}(v(S_t)) \tag{2.9}$$

$$\pi^* = \underset{\pi}{\operatorname{argmax}}(q(S_t, A_t)) \tag{2.10}$$

Note the solution to Equation 2.9 can be found using dynamic programming solutions but it requires knowledge of the environment thus it is not a model free approach. On the other hand once $q_*(S_t, A_t)$ has been estimated then the optimal policy can be obtained simply by choosing the action which maximises $q_*(S_t, A_t)$ for each state. This does not require the knowledge of the environment dynamics. Thus using $q_(S_t, A_t)$ provides an opportunity for a model-free approach.

Actor-Critic methodology

The aim of the learning process is to obtain an optimal policy which in turn requires estimation of the optimal value function. If an optimal action-value function is known then one can simply choose the action which maximizes the value function at each step. However, a major assumption in this approach is that the actions from which one can choose are finite and known. Often in robotics the action space is continuous i.e. there are unlimited actions to choose from at each step so it becomes infeasible to simply select one action

Thus, often the learning process requires the learning of two function approximators: the *Actor* and the *Critic*. Actor is a function approximator used to approximate the policy, and is parameterized by θ. Critic is a function approximator with parameters ϕ, and it approximates the value function. The RL framework is termed *Deep-RL* framework when the Actor and Critic are represented using deep neural networks. A deep neural network is an artificial neural network with multiple hidden layer (Hopfield, 1988)

The learning algorithm tries to find the value of ϕ and θ for which firstly the value function is optimal and secondly the policy maximizes the optimal value function for all states. In a Deep-RL framework the two neural networks are tuned by determining their respective error functions and employing a gradient descent algorithm on their parameters to converge to values for which the errors are minimized. For Critic the error function L_c used in the report is termed as the "TD-error" (Sutton, 1988; Mnih et al., 2015), and is given by:

$$L_c = [R_t + \gamma \max_{A_{t+1}} q_\pi(S_{t+1}, A_{t+1}) - q_\pi(S_t, A_t)]^2 \tag{2.11}$$

Once an optimal value function is known the error function for the Actor L_A can simply become

$$L_A = -q_*(S_t, A_t) \tag{2.12}$$

It is important to note that reducing L_c in Equation 2.11 requires the selection of the next action A_{t+1} such that the corresponding action-value function is maximum. In other words the optimal policy is needed to reduce L_c. On the other hand from Equation 2.12 it is clear that an optimal action-value function is needed to converge to the optimal policy. Hence, an optimal policy is needed to determine the optimal value function and vice-versa. Most learning algorithms try to tune the parameters of Actor and Critic simultaneously.

2.1.3 Learning algorithm

The learning algorithms in which the parameters of the policy are adapted to achieve optimality are termed as *Policy Gradient Algorithms* (Sutton et al., 1999). There are numerous policy-gradient learning algorithms which exist to learn the Actor and the Critic (Weng, 2018). These algorithms can be divided into off-policy and on-policy algorithms. In the off-policy approach, the agent employs an exploratory policy, different from the target policy, to explore the environment and collect relevant data. This data is then used to optimize the target policy. While, in on-policy approach the agent uses the target policy, itself to explore the environment and collect data, which is used to optimize the same target policy. In the context of this report the Deep Deterministic Policy Gradient (DDPG) (Lillicrap et al., 2015) algorithm is used because of its ease of implementation.

DDPG is a model-free, off-policy algorithm. It borrows the idea of a deterministic policy formulation ($\pi_\theta(S_t) = A_t$) from the Deterministic policy Gradient (DPG) (Silver et al., 2014) algorithm. It also borrows the ideas of representing the Critic as a deep neural network and stabilizing its learning using experience replay and the frozen target network from the Deep Q-Network (DQN) (Mnih et al., 2013) algorithm. In experienced replay, data gathered during exploration is stored in a buffer and sampled randomly in order to be used for the tuning of the Actor and the Critic. In order to facilitate appropriate amount of exploration, the exploratory policy $\pi^e(S_t)$ is constructed by adding noise $\mathcal{N}$ to the current policy $\pi(S_t)$.

$$\pi^e(S_t) = \pi(S_t) + \mathcal{N} \tag{2.13}$$

Additionally, DDPG uses soft updates on the Actor parameters and Critic parameters:

$$\begin{aligned} \theta' &\leftarrow \tau\theta + (1-\tau)\theta' \\ \phi' &\leftarrow \tau\phi + (1-\tau)\phi' \end{aligned} \tag{2.14}$$

Where $\tau << 1$. θ' and θ are the parameters of the current Actor and the target Actor respectively. ϕ' and ϕ are the parameters of the current Critic and the target Critic respectively. The current networks are updated at each step of learning by step size determined by gradient descent but the target networks are updated using Equation 2.14 so that the values are constrained to change slowly. The complete DDPG algorithm is depicted in algorithm 1.

Algorithm 1: DDPG

```
Randomly initialize critic network $q(S_t, A_t)$ and actor $\pi(S_t)$ with weights $\phi$ and $\theta$
Initialize target actor $\pi'(S_t)$ and target critic $q'(S_t, A_t)$ with weights $\phi' = \phi$ and $\theta' = \theta$
Initialize replay buffer R
for episode=1,M do
    Initialize a random process $\mathcal{N}$ for action exploration
    Receive initial observation state $S_1$
    for $t$ = 1,T do
        Select action $A_t = \pi_\theta(S_t) + \mathcal{N}$ according to current policy and exploration noise
        Execute action $A_t$ and observe: reward $R_t$ and state $S_{t+1}$
        Store transition $(S_t, A_t, R_t, S_{t+1})$ in R
        Sample a random minibatch of N transitions $(S_i, A_i, R_i, S_{i+1})$ from R
        Set $y_i = R_i + \gamma q'(S_{i+1}, \pi'(S_{i+1}))$
        Update critic by minimizing the loss: $L = \frac{1}{N}\sum_i (y_i - q(S_i, A_i))^2$
        Update actor policy using sampled policy gradient:
        $\nabla_\theta J = \frac{1}{N}\sum_i \nabla_A q(S, A)|_{S=S_i, A=\pi(S_i)} \nabla_\theta \pi(S)|_{Si}$
        Update the target networks:
        $\theta' \leftarrow \tau\theta + (1-\tau)\theta'$
        $\phi' \leftarrow \tau\phi + (1-\tau)\phi'$
    end
end
```

2.2 Passivity

Dissipativity is a property that arises naturally in physical dynamic systems. Passivity forms a special case of dissipativity as introduced in Willems (1972). According to van der Schaft (2016) let a state space system $\sum$ exist with state S_t, input u_t and output y_t. Let, there also exist a real valued function $w(u_t, y_t)$ called the supply rate. This system is dissipative with respect to supply rate w if there exists a non negative storage function $E(S_t)$ such that:

$$E(S_{t_0}) + \int_{t_0}^{t_1} w(u_t, y_t)dt \geq E(S_{t_1}) \text{ , for } t_1 \geq t_0 \tag{2.15}$$

This inequality is called the *dissipation inequality.* It states that the storage function $E(S_{t_1})$ of $\sum$ at any future time t_1 is at most equal to the storage function $E(S_{t_0})$ at present time t_0, plus total external supply rate $\int_{t_0}^{t_1} w(u_t, y_t)dt$ during the time interval $[t_0, t_1]$. Hence there can be no increment in storage function due to "internal" dynamics.

Systems which are dissipative with respect to the supply rate $w(u_t, y_t) = y^t u$, are called passive systems. The supply rate and the storage function in Equation 2.15 can be arbitrarily selected in principle. However, when dealing with physical systems, the natural choice is to select the storage function as the total energy of the system and the supply rate as the power exchanged between the system and its environment (Folkertsma et al., 2018). Thus, if such a choice is made then passivity refers to the property due to which a system is incapable of actively producing energy by itself. A system which is passive with respect to it's power exchange with the environment is incapable of releasing more energy in the environment than what was already stored in it.

Systems comprising of only passive physical components like springs, dampers and masses can never contain more energy than the initial energy present and the energy transferred to them from thier environemt (Ortega et al., 2008; van der Schaft, 2016). Thus, a system comprising of only passive elements is also passive. Passivity is a desirable property while designing control systems because passive systems display the property of stability when input to the system is set to 0, since under mild conditions the storage function acts as Lyapunov function. Detailed discussion on passivity and $L2$-gain theory, dissipative systems, non-linear systems, stability can be found in van der Schaft (2016).

In many applications, robots are controlled in a non-passive way for instance using PID state controller or non-passive state feedback. This results in good performance in structured and known conditions. But, as soon as the system interacts with unstructured environment it can become unstable (Folkertsma and Stramigioli, 2017). Camlibel et al. (2015) showed that for any non-passive system, a passive system can be formulated such that interconnection between the two can give rise to unbounded behaviour. If the environment resembles such a passive system then that particular non-passive realization of robot controller can give rise to energetically unbounded system. This also results in the corollary that if a control system is passive then it is guaranteed to result in a stable system if it interacts with a passive environment (Folkertsma and Stramigioli, 2017).

2.3 Virtual Energy Tanks

In order to ensure passivity of a robot's actuators with respect to the energy they supply to the robot, the *Virtual Energy Tank* approach is often utilized. This approach directly limits the total amount of energy the robot actuators can pump into the system in a model free way. Assuming that the energy transferred by the actuators is coming from a virtual energy source, called the virtual energy tank, and by limiting the initial energy stored in this source, a hard limit on the energy used by the actuators can be enforced. Amount of energy used by the actuators at every instance can be estimated and depleted from the energy tank. This concept was first introduced by Duindam and Stramigioli (2004). Thus, first an energy budget is defined by initializing the energy tank with an initial amount of energy before the task execution begins, then the amount of energy used by the actuators is subtracted from this budget as the task progresses.

$$\begin{aligned} E_0 &= C \text{ initial energy budget} \\ E_{t+1} &= E_t - \triangle E_t \text{ energy update} \end{aligned} \tag{2.16}$$

Where E_t is the amount of energy in the tank at time t and $\triangle E_t$ is the energy used by the actuators between time t and $t+1$. On comparing Equation 2.16 with Equation 2.15, E_t can be considered as the storage function for the actuator and $\triangle E_t$ as the time integral of the supply rate. Passivity with respect to E_t is then achieved by setting the power output of the actuator to 0 when E_t falls bellow 0. For an actuator in a robotic system, this can in principle be done by setting either the force component or the velocity component to 0. Thus never more energy than initially present in the tank is used (Folkertsma et al., 2018).

As highlighted in Figure 2.2 an additional passivity layer can be inserted between the controller and the actuator in order to passivise any controller using the virtual energy tank approach. When $E_t > 0$, the passivity layer allows the velocity or force commands of the controller to be carried out by the actuator in the environment, but when $E_t \leq 0$ the passivity layer sets either the force command or the velocity command to 0. This process refers to the decoupling of the controller from the actuator. This decision process is shown in Figure 2.3

2.4 Energy Sampling

Digital control systems are discrete time systems. In a discrete time system, in order to implement the virtual energy tank approach described in Section 2.3, energy used by the actuators at every instance must be estimated. This can be done by measuring the electrical power supplied to the actuator. However, a considerable proportion of this energy is converted to heat, rather than the mechanical energy used by the actuator. In Stramigioli et al. (2005) it is shown that it is possible to accurately estimate the mechanical energy supplied by the actuator if fol-

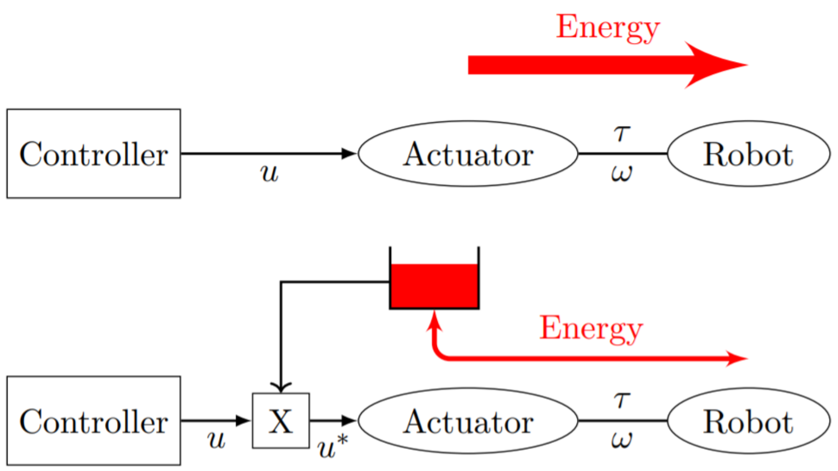


Figure 2.2: Adopted from Folkertsma and Stramigioli (2017), Actuators can inject unlimited amounts of energy into the system if no special care is taken. For passivity and stability, the robot system can be equipped with a virtual energy tank from which all energy that goes into the system must come. Any controller can be augmented with this passivity layer. The "X" block modulates the actuator signal as described in Figure 2.3

lowing assumptions are true: 1) The actuator force or torque remains constant over a sampling period. 2) A position sensor is collocated with the actuator. The second condition is a design choice. The first condition is true if the actuator is current-controlled and the sampling time of the sensor is fast enough compared to the controller's command update time.

Most digital control systems utilize a Zero-Order-Hold (ZOH) to convert the discrete output into a continuous signal. ZOH implies that the output from the controller will remain constant until the next command is issued from the controller. Thus, between subsequent time instances t_0 and t_1 the energy $\triangle E_{t_0}$ supplied by the actuator becomes:

$$\triangle E_{t_0} = \tau_{t_0} \cdot [q_{t_1} - q_{t_0}] \tag{2.17}$$

where τ_{t_0} is the value of force or torque output of the controller at t_0 which is held constant till t_1 due to ZOH, q_t is the position measurement (linear or angular displacement) of the motor at t obtained from the position sensor. The complete derivation and discussion about Equation 2.17 can be found in Folkertsma and Stramigioli (2017); Folkertsma et al. (2018). Further, τ_t can be substituted using $\tau_t = K_m \cdot i_t$, where i_t is the current in the motor windings at time instance t. From Equation 2.17 and Equation 2.16 we get :

$$E_{t+1} = E_t - \tau_t \cdot [q_{t+1} - q_t] \tag{2.18}$$

One important point to note when using energy sampling approach in Equation 2.18 is that at any given time t, τ_t should be executed by the controller in it's environment in order for the position sensor to sense q_{t+1}. Thus, τ_t needs to be executed before $\triangle E_t$ can be estimated using the energy sampling approach and following that the passivity layer of the virtual energy tank architecture can ensure passivity. This is a drawback of using a model free approach to estimate the energy used by the actuator. The consequence of this on the passivity layer is that if $E_t > 0$ for any t, a command issued by the controller is executed in the environment irrespective of how much energy is demanded by the actuator for the command and it's effect on the energy tank can be estimated only after it has been executed in the environment. Thus, if the energy sampling approach is used in the framework then passivity can be guaranteed only across two time steps.

A solution to this drawback requires us to give up the model-free approach to estimate energy flowing out. A forward model can be used to map torque commands directly to the maximum

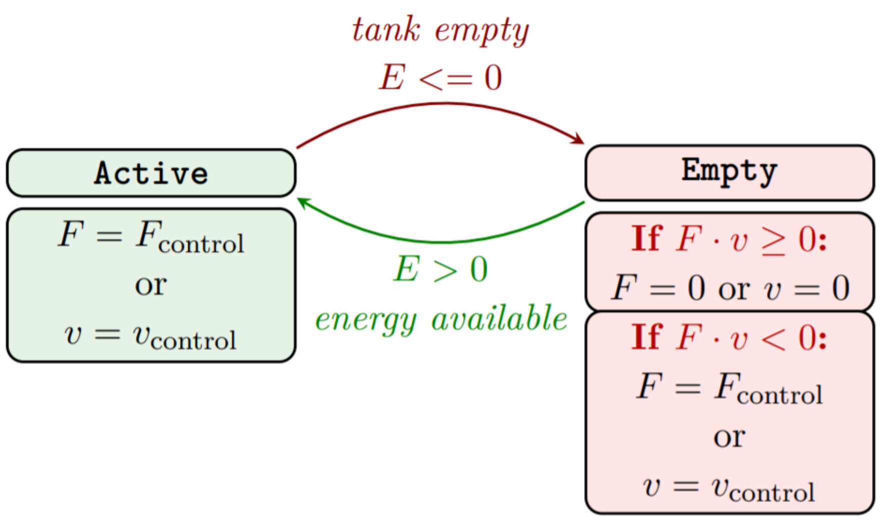

tank empty
$E <= 0$
Active
$F = F_{\text{control}}$ or $v = v_{\text{control}}$
Empty
If $F \cdot v \geq 0$: $F = 0$ or $v = 0$
If $F \cdot v < 0$: $F = F_{\text{control}}$ or $v = v_{\text{control}}$
$E > 0$
energy available

Figure 2.3: Adopted from Folkertsma and Stramigioli (2017), A Finite State Machine implementation that, given the current energy tank level E, prevents further energy injection if the tank is empty

kinetic energy that they will generate, given a robot's construction. However in the context of this report it is assumed that the sampling time and the maximum torque allowed by the actuators are small enough to not lead to an infinite energy release in one time step.

2.5 Safety in pHRI

Defining a safety criterion in physical human robot interaction (pHRI) system is a non-trivial task because a human body is capable of experiencing a variety of injuries including bone fracture, internal injuries, abrasions and lacerations as a result of being cut, stabbed, gashed or crushed (Bustraan, 2021). The conditions under which each of these forms of injury can occur have been studied by Haddadin and Croft (2016) and Tadele et al. (2014) presented a thorough account of literature review about safety metrics in PHRI systems. Tadele et al. (2014) also showed that safety criterion can be based on different physical quantities like force, compression, energy, energy-density and stress. In the context of this report the criterion which limit the rate of energy flow from actuators are relevant.

When considering a limit on energy flow in order to ensure safety it is important to consider which parts of human body are likely to collide with the robots. It is also important to consider the conditions under which such collisions may take place for instance the body part may be constrained or un-constrained when the collision occurs (Laffranchi et al., 2009). Different body parts need different safety requirements but the head, the neck and the chest region are the most important parts for which safety requirements should be strict as serious injury to these parts can be fatal. For instance according to Tadele et al. (2014) the energy that causes a human skull fracture per volume of skull was given as 290 kJ/m^3 and 160 kJ/m^3 for an adult and an infant respectively. Laffranchi et al. (2009) estimated that about 517 J of energy is needed to break a typical human skull. The amount of energy that can result in damage to spinal chord in neck region is estimated to be 35J by Yoganandan et al. (1996). Tadele et al. (2014) suggested a limit on rate of kinetic energy transfer when the area of impact is abdomen and chest.

3 Methodology

3.1 Introduction

This section will introduce and describe the proposed framework in detail. As discussed in Section 2.2 a passive control architecture leads to stable interactions which can be guaranteed only in a passive environment. However, for most complex tasks a robot needs to perform, the environment by definition is unknown hence passivity cannot guarantee stable interaction in every case. Further, a completely passive controller may need not necessarily lead to safe interactions. Also, when dealing with a complex task in an unstructured and dynamic environment a completely passive controller can negatively effect task efficiency since, in these situations active release of energy to complete the task becomes fundamental. Thus, in the proposed framework the idea of universal passivity of the controller is abandoned and the goal of implementing a definite safety criterion and controlled non-passivity when needed is adapted.

Chapter 1 discussed some shortcomings of the RL framework regarding the effect of black-box optimization on safety in pHRI and the effect of sudden decoupling of controller from the actuator on robot's autonomy when using a virtual energy tank scheme. The proposed framework combines the RL framework and the virtual energy tank method such that they cancel out each other's shortcomings when dealing with safety and energy awareness. The energy tank architecture guarantees safety in pHRI by limiting the energy released by the robot's actuators below a safe amount. On the other hand, a possible reward function designed using RL methodology, dynamically refills the energy tank such that robot's actuators keep receiving adequate amount of energy as long as the robot keeps performing well in the task. This action also helps maintain the energy in the energy tank below a safety limit throughout the task. Inclusion of virtual energy tank architecture in an RL loop provides a set of new internal states and internal motivation to the robot's controller which it can now use to attain desired energy aware behaviours. The robot's controller is represented using an RL agent in the proposed framework. Since, this framework uses robot's current task performance measure, borrowed from the RL paradigm, to augment the energy budget for robot's actions in the next timr step, it has been named as the *Neural Energy Budgeting* (NEB) framework

3.2 Energy tank update

In the beginning of each trial the tank is initialized by a positive constant c J amount of energy which is less than the safety energy limit of E_{max}. E_{max} can be decided empirically for a setup depending on the conditions in which pHRI occurs as discussed in Section 2.5. For instance as mentioned in Yoganandan et al. (1996) it can be 35 J which is the minimum energy needed to break the spinal chord in certain situations. If E_t denotes the energy in the tank at time t, then it can be initialized at $t = 0$ as:

$$E_0 = c\ ,\ c < E_{max} \tag{3.1}$$

The energy tank is updated at each time step with the total energy going out of the tank E_t^{out} and energy coming in to the tank E_t^{in} at t. These updates occur in discrete time steps with sampling time of the update being equal to the robot controller's sampling time.

$$E_{t+1} = E_t - E_t^{out} + E_t^{in} \tag{3.2}$$

3.2.1 Energy out

The total energy going out of the tank E_t^{out} is equal to the sum of energy used by each actuator of the robot at time t. In the context of this report it is assumed that robot's actuators can operate in the *motoring* mode and in the *braking* mode. An actuator is said to be in motoring mode when the force or torque produced by the actuator and the velocity of the corresponding degree of freedom (DOF) are in the same direction i.e the dual product of force or torque and velocity is greater than 0. The actuator is in braking mode when the force or torque and velocity are in opposite directions i.e their product is negative. This can be seen in Figure 3.1.

The energy depleted from the tank $E_t^{out,i}$ corresponding to each actuator i, when it operates in the motoring mode is equal to the mechanical energy transferred by it to the robot and is calculated using the energy sampling method described in Section 2.4:

$$E_t^{out,i} = \tau_t^i \cdot (q_{t+1}^i - q_t^i) \tag{3.3}$$

where τ_t^i is the force commanded by the robot's controller to the actuator and q_t^i is the position of the corresponding DOF, as sensed by a collocated displacement sensor. τ_t^i can be the linear force or torque generated by the actuator depending on if the actuator is linear or rotational. Similarly, q_t^i can be the linear displacement or angular displacement depending on the type of actuator.

According to the conventional virtual energy tank approach, when actuator i operates in the braking mode the mechanical energy estimated using Equation 3.3 is negative and gets added back to the tank (Folkertsma et al., 2018). However, in the NEB framework it is assumed that $E_t^{out,i} \geq 0$ for all actuators i. The reason for this assumption will become clear in Section 3.3.2 when the augmented reward function for the RL agent used in the proposed framework will be described. Thus in the braking mode, the mechanical energy generated due to actuation is assumed to be 0, instead a different positive energy which is analogically similar to the metabolic cost paid by animals to activate their muscles is depleted from the tank. This metabolic energy is given by:

$$E_t^{out,i} = m \cdot (\tau_t^i)^2 \tag{3.4}$$

Where m is a positive constant which can be designed empirically and τ_t^i is the force command similar to Equation 3.3. Thus for each actuator i the energy going out of the tank is given by:

$$E_t^{out,i} = \begin{cases} \tau_t^i \cdot (q_{t+1}^i - q_t^i), & \text{if } i \text{ operates in motoring mode} \\ m \cdot (\tau_t^i)^2, & \text{if } i \text{ operates in braking mode} \end{cases} \tag{3.5}$$

Finally the total energy flowing out at t can be calculated by summing the energy used by all the actuators at t:

$$E_t^{out} = \sum_i E_t^{out,i} \tag{3.6}$$

3.2.2 Energy in

This is the novel feature and the non-passive step in the NEB framework. At the end of each time step, the energy supplied to the tank is directly proportional to how well the robot performs in the the task in that time step. This is inspired by the fact that in nature animals receive energy in proportion to the food they acquire. The robot's task performance in RL framework is numerically gauged by a reward function hence a possible reward function is used to refill the tank. There are multiple ways in which a reward function can be designed for a given task but in the proposed framework it is assumed that the reward function used to refill the energy tank is a potential-based reward function as introduced in Ng et al. (1999). If a robot visits a state S_{t+1} right after a state S_t then a potential-based reward function $P(S_t, S_{t+1})$ outputs the difference in a potential function $V(S)$ at points S_t and S_{t+1}.

$$P(S_t, S_{t+1}) = V(S_t) - V(S_{t+1}) \tag{3.7}$$

where $V(S)$ is a continuous potential function defined over the space spanned by S_t with the global minima located at the desired target state of the robot. For instance $V(S)$ can be the

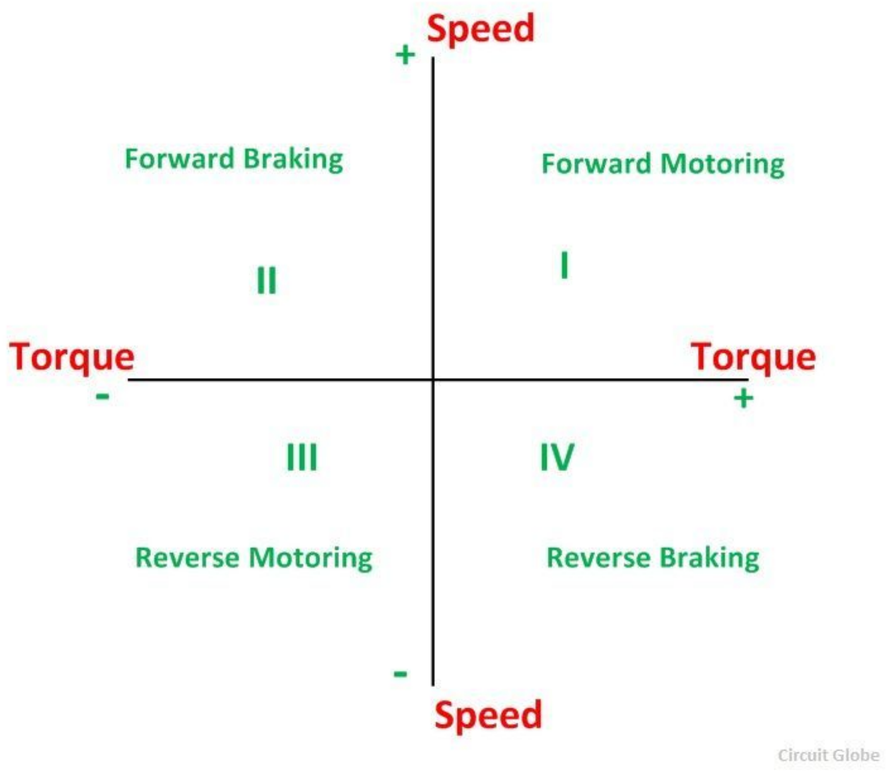


Figure 3.1: The four quadrants of operation of an electric drive.

euclidean distance between robot's current position and the target location hence V(S) will be 0 at the target and keep increasing if the robot keeps moving away from the target. Now, the energy supplied to the energy tank is proposed to be directly proportional to $P(S_t, S_{t+1})$:

$$\begin{aligned} E_t^{in} &\propto P(S_t, S_{t+1}) \\ E_t^{in} &= K \cdot P(S_t, S_{t+1}) \end{aligned} \tag{3.8}$$

Hence, in a simple scenario, energy entering the tank can be K times $P(S_t, S_{t+1})$ where K is a positive constant called the energy coefficient. It should be added that the linear mapping used in Equation 3.8 can be replaced with any monotonic function $F(P(S_t, S_{t+1}))$ that increases and decreases with $P(S_t, S_{t+1})$. Here F can be referred to as a feeding function. In order to limit the energy inside the energy tank within the safety limit E_{max} we limit the value of E_t^{in} to $E_{max} - E_t$

$$E_t^{in} \leq E_{max} - E_t \tag{3.9}$$

In order to estimate the maximum value of E_{t+1} we can simply substitute the lower limit of E_t^{out} ($E_t^{out} \geq 0$ from Equation 3.5) and upper limit of E_t^{in} (from Equation 3.9) in Equation 3.2. On doing so, it can be observed that $E_{t+1} \leq E_{max}$ for all t.

Lastly, an additional condition which states $E_t^{in} \geq 0$ can be included so that the system does not become too conservative.

3.2.3 Passivity Layer

In the proposed framework, limited energy consumption by actuators is guaranteed by borrowing the energy check procedure implemented in the passivity layer from conventional virtual energy tank architecture as described in Section 2.3. In the passivity layer actuator is decoupled from the controller commands as soon as $E_t \leq 0$. This is done in principle by assigning a zero value to either the force component or the velocity component of the actuator. If this condition is satisfied in the learning mode of the NEB framework, then the trial is terminated and the robot's controller receives a huge negative reward for reaching this condition so that it can learn to avoid it. In the context of this report we have:

$$A_t^* = \begin{cases} A_t & \text{if } E_t > 0 \\ 0 & \text{if } E_t \leq 0 \end{cases} \tag{3.10}$$

where A_t is the torque command issued by the robot's controller and A_t^* is the output of the passivity layer.

3.3 RL augmentation

In Section 3.2 it was described how the energy tank approach can be augmented to include dynamic energy tank update. In this section, changes in the RL framework that lead to energy awareness of the robot's controller are discussed. It is desirable for the controller to be aware about the current energy level in the tank so that it can base certain decisions on energy used by the actuators such as learn to avoid situations which lead to tank singularity.

As highlighted in Section 2.1, in an RL framework, policy is a function that maps the state vector S_t of the robot to an action vector A_t. The policy can be represented by any function approximator from a simple look up table to a deep neural network (Arulkumaran et al., 2017). The RL framework provides a way to tune the policy parameters such that an objective function which mathematically defines the proficiency of the robot at the given task, is maximized for all S_t.

3.3.1 State Augmentation

The state vector S_t represents all the information about the environment that the robot needs to make a decision. In a conventional robotic task it may include information such as distances from obstacles, distance from the target position, speed of actuators etc. Using energy tank approach in the loop with RL framework enables us to extend S_t by including states based on energy tank such as E_t, E_t^{in} and E_t^{out} . Including E_t^{in} and E_t^{out} can help the RL agent answer the question of affordability (Beeler and Mourra, 2018) i.e. how affordable is a particular action in-terms of energy spent and gained in doing it.

3.3.2 Reward Augmentation

In the NEB framework the reward function R_t used for training the policy can be different from the one (P) used in Equation 3.8 to refill the tank. In general, a reward function maps a subset of S_t or a subset of 2 consecutive states S_t and S_{t+1} to a scalar quantity that determines how good is it to be in S_t or how good is to move to S_{t+1} from S_t. A reward function which rewards progress towards multiple independent objectives is a multi-objective reward function. It is constructed by adding reward functions designed for individual objectives in proportion to their importance.

$$R_t = \sum_i \alpha_i \cdot R_t^i \tag{3.11}$$

where R_t^i is the reward function for i^{th} objective and α_i is a coefficient which determines it's importance compared to other reward functions. With the inclusion of energy states is S_t the policy can now be tuned to demonstrate certain desired behaviour based on internal energy. For instance supplying a negative reward in proportion to the difference between E_t and E_{max} can motivate the RL agent to become energy efficient and avoid energy tank singularity while doing the task. A reward function R_{tank} can be designed such that:

$$R_t^{tank} = E_t - E_{max} \tag{3.12}$$

Note that E_t is a subset of S_t. It is important to understand here that when such a reward function is designed based on energy in the virtual tank then the energy tank no longer just serves the purpose of an energy limiting system but also becomes the storage for internal energy of the RL-agent which the agent seeks to increase. Special attention needs to be paid on how energy enters the tank in order to obtain a predictable behaviour. For instance if energy was allowed to be added to the tank when the actuators operate in braking mode then the RL agent would seek to gain reward by operating predominantly in braking mode.

R_t^{tank} can now be added in proportion to it's importance α_{tank} to the reward function R_t^{task} which provides feedback about the main task itself. Thus the multi-objective reward function becomes:

$$R_t = R_t^{task} + \alpha_{tank} \cdot R_t^{tank} \tag{3.13}$$

In Equation 3.12 R_t^{tank} provides an internal motivation to the robot corresponding to "hunger or thirst" in animals. Such reward functions based on internal states fall under the category of intrinsic reward function and are task independent. Other types of intrinsic motivations include exploration and entropy minimization (Singh et al., 2009; Kulkarni et al., 2016).

3.4 Summary

The NEB framework is depicted in Figure 3.2 and Figure 3.3. It operates in a learning mode and in an inference mode. The sequence of operation flow in learning mode can be understood using the pseudo-code shown in algorithm 2. The RL agent uses both external task based reward and internal energy based reward function to optimize it's policy. A similar pseudo code showing the operation flow and causality in inference mode is shown in algorithm 3. It can be seen that the outer loop which feeds energy back in the tank using $P(S_t, S_{t+1})$ is active in both the modes thus the energy tank approach is able to maintain energy level below the safety limit and provide energy awareness in both the modes.

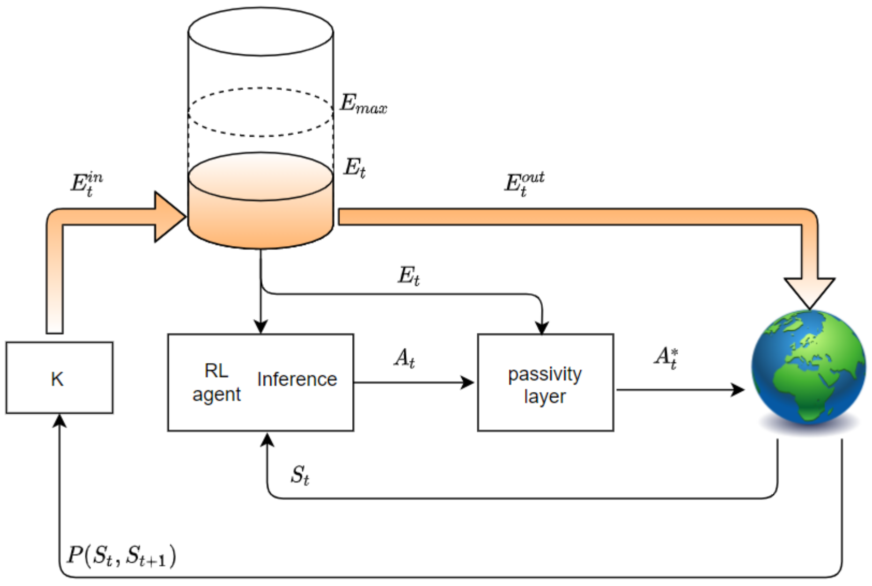


Figure 3.2: NEB framework in Inference mode

3.5 Intelligent Selective Passivity

This section will introduce and prove the property of *Intelligent Selective Passivity* (ISP) which the framework provides to the robot's actuators. For the NEB framework, if energy in the virtual tank E_t is selected as the storage function and the time integral of the supply rate is estimated

Algorithm 2: Learning Mode

```
for trials=1,2,....,M do
    Initialize energy tank with c ($E_0 = c$)
    Sample initial state $S_0$ from the environment
    Initialize a Boolean variable "flag" to "false": $flag = false$
    for t=0,1,2,......N do
        Obtain $A_t$ from the RL agent using $S_t$
        Execute $A_t$ in the environment ($A_t^* = A_t$) then obtain $R_t^{task}$ and $S_{t+1}$
        Calculate $E_t^{out}$ as described in Section 3.2.1
        Update $E_t$ as: $E_t = E_t - E_t^{out}$
        if $E_t > 0$ then
            Calculate $E_t^{in}$ as $E_t^{in} = K \cdot P(S_t, S_{t+1})$
            Clip $E_t^{in}$ between 0 and $E_{max} - E_t$
            Update $E_t$ to $E_{t+1}$ as: $E_{t+1} = E_t + E_t^{in}$
            Obtain $R_t^{tank}$ based on $E_t$ and $E_{t+1}$
            Obtain final reward using weighted sum: $R_t = R_t^{task} + \alpha_{tank} \cdot R_t^{tank}$
            if task is completed successfully then
                Change "flag" to "true": $flag = true$
                Obtain a large positive reward "P": $R_t = R_t + P$
            end
        else
            Change "flag" to "true": $flag = true$
            Obtain a large negative reward "N": $R_t = R_t^{task} - N$
        end
        Append external state $S_{t+1}$ with internal energy states such as: $E_t$, $E_{t+1}$, $E_{t+1}^{in}$,
        $E_{t+1}^{out}$
        Improve the RL agent using an RL algorithm like DDPG
        Update state with the new state: $S_t = S_{t+1}$
        if flag is true then
            Restore "flag" to "false": $flag = false$
            Break the inner loop
        end
    end
end
```

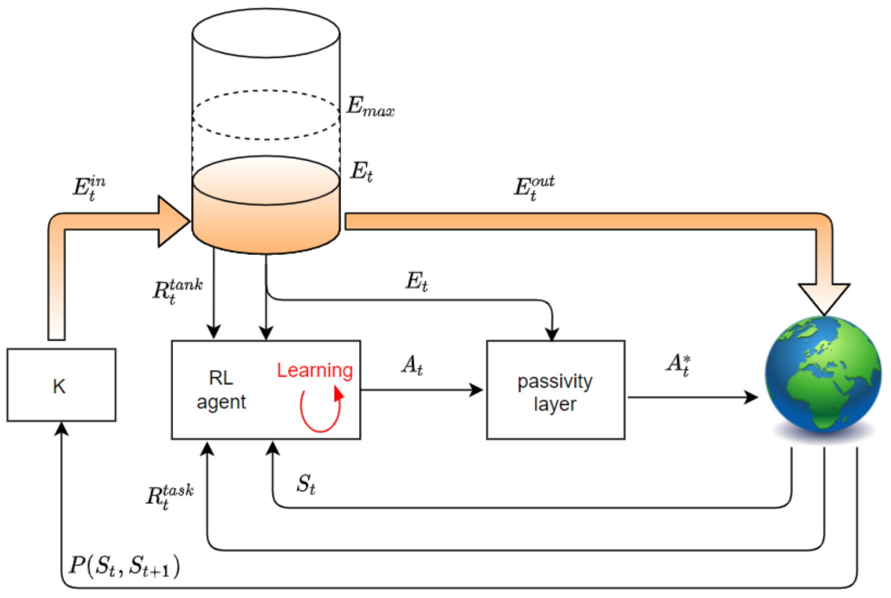


Figure 3.3: NEB framework in Learning mode.

using Equation 3.5, then the passivity condition according to Equation 2.15 becomes:

$$E_{t+1} \leq E_t - E_t^{out} \tag{3.14}$$

here it should be noted that $E_t^{out} \geq 0$. Upon substitution of E_{t+1} from Equation 3.2 we get:

$$\begin{aligned} E_t - E_t^{out} + E_t^{in} &\leq E_t - E_t^{out} \\ E_t^{in} &\leq 0. \end{aligned} \tag{3.15}$$

Thus if $E_t^{in} \leq 0$ the actuators become passive with respect to E_t and $\dot{E}_t^{out}$. On further substitution of E_t^{in} from Equation 3.8 we arrive at:

$$K \cdot P(S_t, S_{t+1}) \leq 0. \tag{3.16}$$

In the above equation $P(S_t, S_{t+1})$ is a potential based reward function defined on a potential function $V(S)$ with it's minima located at the target state S_t^* as described in Equation 3.7.

$$\begin{aligned} &K \cdot (V(S_t) - V(S_{t+1})) \leq 0 \\ &V(S_t) \leq V(S_{t+1}) \end{aligned} \tag{3.17}$$

This implies that if $V(S_t) \leq V(S_{t+1})$ then the robot's actuators behave passively with respect to E_t else they behave in a non-passive way as shown in Figure 3.4. Thus, if the robot tries to move away from the global minima of $V(S_t)$, then the behaviour of it's actuators is completely passive with respect to E_t. On the other hand, if the robot moves towards the target in the potential field then the energy tank receives energy in direct proportion to the decrease in the potential $V(S_t)$ and the passivity is lost. This selective nature of passivity displayed by the robot's actuator is called as *Intelligent Selective Passivity* (ISP).

ISP provides a solution to the problem of loss of autonomy due to sudden decoupling of controller from the actuators when virtual energy tank approach is use, as posed by Raiola et al. (2018). Under the effect of ISP property the robot's controller gets an opportunity to avoid the decoupling by performing well in the task and moving towards the target state. The NEB framework motivates the robot to do so by providing positive rewards when the actuators act non-passively i.e. when E_t increases (Equation 3.12).

Algorithm 3: Inference Mode

```
Randomly initialize a trial
Initialize energy tank with c ($E_0 = c$)
Sample initial state $S_0$ from the environment
for t=0,1,2,......N do
    Obtain $A_t$ from the optimized RL agent using $S_t$
    Execute $A_t$ in the environment ($A_t^* = A_t$) then obtain $S_{t+1}$
    Calculate $E_t^{out}$ as described in Section 3.2.1
    Update $E_t$ as: $E_t = E_t - E_t^{out}$
    if $E_t > 0$ then
        Calculate $E_t^{in}$ as $E_t^{in} = K \cdot P(S_t, S_{t+1})$
        Clip $E_t^{in}$ between 0 and $E_{max} - E_t$
        Update $E_t$ to $E_{t+1}$ as: $E_{t+1} = E_t + E_t^{in}$
        if task is completed successfully then
            break the loop
        end
    else
        Decouple the controller: $A_t^* = 0$
        break the loop
    end
    Append external state $S_{t+1}$ with internal energy states such as: $E_t$, $E_{t+1}$, $E_{t+1}^{in}$, $E_{t+1}^{out}$
    Update state with the new state: $S_t = S_{t+1}$
end
```

It should be noted that even though energy is provided to the robot if it moves towards the target, E_t can increase only if the energy used by the robot during the movement is less than the energy it gains. In other words, the robot should progress with a certain minimum energy efficiency towards the target state in order for E_t to increase. Thus Equation 3.12 not only encourages robot to avoid decoupling by the passivity layer but also encourages energy efficiency. The energy coefficient K in Equation 3.8 plays a centric role in determining this minimum energy efficiency value. The importance of K and other hyper-parameters of the NEB framework will be discussed at length in the following chapters.

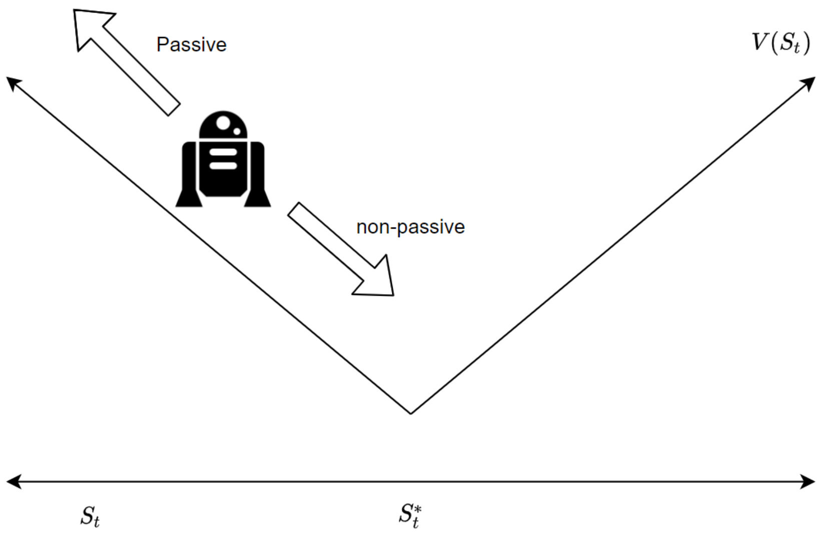


Figure 3.4: The robot's actuator are non-passive when the robot moves towards the global minima of $V(S_t)$ and passive otherwise. The global minima is located at the target state S_t^*

4 Experiment 1 : Proof of Concept

4.1 Introduction

The aim of this experiment is to provide a proof of concept for the *Neural Energy Budgeting* framework. It will be shown that under this framework, the robotic system obeys the safety metric defined not only during the inference mode but also in the learning mode, even when the task energy remains unknown. In the experiment, the objective of a 2-DOF robot is to learn to reach and follow a target in a 2D plane. In each trial, the target is initialized at a random location and then it moves around the robot's base with a random and constant angular velocity. Thus, the required task energy for a particular trial is unknown and unbounded . The robot needs to learn to do the task by respecting the safety norm in learning mode. Finally, in the inference mode, safety must also be maintained in situations where the environment changes for instance if there are obstacles present in the environment, energy released by the robot should not exceed the safety metric.

4.2 Design

This section describes the empirical design of the physical setup of the experiment, the energy tank update laws and the RL framework used.

4.2.1 Physical Design

The setup was adopted from the *reacher-v2* environment of OpenAI Gym (Brockman et al., 2016) with certain modifications. The robot operates in the XY plane with an offset of 0.2 m along the Z-axis. It comprises of 2 capsule-shaped links with radius 10 cm and length 75 cm each as shown in Figure 4.1. The density of each link is 10 kg/m^3. The robot consists of two, 1-DOF joints connecting the base to the first link and the first link to the second link. Both the joint rotate along the Z axis thus restricting the robot in the XY plane. The motors attached to the joints are torque controlled capable of exerting maximum torque of magnitude 1 Nm. Each joint also suffers from a torque loss of 0.5 Nm i.e. motor needs to exert a minimum torque of magnitude 0.5 Nm to observe any movement. The design parameters are summarized in Table 4.1. A simple diagram depicting the top view of the setup can be seen in Figure 5.1 and the rendered view can be seen in Figure 4.3. MuJoCo (Todorov et al., 2012) physics engine is used to render and simulate the task.

Constants	Value [SI units]
Radius of link1	0.1 [m]
Radius of link2	0.1 [m]
Length of link1(L1)	0.75 [m]
Length of link2(L2)	0.75 [m]
Density of link1 & link2	10 [kg/m^3]
Maximum possible torque in motor1 & motor2	1 [Nm]
Torque loss in motor1 & motor2	0.5 [Nm]

Table 4.1: Summary of design parameters for the 2DOF robot

4.2.2 Energy tank update

For each trial the energy tank is initialized to 7J ($E_0 = 7J$). The maximum energy allowed in the tank is limited to 10J ($E_{max} = 10J$). Energy in the tank at time t is denoted by E_t and is

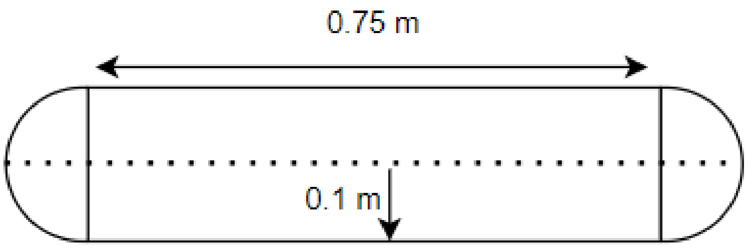


Figure 4.1: Dimensions of a link

recursively updated as:

$$E_{t+1} = E_t - E_t^{out} + E_t^{in} \tag{4.1}$$

Energy going out: E$_t^{out}$

The total energy going out is the sum of energy used by motors 1 and 2. For each motor, the energy going out can be calculated using Equation 3.5. Thus in motoring mode the energy going out corresponding to motor 1, $E_t^{out,1}$ is given by:

$$E_t^{out,1} = \tau_{1,t} \cdot (q_{1,(t+1)} - q_{1,t}) \tag{4.2}$$

where $\tau_{1,t}$ is the torque commanded to motor 1 and $q_{1,t}$ is the joint 1 angle at time t. However, instead of adding energy back to the tank when motor 1 operates in a braking mode energy going out is estimated using $m = 1$ in Equation 3.4 as:

$$E_t^{out,1} = \tau_{1,t}^2 \tag{4.3}$$

energy going out corresponding to motor 2, $E_t^{out,2}$ is calculated in a similar way.

Energy coming in: E$_t^{in}$

Since this is the step where energy is added to the tank, certain conditions must be checked before executing it. Firstly, the energy in the tank before adding any energy should be greater than 0 ($E_t > 0$). Secondly, the maximum energy that can enter the tank at time t should be less than $E_{max} - E_t$ so that E_t never exceeds the safety criterion. Finally, the energy entering should be greater than 0. If all the conditions are satisfied the energy entering the tank is given by:

$$E_t^{in} = K \cdot (D_t - D_{t+1}) \tag{4.4}$$

Where the value of K is 15 J/m and D_t is the euclidean distance in X and Y coordinate between the target and the end-effector at t. Which means 15J of energy gets added in the tank if D_t decreases by 1 m over the next time step. Greater is the decrement in D_t, greater is the amount of energy added. In this setup $D_t - D_{t+1}$ serves as an example of potential-based reward function $P(S_t, S_{t+1})$ used in Equation 3.8. The decision process is summarized in Figure 4.4.

4.2.3 RL framework

The RL agent which learns the task, is implemented by following the notations and methodology described in Section 2.1. Firstly, the state S_t defines a vector containing all the information

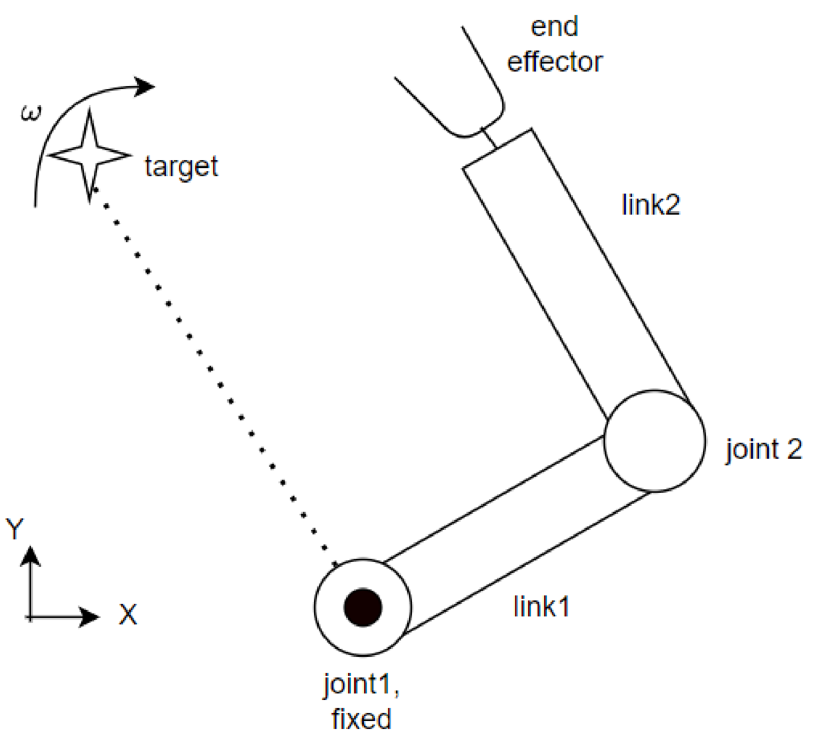


Figure 4.2: Top view of the system. The target revolves with a random ω around the robot's base joint in the XY plane

at time t necessary for the RL agent to decide the next action. For this experiment S_t is defined as:

$$S_t = \begin{bmatrix} cos(q_{1,t}) \\ cos(q_{2,t}) \\ sin(q_{1,t}) \\ sin(q_{2,t}) \\ \dot{q}_{1,t} \\ \dot{q}_{2,t} \\ X_t^{tar} \\ Y_t^{tar} \\ X_t^{EE} \\ Y_t^{EE} \\ E_t \\ E_t^{out,1} \\ E_t^{out,2} \\ E_t^{in} \end{bmatrix} \tag{4.5}$$

where, $q_{1,t}$ and $q_{2,t}$ are the joint angles of joints 1 and 2 measured in radian at time t. $\dot{q}_{1,t}$ and $\dot{q}_{2,t}$ are the angular velocities of joint 1 and 2 at t. X_t^{tar} and Y_t^{tar} denote the X and Y coordinates of the target with origin at the robot's base and they keep changing according to a random angular velocity ω. X_t^{EE} and Y_t^{EE} denote the X and Y coordinates of the robot's end-effector in the same coordinate system at time t. The level of energy in the tank at t is given by E_t. The energy leaving the tank at t through motors 1 and 2 are given by $E_t^{out,1}$ and $E_t^{out,2}$ respectively. Finally E_t^{in} denotes the energy dynamically added to the tank at time t. E_t^{out} and

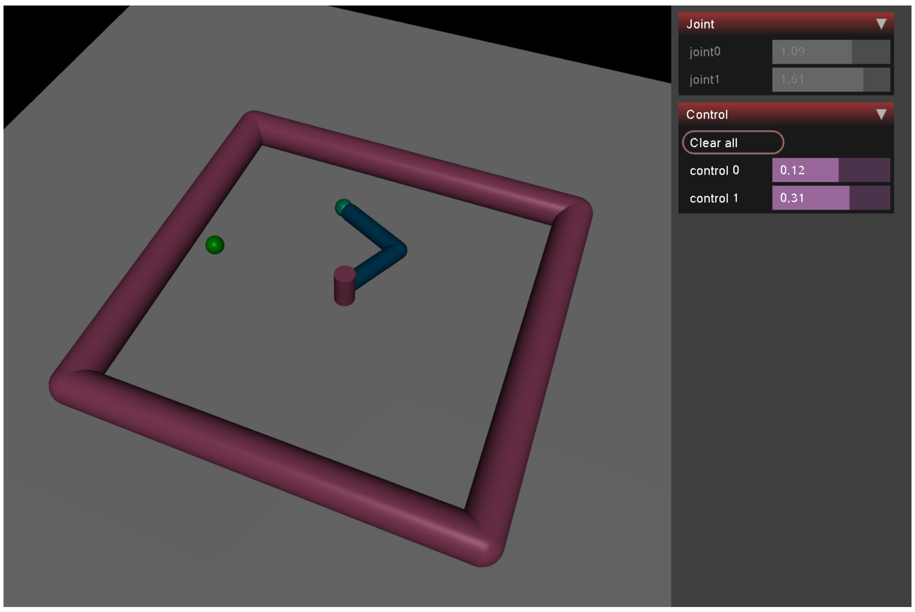


Figure 4.3: In the MuJoCo rendered environment target is depicted by the green sphere

E_t^{in} are estimated as described in Section 4.2.2. It should be noted that the ω itself is not a part of the state vector

The reward function defines how well does the agent perform in the given task at t. For this particular task the reward function comprises of 3 sub reward functions. Firstly, $R_t^{task} = -0.5D_t$ penalizes the distance D_t between end-effector and target at time t. Secondly, $R_t^{energy} = E_t^{in} - E_t^{out}$ appreciates energy coming in and penalizes energy going out at t. Lastly, $R_t^{tank} = -5(1 - \frac{E_t}{E_{max}})$ penalizes the agent if the energy in tank at t is less than E_{max}. R_t^{tank} and R_t^{energy} encourage the agent to avoid the tank singularity condition, $E_t \leq 0$. Thus, the final reward value is calculated as:

$$\begin{aligned} R_t &= R_t^{task} + R_t^{energy} + R_t^{tank} \\ R_t &= -0.5D_t + E_t^{in} - E_t^{out} - 5(1 - \frac{E_t}{E_{max}}) \end{aligned} \tag{4.6}$$

Additionally, a positive reward of 20 is added to R_t whenever $D_t < 0.05m$ i.e. end-effector is close enough to the target. If the energy in the tank runs out a negative reward of -500 is deducted from R_t and the trial is terminated. The action vector A_t produced by the Actor directly outputs the torque value between -1 Nm and 1 Nm for both the motors.

$$A_t = \begin{bmatrix} \tau_{1,t} \\ \tau_{2,t} \end{bmatrix} \tag{4.7}$$

Where, τ_{nt} is the torque output of n^{th} DOF ranging between -1 Nm and 1 Nm.

The Actor is modeled using deep neural network which maps S_t to A_t. Actor consists of 3 fully connected hidden layers containing 512, 256 and 128 neurons starting from S_t, progressing towards A_t. All neurons of the hidden layers of the Actor are activated by a Rectified Linear Unit (ReLU) function. In a ReLU function the output is equal to input, if the input is greater than 0 else, the output is equal to 0. The output layer is activated using a $tanh$ function. The Critic is represented using a 2 layer, fully connected, deep neural network which maps the state, action pair (S_t, A_t) to the value of the pair $q(S_t, A_t)$ at t. The Critic consists of 400 and 300 ReLU activated neurons in the hidden layers. The output of the Critic is activated using a linear transformation where output is equal to the input scaled by a constant weight. The topology of

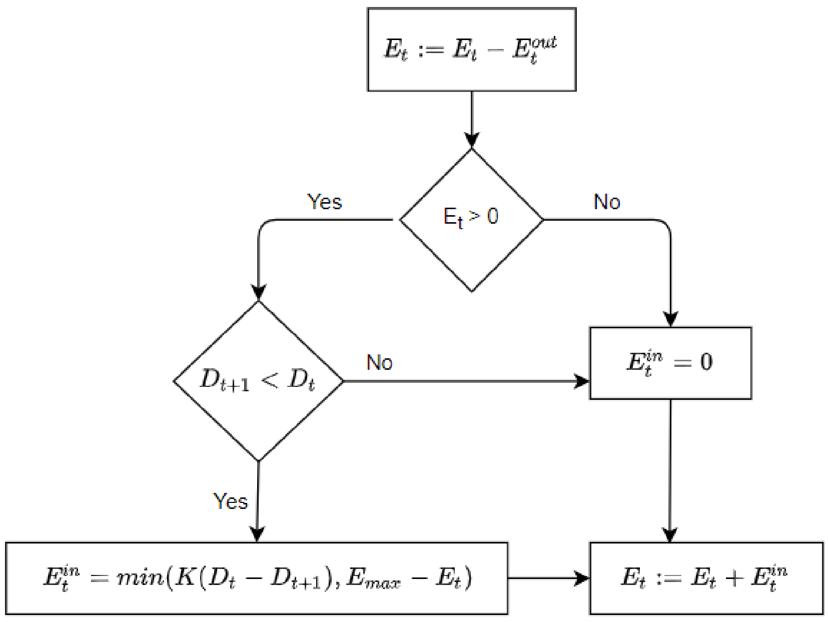


Figure 4.4: Decision flow to decide E_t^{in}

the networks can be summarized in Figure 4.5. The RL agent learns using a DDPG algorithm summarized in Section 2.1.3, with hyper-parameters listed in Table 4.2. The neural networks are modeled using the Pytorch library (Paszke et al., 2019) developed in Python. They were trained using an Adam Optimizer (Kingma and Ba, 2014) on an NVIDIA Quadro P1000 GPU.

Constants	Value
learning rate actor	0.001
learning rate critic	0.001
discount factor γ	0.99
soft update rate τ	0.001
batch size	256

Table 4.2: Hyper-parameters for learning algorithm

4.3 Results

The results of the experiment are analyzed under two modes of operation of the framework; learning mode and inference mode. In the learning mode the agent is still in the process of optimizing the controller while in inference mode the controller has achieved a sufficient level of proficiency in doing the task.

4.3.1 Learning Mode

In the learning mode the controller is not yet optimal and gradually learns to do the task. The training occurs over 5000 trials with each trial having maximum length of 200 steps. In the beginning of each trial the target is initialized to a random location and as soon as the trial begins it starts revolving around the base of the robot with a random angular velocity between -1 and 1 rad/s. Figure 4.6 shows the graph of total rewards achieved over the trials. Since the target moves with different speeds in different trials the total task energy keeps varying. It can be seen in Figure 4.7 that the energy in the tank never exceeds the safety metric of 10J even though the task energy varies and is often much more than 10J.

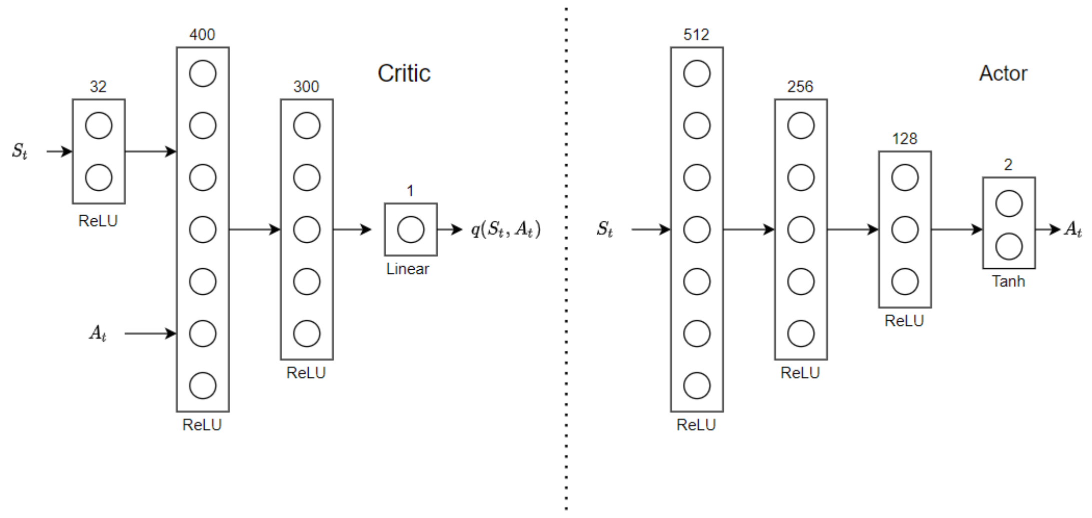


Figure 4.5: Each layer is represented using vertical rectangle. The number of neuron in a layer is mentioned above the rectangle and the activation function used is mentioned below the rectangle. Arrow represents dense connections

4.3.2 Inference mode

In the inference mode the agent has achieved a required level of proficiency in the task. The learned agent is observed in different scenarios i.e. for different target positions and angular velocities. The robot reaches the target and tracks it with proficiency. Figure 4.8 shows the total energy used by the robot and the energy in the tank at 4 different instances in a trial, in inference mode. It can be seen that while tracking the target the energy in the tank never exceeds the safety limit of 10J.

In order to test the ISP property of the actuators, fixed obstacles are introduced in the arena at random positions as shown in Figure 4.9. As expected, whenever the robot collides with an obstacle and tries to exert energy to move them, its actuators enter a passive mode and energy drains out of the tank quicker since no energy is being added to the energy tank.

4.4 Discussion

In the experiment two critical parameters that were empirically set were K and E_{max} in Section 4.2.2. E_{max} is the maximum energy allowed in the tank. For a given task it depends on the safety requirements in terms of maximum energy allowed per unit time which in-turn depend on the task at hand and which body part of humans working with the robot have maximum probability of collision. This has been discussed in Section 2.5. Intuitively, lower the value of E_{max}, safer the system becomes as the energy-tank will run out of energy sooner in case of a collision. However, lower E_{max} value makes learning difficult for the agent as it restricts the state-action space which the agent can explore.

In the context of the NEB framework, energy coefficient K in Equation 4.4 is a very important parameter as it decides the amount of energy entering the tank per unit distance robot moves towards the target. K forces the RL-agent to answer the question of affordability of an action in terms of energy i.e. is it worth spending certain amount of energy on the action if a certain amount of energy is gained in doing so? The amount of energy gained is scaled by K. This aspect is highlighted in the next experiment. Thus, K defines the energy economy in which the robot operates. Higher values of K will allow the robot to implement trajectories using more power on average. This effect of value of K is more visible in learning mode when the chance of taking random actions is higher than inference mode. In the inference mode when the agent has learned an energy efficient way to reach the target this effect of K is less visible.

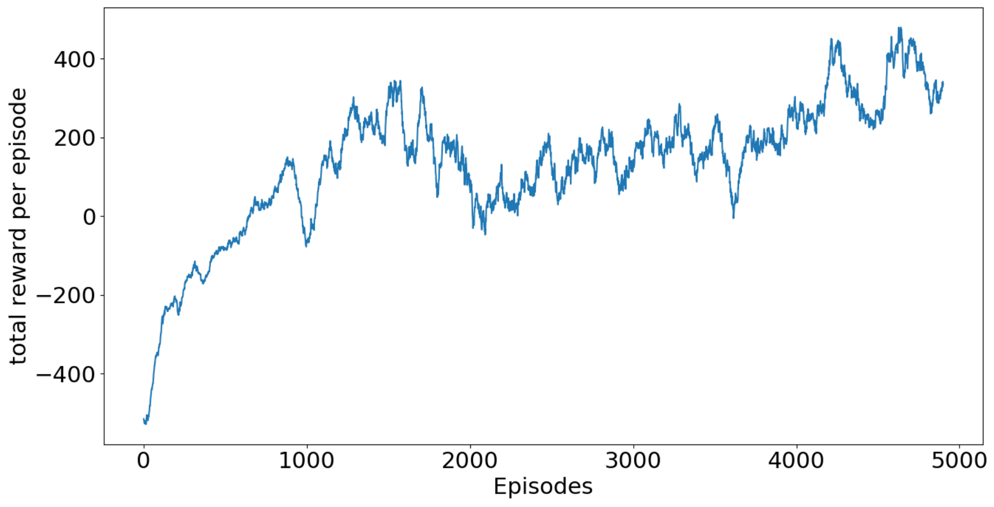


Figure 4.6: Result of training

In Figure 4.10 it can be seen that when K = 1000J/m the average power used by the robot actuators per trial while learning is high (around 4 W) in the beginning when agent explores the environment with more randomness, while when K = 8 J/m this value decreases significantly. Average power is calculated by summing all the energy used by the robot per trial and dividing the sum by total length of the trial. The targets are kept stationary in-order to gauge the effect of varying *K* on average power, so that a comparison among different trial is possible. It must also be added that too low value of K makes the system more conservative with respect to energy usage, which hinders adequate exploration of the state space and thus leads to poor learning.

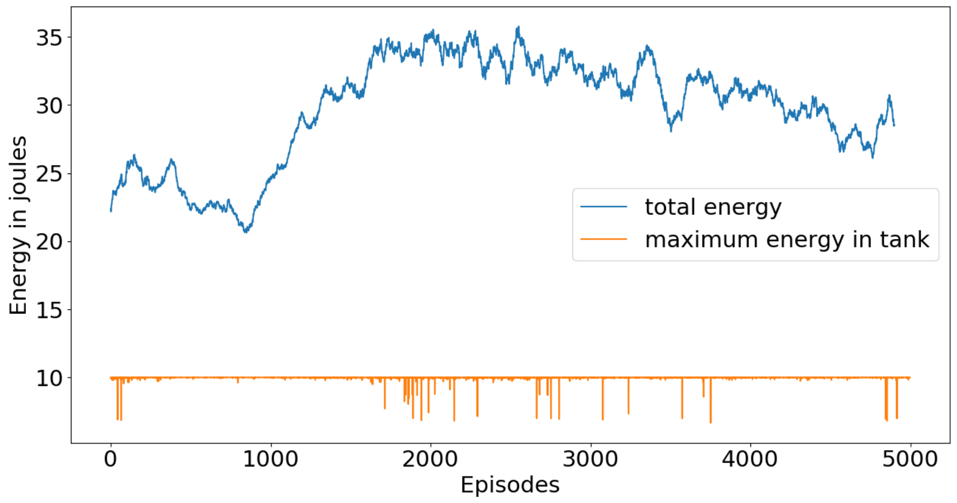


Figure 4.7: Comparison of energy used in each trial and maximum energy in the tank for the trial, while learning

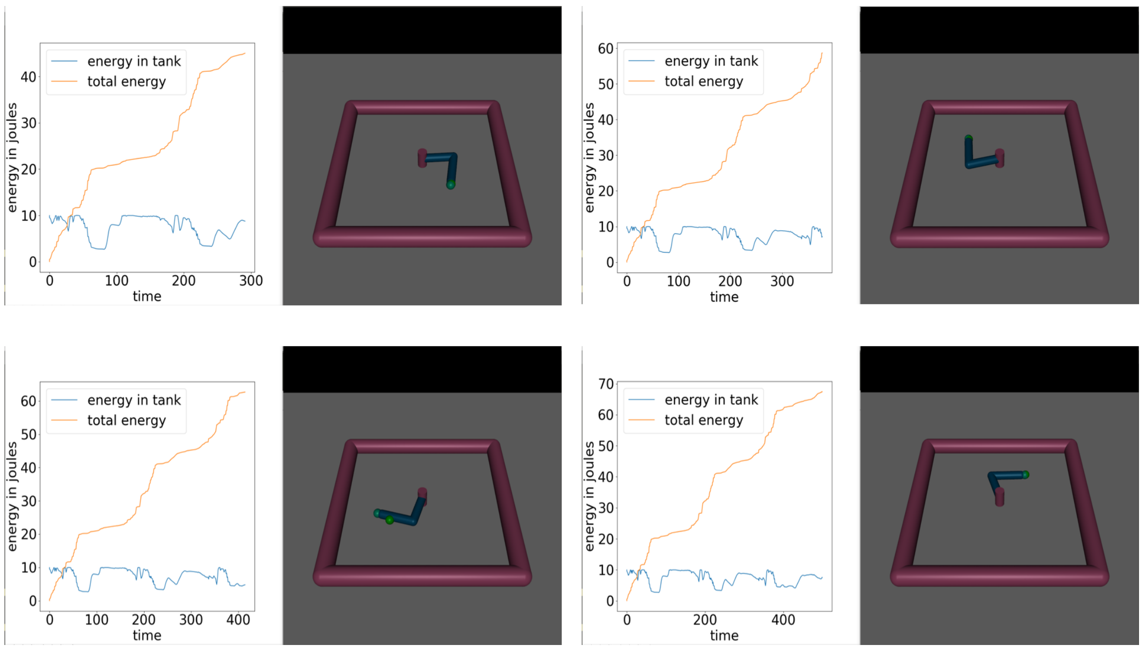


Figure 4.8: Rendered view of robot tracking the target at 4 different instances in a trial. On left of every rendered view is a plot representing the total energy used by the actuators till that instance and the energy in the tank at that instance

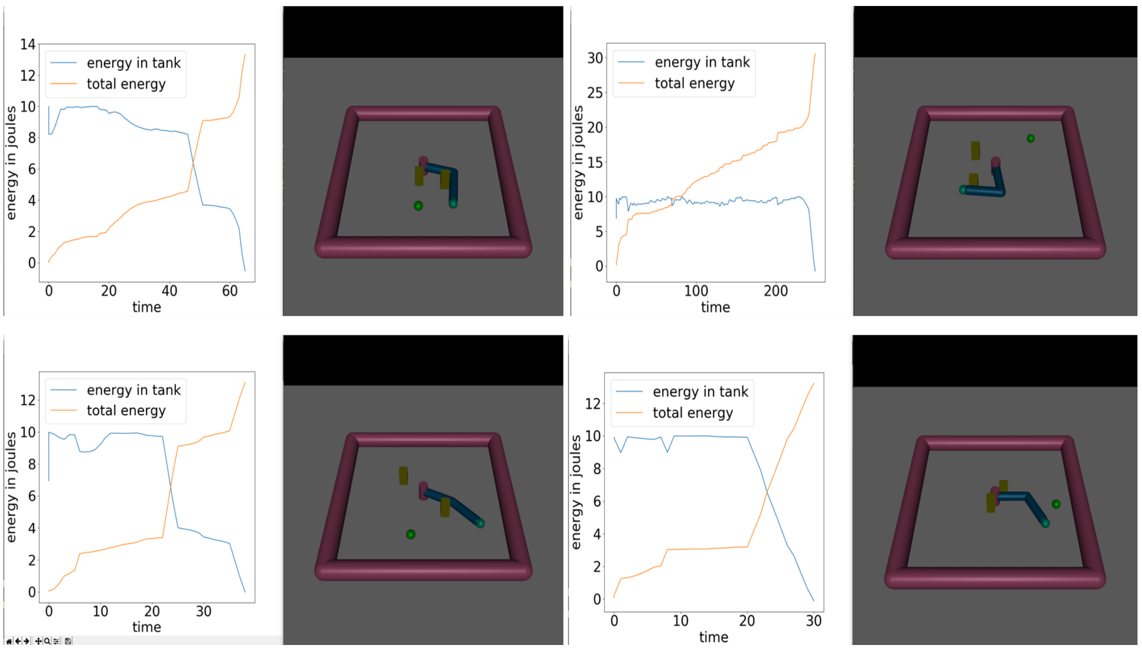


Figure 4.9: For different trials obstacles are randomly placed in the arena. The sudden dip of the energy level of the tank shows robot tries to exert force against the obstacle. As soon as the energy level reaches 0 the controller is decoupled from the robot

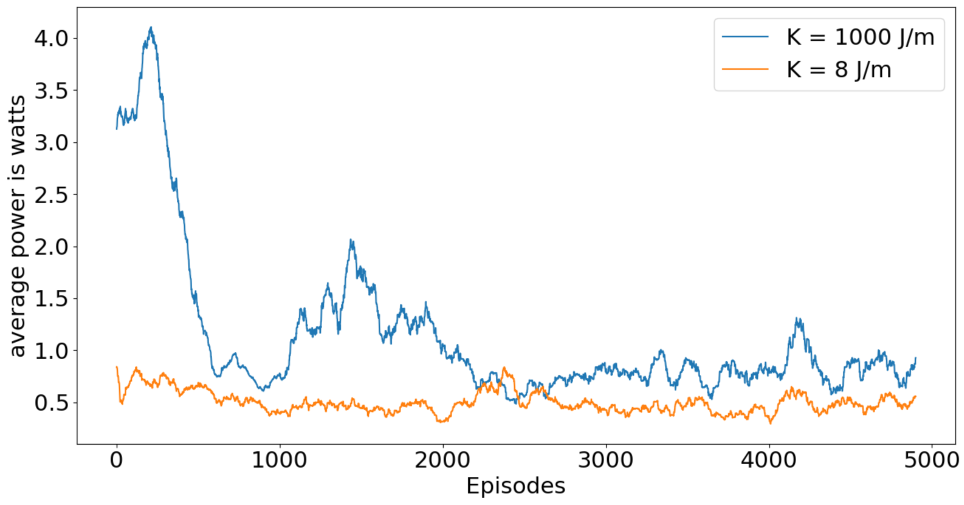


Figure 4.10: In order to compare the average power for different K, the targets are initialized randomly while learning, but are kept stationary after that. Average power is then calculated by total energy used for the trial divided by the duration of the trial

5 Experiment 2 : Energy Awareness

5.1 Introduction

This experiment highlights the benifit of including the internal energy states based on the virtual energy tank in an RL framework. Sensing the energy profile in the tank and defining a reward function based on it can lead to certain desirable energy aware behaviour. More specifically, in this experiment the aim of a 1-DOF rotating robot is to reach the target placed at other end of the arena. It can do so by either rotation clockwise or anticlockwise. Both the paths leading to the target may be obstructed by movable obstacles with different masses and the robot is not equipped with any type of force or pressure sensor. The controller of the robot which is represented by an RL agent needs to decide whether it is affordable to displace an obstacle with a certain mass in-order to reach the target, based on the change of the energy level in energy tank. If it is affordable then the controller can continue towards the target by moving the obstacle, else the controller needs to change its course of action and try reaching the target from the other side. Thus, the controller needs to sense the level of energy in the energy tank in order to sense the presence of the obstacle, estimate its mass and take actions accordingly.

5.2 Design

The design of this experiment is very similar to Chapter 4 thus mainly the changes will be highlighted in the section.

5.2.1 Physical design

The environment was empirically designed using the XML file format required for MuJoCo rendering. The robot is a single DOF robot operating in the XY plane with an offset of 0.2 m along the Z-axis. It is composed of a single capsule link with dimensions and density similar to Experiment 1 as shown in **??**fig:capsule). The link rotates around a single DOF joint located on one of its end around the Z axis. The DOF is powered by a torque controlled motor capable of exerting maximum torque of magnitude 3 Nm. The joint suffers from a torque loss of 0.5 Nm. The coefficient of friction between the floor and the obstacles is 0.1. The origin of the system lies at the base of the robot. In the beginning of each trial the target is located at 3.14 rad from the X axis, at a distance of 0.75m from the origin with an offset of $Z = 0.2$ m from the XY plane. There are two obstacles modeled by movable boxes which are $0.1m \times 0.1m$ at the base and $0.2m$ in height. Both obstacles are initialized by random masses in the beginning of the trial and are located at an angle of $\pm\pi/4$ with respect to X axis. The design parameters are summarized in Table 5.1. The top view of the experiment is shown in Figure 5.1 a rendered view can be seen in Figure 5.2.

Constants	Value
Radius of link	0.1 [m]
Length of link	0.75 [m]
Density of link	10 [kg/m^3]
Maximum possible torque in motor	3 [Nm]
Torque loss in motor	0.5 [Nm]
Coefficient of friction	0.1
base of obstacle	0.1 x 0.1 [m^2]
height of obstacle	0.2 [m]

Table 5.1: Summary of design parameters for the robot

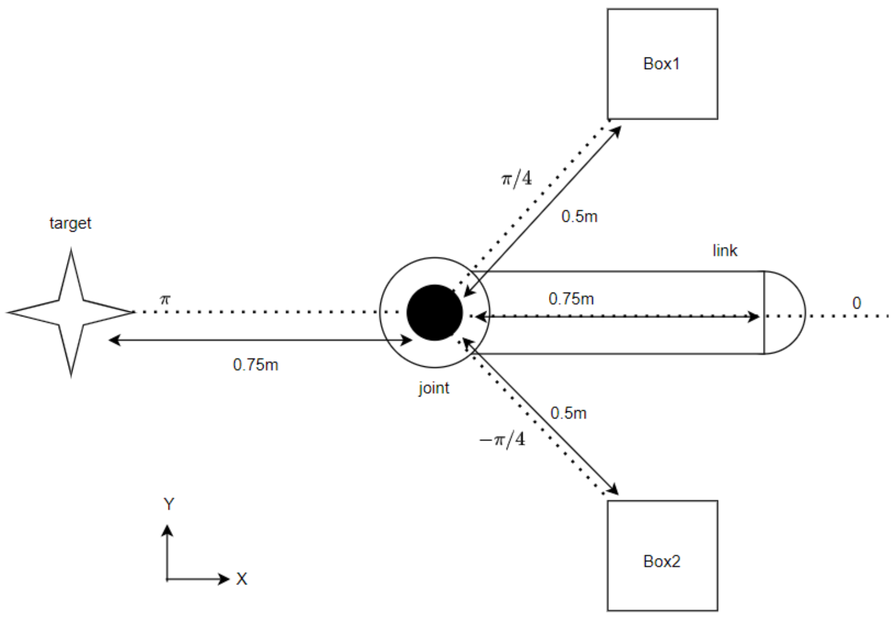


Figure 5.1: Top-view of the setup

5.2.2 Energy tank update

The energy tank is updated in a similar way as shown in Chapter 4. In beginning of each trial the tank is initialized with $5J$ of energy. The maximum limit E_{max} is set to $8J$. Since, in this setup the robot contains just 1 actuator, the energy leaving the tank corresponding to the actuator is equal to the total energy leaving the tank. The energy going in the tank is given by

$$E_t^{in} = K * (D_t - D_{t-1}) \tag{5.1}$$

where the value of K is set to 1.5 J. Here D_t does not return the euclidean distance between the target and the end-effector but the absolute difference between the cosine of current angle of the robot's link and target angle of π rad.

$$D_t = |cos(\pi) - cos(q_t)| \tag{5.2}$$

5.2.3 RL framework

S_t for this set up is designed as follows:

$$S_t = \begin{bmatrix} cos(q_t) \\ sin(q_t) \\ cos(q_{t-1}) \\ sin(q_{t-1}) \\ \dot{q}_t \\ \dot{q}_{t-1} \\ cos(\pi) \\ E_t/E_{max} \\ E_{t-1}/E_{max} \\ E_t^{in} \\ E_{t-1}^{in} \\ E_t^{out} \\ E_{t-1}^{out} \\ A_{t-1} \end{bmatrix} \tag{5.3}$$

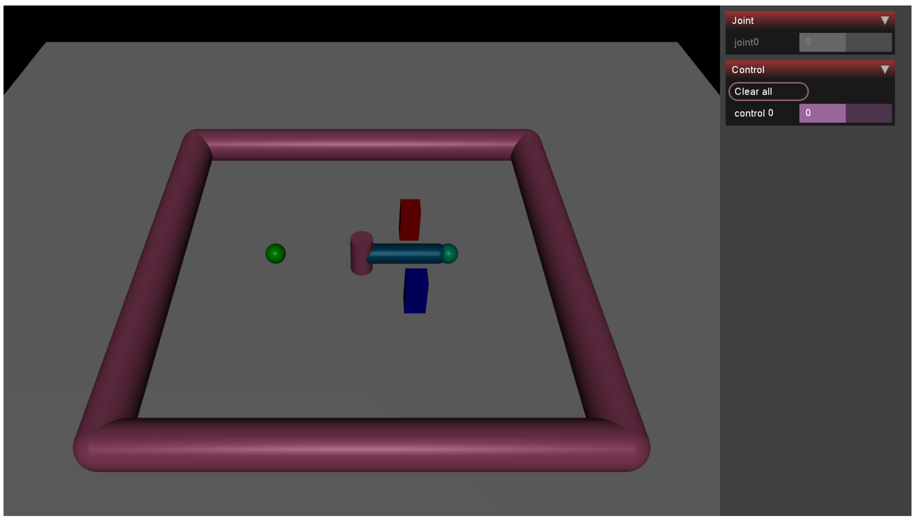


Figure 5.2: Rendered environment in MuJoCo

where q_t is the robot's joint angle and $\dot{q}_t$ is the joint velocity at time t. A constant state of $cos(\pi)$ is present in the vector to specify the target's position. The tank energy is represented by E_t and is scaled by E_{max} before inclusion in the state vector. E_t^{out} and E_t^{in} represent energy leaving and entering the energy tank respectively. Finally, it should be noted that the recent past values of every state i.e. at $t-1$ is also included in the vector. The suggestion of including the history of states was adapted from Hwangbo et al. (2019) as in their work it is hypothesised that inclusion of the history of states will enable contact detection.

The reward function is based on the same structure and reasoning as discussed in experiment 1 in Equation 4.6. It is formulated as:

$$R_t = E_t^{in} - E_t^{out} - 0.2D_t - 2(1 - \frac{E_t}{E_{max}}) \tag{5.4}$$

An additional reward of 500 is given if the end-effector reaches the target ($D_t < 0.01$). A negative reward of -500 is provided if the energy in the tank runs out and the trial is restarted. The action output A_t from the actor is just a single value representing the torque of the motor.

$$A_t = \tau_t \tag{5.5}$$

The topology of Actor and Critic is depicted in Figure 5.3. A DDPG algorithm was used to train the agent with hyper-parameters similar to the ones used in Chapter 4 mentioned in Table 4.2 with the difference of batch size which was 512 for this experiment. The software structure and hardware used for training is again similar to the one used in Chapter 4.

5.3 Results

The RL agent was trained for 10000 episodes where maximum length of each episode was 200 steps. The result of training is visible in Figure 5.4. The experiment was designed to check if the RL agent can use the energy in the tank and answer the question of affordability of an action. More specifically an action is considered affordable if the Rl agent decides that it receives enough energy from the environment, in return of energy spent on doing the action and hence it continues doing it.

In the setup, the energy spent on an action is directly proportional to the torque applied by the motor while undertaking it. When the robot collides with an object placed on the floor, the force needed to move the object depends on the its mass (if friction between the object and the floor is present). More the mass of the object more is the force needed to move it and more

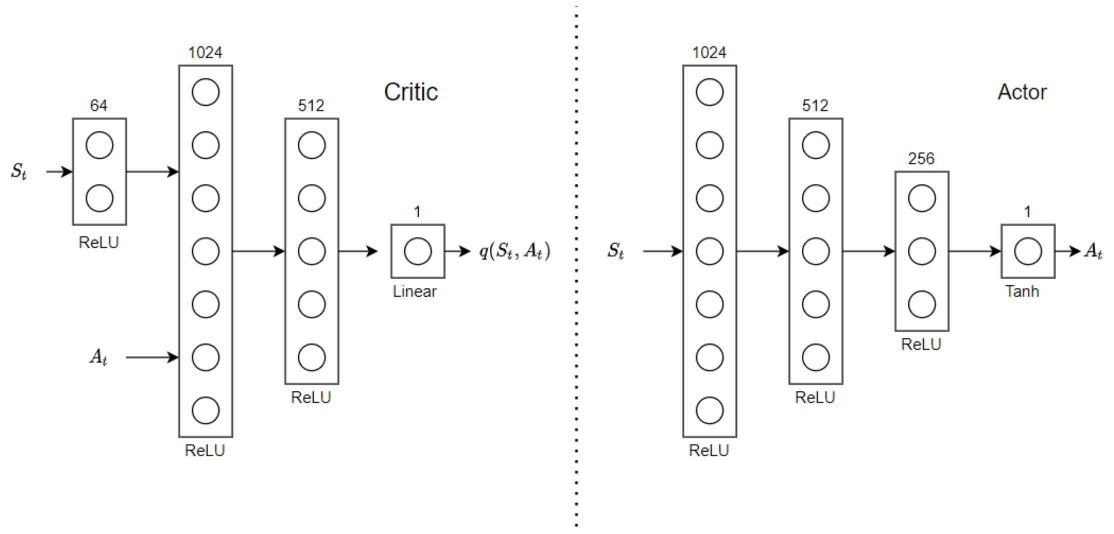


Figure 5.3: Topology of Actor and Critic. The notation is similar to Figure 4.5

will be the energy leaving the tank. Thus in this scenario, the question of affordability can be re-framed in terms of the maximum mass of obstacle which, the robot has decided to displace in-order to reach the target position.

In the initial position, the robot is capable of displacing a maximum mass of $3Kg$ when the motor exerts the maximum torque of $3Nm$. However, after training is complete it is observed that when the robot encounters a mass greater than $2.5Kg$, instead of displacing it the robot tries to reach the target using the other side. Thus $2.5Kg$ becomes a soft-limit or affordable limit. Any mass less than $2.5Kg$ is deemed affordable by the robot and the robot decides to displace it while moving towards the target. Such a affordable limit can be changed by changing the value of K. Table 5.2 shows the soft-limit observed when value of K is varied. It can be seen that as the value of K is gradually increased, the affordable limit also gradually increases [1]. This occurs because robot receives more energy for higher values of K for moving closer to the target by the same distance. Thus, this also shows how varying the value of K can change the behaviour of the robot.

K[J]	Soft Limit[Kg]
1.5	2.5
2.0	2.6
2.5	3.0

Table 5.2: Correlation between K and soft-limit

[1]video available here

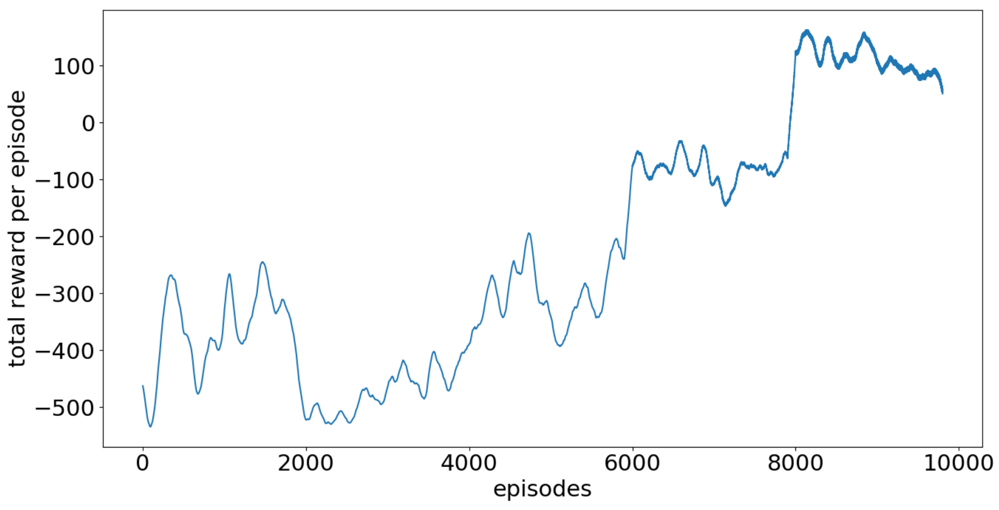


Figure 5.4: result of training the agent

6 Experiment 3 : Early Stopping while Learning

6.1 Introduction

State of the art RL algorithms often use an *Expirience Replay* (Schaul et al., 2015) technique in order to tune the RL agent. Under this technique the current state (S_t), current action (A_t), current reward value (R_t) and the next state S_{t+1} encountered by an agent in all trials are stored as a data point $e_t = (S_t, A_t, R_t, S_{t+1})$ in a buffer memory $D = (e_1, e_2, e_3....e_t)$. During the learning step, samples are drawn at random from D_t in-order to calculate the error functions for the Actor and the Critic. This technique enables multiple use of one sample for learning it also improves data efficiency and removes correlations in the observation sequence.

In RL paradigm *Infinite horizon* task problems refer to learning a task which does not has a specific end time or end-position. For instance training a bipedal robot to walk in straight line. When dealing with such tasks, the length of learning trials is set to be long enough to observe that the desired infinite task behaviour is performed with proficiency. While learning, all the data points from a learning trial get stored in D_t. Thus, if a learning trial itself is large, then it will consume more memory space when stored. For instance, a simple humanoid model such as the OpenAI Gym (Brockman et al., 2016) "Humanoid-v2" can consume upto 3 GB of memory (Imre, 2021). If the memory space available on the computer being used to train the RL agent is limited then, it may not be possible to store all the learning trials in the memory. This may further decrease the learning efficiency.

The proposed framework when applied to an infinite horizon task helps reduce the memory consumed while learning. This occurs because a learning trial is terminated earlier than the usual set length if the energy in the tank runs out, thus resulting in less memory usage. This experiment provides an example of the utility of this property during the learning mode. In this experiment the task of a "cheetah" shaped robot is to learn to run towards positive X direction with a given energy efficiency. This task qualifies as an infinite horizon problem

6.2 Design

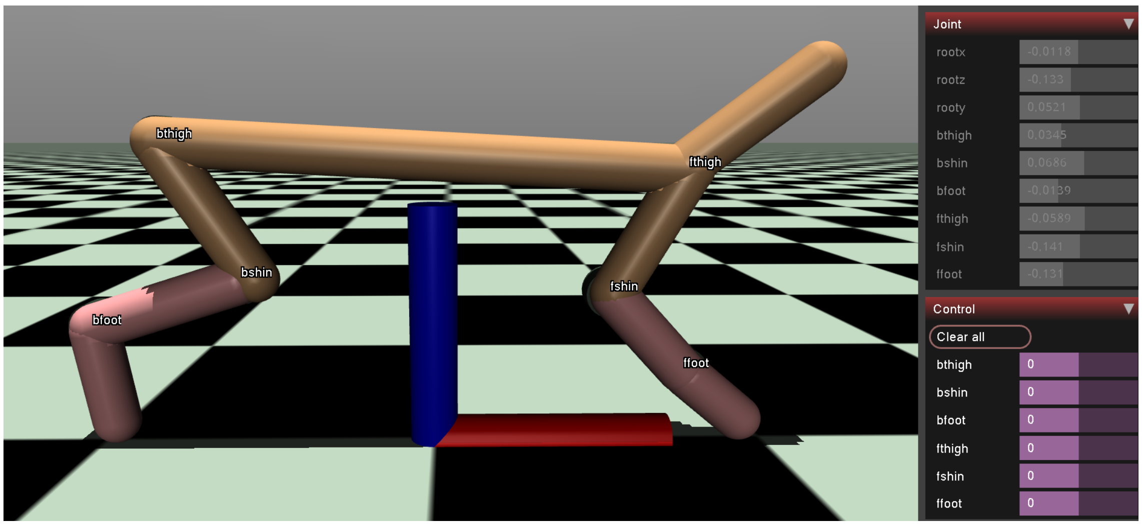
The setup used for this experiment is provided by the OpenAI Gym *HalfCheetah-v2* environment.

6.2.1 Physical Design

The robot used in the simulation is a 2-legged robot resembling a fore-leg and a hind-leg of a cheetah. Its COM is restricted in the XZ plane. The robot has 6 DOF in the form of hinge joints, corresponding to the thigh, shin and foot in each leg. They are actuated by torque controlled motors. The absolute torque limits of the motors can be seen in Table 6.1 and a MuJoCo rendered view of the setup can be seen in Figure 6.1. For each trial the robot is initialized at the origin of the environment.

Motor	Absolute torque limit [Nm]
bthigh (back thigh)	120
bshin (back shin)	90
bfoot (back foot)	60
fthigh (forward thigh)	120
fshin (forward shin)	60
ffoot (forward foot)	30

Table 6.1: absolute torque limits for robot's motors



rootx	-0.0118
rootz	-0.133
rooty	0.0521
bthigh	0.0345
bshin	0.0686
bfoot	-0.0139
fthigh	-0.0589
fshin	-0.141
ffoot	-0.131

bthigh	0
bshin	0
bfoot	0
fthigh	0
fshin	0
ffoot	0

Figure 6.1: Rendered environment in MuJoCo

6.2.2 Energy tank update

The energy tank architecture is defined in a similar way to previous 2 experiments. For each trial, the energy tank is initialized with 40J of energy ($E_0 = 40$). Since, safety is not considered for this task there is no limit on the maximum amount of energy that can be stored in the tank. The energy leaving the tank is calculated by summing the energy consumed by each motor which is estimated using Equation 3.3 when the motors operate in motoring mode. In the braking mode Equation 3.4 is used and the value of m for each motor is set as square of its torque-limit divided by 10^4.

Energy is supplied back to the tank using the following equation:

$$E_t^{in} = K \cdot (X_{t+1} - X_t) \tag{6.1}$$

where X_t is the X coordinate of the COM of the robot at time t. K is the energy coefficient which is set as 30 J/m. Thus the energy tank receives 30 J of energy if the robot moves 1 m in the positive X direction. If the energy in the tank runs out then the current trial is terminated and a new trial is started.

6.2.3 RL framework

S_t for this experiment is designed as:

$$S_t = \begin{bmatrix} q_t^{COM} \\ q_t^{joint} \\ \dot{q}_t^{COM} \\ \dot{q}_t^{joint} \\ E_t/E_0 \\ E_{t-1}/E_0 \\ E_t^{in} \\ E_t^{out,v} \end{bmatrix} \tag{6.2}$$

where q_t^{COM} itself is a vector containing the coordinates of the COM in the XZ plane and its rotation around the Y axis. Elements of vector $\dot{q}_t^{COM}$ contain time derivatives of the corresponding elements in q_t^{COM}. Similarly, q_t^{joint} represents a vector containing the angular displacements sensed by the displacement sensors attached to the 6 hinge-joints of the robot and the vector $\dot{q}_t^{joint}$ contains the angular velocities of the same joints. Energy in the tank E_t is scaled by the in initial energy E_0 before being added to the state vector. E_t^{in} represents the energy given to the tank at t. Finally, $E_t^{out,v}$ is a vector of the energy used by all the 6 motors at t used to calculate the energy leaving the tank.

The reward function for the task is formulated as:

$$R_t = E_t^{in} - E_t^{out} - 30 \cdot (1 - \frac{E_t}{E_0}) \tag{6.3}$$

This is again similar to Equation 4.6 except here, if E_t is less than the starting energy E_0 then the system is penalized and if E_t exceeds E_0 then the RL agent receives positive reward. One more thing to note in Equation 6.3 is that unlike the previous two experiments the task reward is not present explicitly but it is implicitly present in $E_t^{in} - E^{out}$. The RL agent also receives a surviving reward of 0.5 at every time step. If tank singularity is encountered then the trial is terminated and the RL agent receives a reward of -35. The action vector A_t represents the torque command issued by the RL agent to the 6 motors actuating the robot.

The deep-neural networks used to approximate the Actor and the Critic are exactly the same as used for the first experiment and shown in Figure 4.5, except that the Actor has 6 outputs corresponding to A_t. A DDPG algorithm is used to train the RL agent with hyper-parameters similar to the first experiment as presented in Table 4.2 on the same hardware.

6.3 Results

The RL agent is trained for $200,000$ steps and the memory storage capacity of the buffer is limited to $20,000$ data points while training. The buffer cannot store all the data that is generated while Rl agent is training hence the data which gets stored in the buffer also leaves it using the principle of *First in First out* buffer system. The RL agent is trained in two settings. In the first setting early stopping is enable by virtue of the energy tank i.e. when the energy in the tank runs out the trial is terminated and a new trial is started. In the other setting no energy tank or early stopping is present and the agent is trained in a conventional way but, with limited buffer memory. The comparison of the two settings is visible in Figure 6.2. It is to be noted that the comparison is presented with respect to number of training steps and not number of trials because the length of a trial can be different for both the settings.

In Figure 6.2 it can be observed that when the early stopping of trial is enabled the agent learns better than when it is not. This can be attributed to 2 reasons. Firstly, the trials in which energy runs out early are terminated early, thus there is more space available for the trials in which energy does not runs out to be stored in the buffer. Secondly, the data points that get stored represents the step in which, energy in the tank is greater than 0. These data points indirectly correspond to instances in which the agent's performance is relatively better as the energy in the tank remains positive only if the robot continues to move forward. This leads to the learning algorithm encountering more "successful" state-action pair.

6.4 Discussion

There are multiple ways in which early stopping can be achieved in an infinite horizon task. The technique presented here demonstrated a model free way of achieving early stopping. In summary, the trial is stopped when the energy in the virtual tank associated with the actuators of the robot runs out. The tank looses energy when the robot undertakes any action but gains energy only when the robot progresses towards the goal. One important thing to note is that not only should the robot progress towards the target in-order to survive but also must do so by loosing lesser energy than gaining. This is only possible by learning to do the task with a certain minimum energy efficiency.

In the experiment, the energy efficiency was determined by the energy coefficient K. Since K determines the amount of energy that the tank receives when the robot moves 1 m in positive

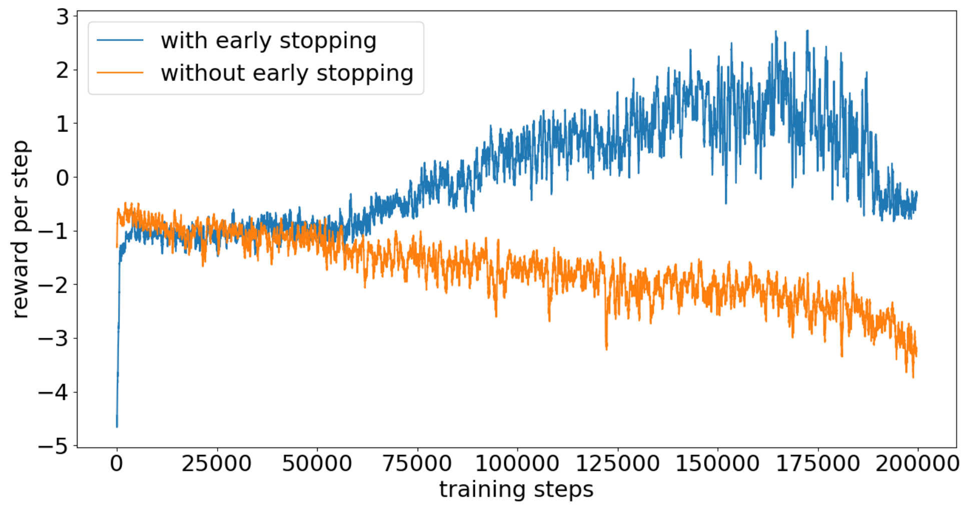


Figure 6.2: Results of training

X direction. If the robot spends more amount of energy than K J of energy in moving 1 m then eventually the energy in the tank will run out and the RL agent receives a negative reward and the trial will be terminated. Thus, in order to survive the robot has to move with at least the efficiency of K J/m. Finally, It should be noted that the initial amount of energy in the tank E_0 can be changed in the beginning of each trial to change the length of each trial. Very large value of E_0 can prevent early stopping as the energy in the tank may never run out. On the other hand a small E_0 can hinder exploration and can negatively affect the learning efficiency.

7 Discussion

7.1 Introduction

In this section it is shown how the presented framework can become an example of the *homeostatic reinforcement learning* framework (Keramati and Gutkin, 2014) applied to robotics by simple modifications to the reward function used for learning purpose. Further some limitations and future work will be discussed.

7.2 Homeostatic Reinforcement Learning

Homeostasis refers to the process of self-regulating the internal states by an organism such that the internal states are maintained within a certain range. Homeostasis is critical for the survival of the organism. The stability attained is actually a dynamic equilibrium, in which continuous changes occur yet relatively uniform conditions prevail. Some examples of internal states for organisms include internal temperature, blood pressure, glucose level etc. Efficient regulation of homeostasis and defending it against perturbations requires adaptive behavioral strategies

Homeostatic reinforcement learning (HRL) as introduced by Keramati and Gutkin (2014) bridges the gap between associative and homeostatic learning process by proving that seeking reward is equivalent to the fundamental objective of physiological stability. The HRL framework can be viewed in Figure 7.1. The homeostatic space is defined as a multidimensional metric space in which each dimension represents one physiologically-regulated variable. At a time step t the physiological or internal state of the animal can be represented by a point in the homeostatic space denoted by $H_t = (h_{1,t}, h_{2,t}, h_{3,t}..., h_{N,t})$ where, $h_{i,t}$ is the value of i^{th} physiological state. The optimal homeostatic state can be denoted by $H^* = (h_1^*, h_2^*, h_3^*, ..., h_N^*)$. The actions taken by the organism in its environment lead to a particular outcome $K_t = (k_{1,t}, k_{2,t}, k_{3,t}..., k_{N,t})$ from the environment. The outcome changes the physiological state as $H_{t+1} = H_t + K_t$. In order to survive the organism should strive to take actions which do not deviate its physiological state H_t too far away from the optimal homeostatic state H^*.

In an HRL framework the *Drive* $D(H_t)$ of an organism in a physiological state H_t is defined as a non-linear mapping from the homeostatic space to a motivation space as shown in Figure 7.1. It is a metric of distance between the current physiological state H_t and the optimal homeostatic state H^*:

$$D(H_t) = \sqrt[m]{\sum_{i=1}^{N} |h_i^* - h_{i,t}|^n} \tag{7.1}$$

Where m and n are empirically set parameters which result in important non-linear effect on the mapping. The non-linear effect is important as it can be used to obtain a concave landscape which is inspired from the classical utility theory (Aumann, 1962). Moreover if the Drive is defined as $D(H_t) = -\ln p(H_t)$, then it can be the linked to the concept of *surprise* from information theory (Itti and Baldi, 2009).

Homeostatic reward function R_t^{HRL} for the organism can now be defined as the difference between the drives of 2 consecutive states

$$R_t^{HRL} = D(H_t) - D(H_{t+1}) \tag{7.2}$$

Thus the organism receives positive reward for reducing the drive i.e. moving closer to the optimal homeostatic state H^* and negative reward if the drive increases. This resonates with the *Drive Reduction Theory* (Seward, 1956). Finally, the homeostatic reward can be used by a regular RL-agent as a reward signal to learn the optimal action. Using the HRL framework the agent learns to stay close to the optimal homeostatic state.

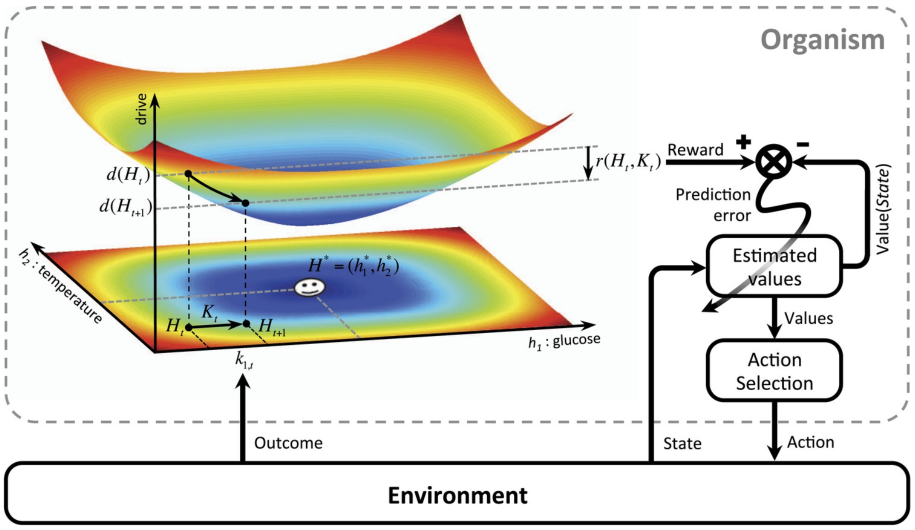


Figure 7.1: example of model with a 2D homeostatic space comprising of glucose and temperature of an organism

The NEB framework presented in the report can resemble the HRL framework by assuming the homeostatic space as a single dimensional state-space represented by the energy of the virtual tank $H_t = E_t$. The energy in the tank varies by $E_t^{in} - E_t^{out}$ in the NEB framework thus, the outcome K_t can be linked to this change as, $K_t = E_t^{in} - E_t^{out}$. The optimal homeostatic state H^* can be set equal to the maximum possible energy in the tank, $H^* = E_{max}$. Finally, the homeostatic reward function described in Equation 7.2 can be used by the agent to learn a policy that results in actions so that the homeostasis in-terms of the energy present in the virtual tank is always maintained.

The utility of connecting the two frameworks lies in the fact that behaviours which emerge in animals naturally due to their motivation to maintain energy homeostasis can be modelled and studied using robots. The model can be easily extended to account for changes that occur in the homeostatic equilibrium state during the life time of animals and the effect these changes have on the higher-level controller's decision process. This may enhance the efficiency and stability of robots in complex scenarios and result in more robust controllers.

7.3 Limitations

There are 2 major shortcomings in the current version of the NEB framework

7.3.1 Feeding function

As discussed in Section 3.2.2 it is assumed that the feeding function is designed by using a potential-based reward function defined for the task at hand. Thus, it is assumed that a reward function having such a characterstic can be defined for the task. In robotics, sometimes the task is too complex to come up with a well-behaved potential-based reward function. For instance, pick and place tasks, object manipulation task etc. In such cases it is only possible to define a sparse reward function for the task. When the reward function is sparse, feedback regarding robot's task proficiency is available only for a limited number of states. In such cases the present version of the framework cannot be implemented in a straightforward way.

7.3.2 Moving Away From Target

In some complex scenarios the robot may get stuck is a local minima of the feeding function whereby the optimal action may be to move away from the target so that later the robot can move towards the global minima. For instance consider a robot which is trying to reduces its distance from the target but is stuck in a state as shown in Figure 7.2. In-order to get to the target the robot needs to come out of the local minima by undertaking action A_t^{away}. It can be observed intuitively that this action will need the robot to move up the potential on which the feeding function is based. In the current version of the framework this is only possible if the energy stored in the tank before A_t^{away} is undertaken, is more than the energy used up by the actuators for undertaking A_t^{away}. This is because as soon as the robot moves away from the target its actuators act passively with respect to the energy stored in the tank due to the ISP property. Thus if the energy needed to undertake A_t^{away} is more than the energy stored in the tank, then the robot will not be able to come out of the situation and progress towards the target. This requires us to raise the maximum amount of energy that can be stored in the tank which may compromise the safety of the robot.

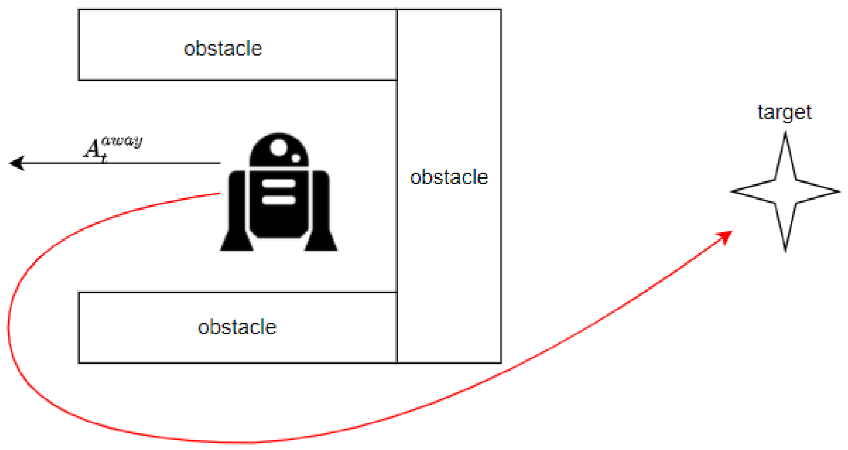


Figure 7.2: The robot is surrounded by obstacles from 3 sides and in order to reach the target it must move away from the target by undertaking action A_t^{away}

7.4 Future Work

The current version of the framework explores a deterministic way of adding energy in the tank based on a potential-based reward function. The future work will explore a different way of supplying energy back in the tank in a non-deterministic and more adaptive way. More specifically, a modulated Critic from a teacher agent can be used for this purpose. Here, a teacher agent refers to an RL agent whose Critic has accurately learned the value function for the task by using a general reward function. Note, the reward function need not be a potential based reward function, it can be sparse. The value function by definition, assigns a value to every state in the state space. Thus if the value function for the task is accurately learned by the Critic, a feedback about task efficiency is available for every state. Now, the difference between these feedback signals can be used to refill the energy tank of a student RL agent which has to learn the task with energy awareness in the NEB framework.

In summary, firstly a teacher RL agent can be trained in a simulation to do the task at hand in a conventional manner i.e. without the assumption of the reward function being a potential-based reward function and without the inclusion of an energy tank architecture, as shown in Figure 7.3. Then, the Critic of the teacher agent is used to refill the energy tank of an NEB framework in which a student RL-agent learns to the task with energy awareness see Figure 7.4. The function F in Figure 7.4 maps the value function of the teacher to energy going in student's energy tank.

Critic in a Deep RL framework is essentially a neural network, so the teacher provides energy using a black-box mapping to the student. Hence, it is important that F should be designed such that the energy in the tank never exceed the safety limit. This new way of refilling energy can tackle the limitations discussed in Section 7.3. Firstly, Critic can convert a sparse reward function to a continuous feedback signal defined for all the states. Secondly, if the teacher's Critic is able to learn the optimal value function then the problem of loosing energy while escaping the local minima may be solved as the optimal value function will provide more value to the action of escaping the local minima, for instance actions like A_t^{away} in Figure 7.2 will have higher value and hence will be provided with energy. Thus, while undertaking A_t^{away}, the actuators of the robot will operate in a non-passive mode.

Future work will explore this new way of supplying energy back to the tank.

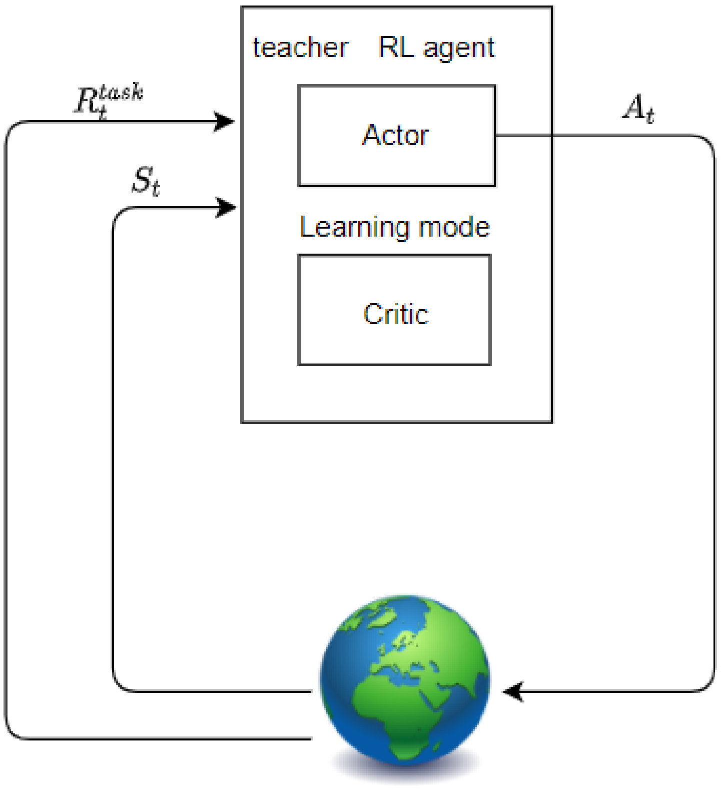


Figure 7.3: The teacher RL agent learns the value function which by definition is continuous on state space.

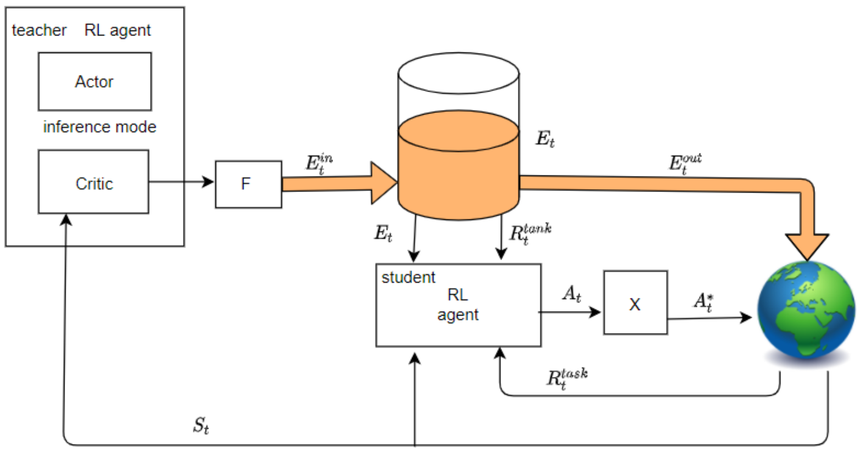


Figure 7.4: The critic from the teacher RL agent provides energy to refill the energy tank of the NEB framework in which student RL operates. F is used to modulate this mapping. The remaining structure and notations are similar to Figure 3.3

8 Conclusion

The report introduced a novel framework which combines the virtual energy tank architecture with the RL framework by embedding a virtual energy tank inside the RL loop. The energy tank provides a guaranteed passivity with respect to the energy stored in it to the robot's actuator while the RL framework provides a way to maintain the energy in the tank below a safety threshold. This is possible by initializing the energy tank with a certain small amount of energy and dynamically refilling it in proportion to a potential-based reward function. Updating the energy tank in this way maintains the energy below the safety limit not only in the inference mode but also in the learning mode.

By refilling the tank in proportion to how well the robot performs in a task, the framework endows the robot's actuators with the property of *Intelligent Selective Passivity* (ISP). Under ISP, the robot's actuators operate passively with respect to the energy stored in the virtual tank if the robot moves away from the target state. On the other hand if the robot moves towards the target state, the energy tank is refilled with proportional amount of energy and the actuators become non-passive. The RL framework provides positive reward to this non-passive state of actuators. Thus providing the robot with an opportunity to learn to avoid the energy tank singularity by performing well in the task. This approach solves the problem of loss of robot's autonomy due to hard passivity check encountered in virtual energy tank architecture. Further, no computationally expensive simulation is required to refill the energy tank

Chapter 4 provided a proof of concept for the framework. It also highlighted the role of hyper-parameters used in the framework. Of particular interest is the energy coefficient K. K defines the amount of energy entering the tank per unit distance the robot moves towards the target in a potential field defined around the target. Thus, K defines the energy economy in which the robot operates. It was shown how high value of K can lead to trajectories having high power profile.

The framework enables energy awareness in robot which can lead to high-level decision processes being based on internal energy states. For instance in Chapter 5 it was shown that how the RL-agent uses energy profile of the tank to answer the question of affordability. Such energy based behaviours can again be modulated in a predictable way by changing the value of energy coefficient K.

In Chapter 6 another application of the framework in terms of memory efficient learning was illustrated. The early stopping property provided by the passivity layer not only reduces the memory consumption by trimming the unsuccessful trials earlier but also indirectly increases the probability of sampling data-poinst having higher reward value while learning. This in-turn helps the learning algorithm to converge quicker.

In Chapter 7 it was proven how the proposed framework is a special case of the more general *Homeostatic Reinforcement learning* (HRL) framework. This analogy was made so that it can be utilized in the field of *bio-mimetic robotics* wherein theories such as the *Dynamic Energy Budgeting* in animals and their effect on animal behaviour are analyzed through the lens of robotics. Finally, two main limitations of the framework arising from the assumptions made regarding the energy feeding function and simplicity of the tasks chosen were discussed. Further, a way in which these limitations can be circumvented by shifting to a non-deterministic energy feeding function was briefly discussed and was attributed towards future work.

Bibliography

Arulkumaran, K., M. P. Deisenroth, M. Brundage and A. A. Bharath (2017), Deep reinforcement learning: A brief survey, **vol. 34**, no.6, pp. 26–38.

Aumann, R. J. (1962), Utility theory without the completeness axiom, *Econometrica: Journal of the Econometric Society*, pp. 445–462.

Barron, E. and H. Ishii (1989), The Bellman equation for minimizing the maximum cost, **vol. 13**, no.9, pp. 1067–1090.

Beeler, J. A. and D. Mourra (2018), To do or not to do: dopamine, affordability and the economics of opportunity, *Frontiers in integrative neuroscience*, **vol. 12**, p. 6.

Bertsekas, D. P. et al. (2000), *Dynamic programming and optimal control: Vol. 1*, Athena scientific Belmont.

Brockman, G., V. Cheung, L. Pettersson, J. Schneider, J. Schulman, J. Tang and W. Zaremba (2016), Openai gym, *arXiv preprint arXiv:1606.01540*.

Brodskiy, Y. (2013), *Robust autonomy for interactive robots*, volume 4.

Bustraan, M. C. (2021), *Developing quadratic programming constraints for human safety: an application of unified safety criteria to multi-task robot control*, Master's thesis, University of Twente.

Camlibel, K., A. A. Julius, R. Pasumarthy and J. M. Scherpen (2015), *Mathematical Control Theory*, Springer.

Duindam, V. and S. Stramigioli (2004), Port-based asymptotic curve tracking for mechanical systems, **vol. 10**, no.5, pp. 411–420.

Folkertsma, G. and S. Stramigioli (2017), Energy in Robotics, **vol. 6**, no.3, pp. 140–210, ISSN 1935-8253, doi:10.1561/2300000038.

Folkertsma, G. A., S. S. Groothuis and S. Stramigioli (2018), Safety and guaranteed stability through embedded energy-aware actuators, in *2018 IEEE International Conference on Robotics and Automation (ICRA)*, IEEE, pp. 2902–2908.

Haddadin, S. and E. Croft (2016), Physical human–robot interaction, in *Springer handbook of robotics*, Springer, pp. 1835–1874.

Hopfield, J. J. (1988), Artificial neural networks, **vol. 4**, no.5, pp. 3–10.

Hwangbo, J., J. Lee, A. Dosovitskiy, D. Bellicoso, V. Tsounis, V. Koltun and M. Hutter (2019), Learning agile and dynamic motor skills for legged robots, **vol. 4**, no.26.

Imre, B. (2021), An investigation of generative replay in deep reinforcement learning. http://essay.utwente.nl/85772/

Itti, L. and P. Baldi (2009), Bayesian surprise attracts human attention, **vol. 49**, no.10, pp. 1295–1306.

Keramati, M. and B. Gutkin (2014), Homeostatic reinforcement learning for integrating reward collection and physiological stability, *Elife*, **vol. 3**, p. e04811.

Kingma, D. P. and J. Ba (2014), Adam: A method for stochastic optimization, *arXiv preprint arXiv:1412.6980*.

Kiumarsi, B., K. G. Vamvoudakis, H. Modares and F. L. Lewis (2017), Optimal and autonomous control using reinforcement learning: A survey, **vol. 29**, no.6, pp. 2042–2062.

Kooijman, B. and S. Kooijman (2010), *Dynamic energy budget theory for metabolic organisation*, Cambridge university press.

Kulkarni, T. D., K. R. Narasimhan, A. Saeedi and J. B. Tenenbaum (2016), Hierarchical deep reinforcement learning: Integrating temporal abstraction and intrinsic motivation, *arXiv* *preprint arXiv:1604.06057*.

Laffranchi, M., N. G. Tsagarakis and D. G. Caldwell (2009), Safe human robot interaction via energy regulation control, in *2009 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 35–41, doi:10.1109/IROS.2009.5354803.

Lewis, F. L., D. Vrabie and V. L. Syrmos (2012), *Optimal control*, John Wiley & Sons.

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

Lygeros, J. (2004), On reachability and minimum cost optimal control, **vol. 40**, no.6, pp. 917–927.

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

Mnih, V., K. Kavukcuoglu, D. Silver, A. A. Rusu, J. Veness, M. G. Bellemare, A. Graves, M. Riedmiller, A. K. Fidjeland, G. Ostrovski et al. (2015), Human-level control through deep reinforcement learning, **vol. 518**, no.7540, pp. 529–533.

Ng, A. Y., D. Harada and S. Russell (1999), Policy invariance under reward transformations: Theory and application to reward shaping, in *Icml*, volume 99, pp. 278–287.

Ortega, R., A. Van Der Schaft, F. Castanos and A. Astolfi (2008), Control by interconnection and standard passivity-based control of port-Hamiltonian systems, **vol. 53**, no.11, pp. 2527–2542.

Paszke, A., S. Gross, F. Massa, A. Lerer, J. Bradbury, G. Chanan, T. Killeen, Z. Lin, N. Gimelshein, L. Antiga, A. Desmaison, A. Kopf, E. Yang, Z. DeVito, M. Raison, A. Tejani, S. Chilamkurthy, B. Steiner, L. Fang, J. Bai and S. Chintala (2019), PyTorch: An Imperative Style, High-Performance Deep Learning Library, in *Advances in Neural Information Processing Systems 32*, Eds. H. Wallach, H. Larochelle, A. Beygelzimer, F. d'Alché-Buc, E. Fox and R. Garnett, Curran Associates, Inc., pp. 8024–8035. http://papers.neurips.cc/paper/9015-pytorch-an-imperative-style-high-performance-deep-learning-library.pdf

Puterman, M. L. (1990), Chapter 8 Markov decision processes, in *Stochastic Models*, volume 2 of *Handbooks in Operations Research and Management Science*, Elsevier, pp. 331–434, doi:https://doi.org/10.1016/S0927-0507(05)80172-0. https://www.sciencedirect.com/science/article/pii/S0927050705801720

Raiola, G., C. A. Cardenas, T. S. Tadele, T. de Vries and S. Stramigioli (2018), Development of a Safety- and Energy-Aware Impedance Controller for Collaborative Robots, **vol. 3**, no.2, pp. 1237–1244, doi:10.1109/LRA.2018.2795639.

Rangel, A., C. Camerer and P. R. Montague (2008), A framework for studying the neurobiology of value-based decision making, **vol. 9**, no.7, pp. 545–556.

van der Schaft, A. (2016), *L2-Gain and Passivity Techniques in Nonlinear Control*, Springer.

Schaul, T., J. Quan, I. Antonoglou and D. Silver (2015), Prioritized experience replay, *arXiv preprint arXiv:1511.05952*.

Schindlbeck, C. and S. Haddadin (2015), Unified passivity-based Cartesian force/impedance control for rigid and flexible joint robots via task-energy tanks, in *2015 IEEE International Conference on Robotics and Automation (ICRA)*, pp. 440–447, doi:10.1109/ICRA.2015.7139036.

Seward, J. P. (1956), Drive, incentive, and reinforcement., **vol. 63**, no.3, p. 195.

Silver, D., G. Lever, N. Heess, T. Degris, D. Wierstra and M. Riedmiller (2014), Deterministic policy gradient algorithms, in *International conference on machine learning*, PMLR, pp. 387–395.

Singh, S., R. L. Lewis and A. G. Barto (2009), Where do rewards come from, in *Proceedings of the annual conference of the cognitive science society*, Cognitive Science Society, pp. 2601–2606.

Stramigioli, S., C. Secchi, A. J. van der Schaft and C. Fantuzzi (2005), Sampled data systems passivity and discrete port-hamiltonian systems, **vol. 21**, no.4, pp. 574–587.

Sutton, R. S. (1988), Learning to predict by the methods of temporal differences, **vol. 3**, no.1, pp. 9–44.

Sutton, R. S. and A. G. Barto (2011), Reinforcement learning: An introduction.

Sutton, R. S., D. A. McAllester, S. P. Singh, Y. Mansour et al. (1999), Policy gradient methods for reinforcement learning with function approximation., in *NIPs*, volume 99, Citeseer, pp. 1057–1063.

Tadele, T. S., T. de Vries and S. Stramigioli (2014), The safety of domestic robotics: A survey of various safety-related publications, **vol. 21**, no.3, pp. 134–142.

Todorov, E., T. Erez and Y. Tassa (2012), MuJoCo: A physics engine for model-based control, in *2012 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pp. 5026–5033, doi:10.1109/IROS.2012.6386109.

Weng, L. (2018), Policy Gradient Algorithms, *lilianweng.github.io/lil-log*. https://lilianweng.github.io/lil-log/2018/04/08/policy-gradient-algorithms.html

Willems, J. C. (1972), Dissipative dynamical systems part I: General theory, **vol. 45**, no.5, pp. 321–351.

Yoganandan, N., F. Pintar, D. Maiman, J. Cusick, A. Sances and P. Walsh (1996), Human head-neck biomechanics under axial tension, **vol. 18**, no.4, pp. 289–294, ISSN 1350-4533, doi:https://doi.org/10.1016/1350-4533(95)00054-2. https://www.sciencedirect.com/science/article/pii/1350453395000542

Zult, J. (2020), *Achieving Stable and Safe Physical Interaction for a Fully Actuated Aerial Robot using Energy Tank-Based Interaction Control*, Master's thesis, University of Twente.