31010: Directed Study 3

Reinforcement Learning

Deep Q-Learning for

Controlling Grid-connected Inverter

Student Name: Rohan Saha Turai

Student ID: 13093012

Table of Contents

List of Figures

List of Tables

1. Problem Statement

The report outlines an attempt to design a controller using Deep Q-learning for a H-Bridge. Power electronic inverters transfer power from a DC-source to an AC-load. This is achieved by manipulating the internal power switches (on or off) to synthesise an AC-voltage in order to generate a desired AC-current. A key feature of these power converters is that they are autonomous systems. Hence, a controller is needed to properly operate them.

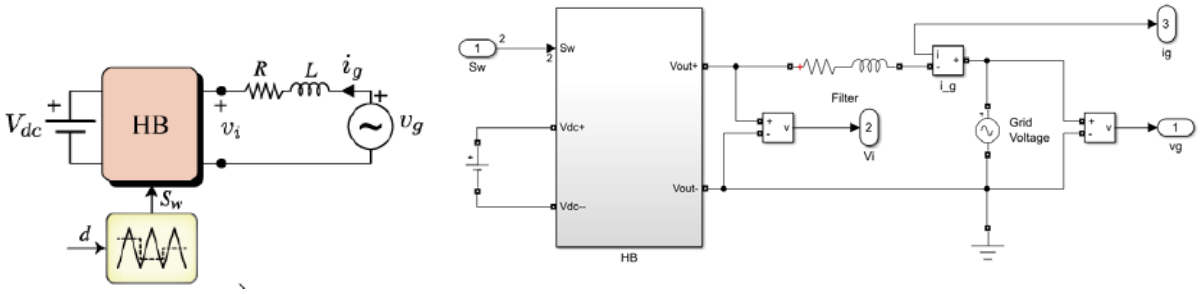

Figure 1: H-bridge based grid-connected inverter a) Block diagram b) Simulink model

Figure 1 shows a H-Bridge inverter connected to a grid load. It is required that a digital controller provides a suitable duty cycle, d, in order to track a desired sinusoidal grid current reference, i_g. The signal d is pulse modulated with a triangular carrier wave to generate the switching sequences $S_w \in \{\{0,0\}, \{0,1\}, \{1,0\}\}$. A low pass *RL* filter is interfaced with the grid which helps filter out high frequency harmonics produced by the switched inverter voltage, v_i. The grid voltage load is a standard single-phase low voltage in Australia, which is $240\ V_{rms}$ with a frequency of $f_0 = 50\ Hz$.

In summary, the object of this task is to design a reinforcement learning agent using deep Q-learning algorithm to control the switching of the H-Bridge inverter so that it is able to track a sinusoidal grid current i_g of frequency $f_0 = 50\ Hz$.

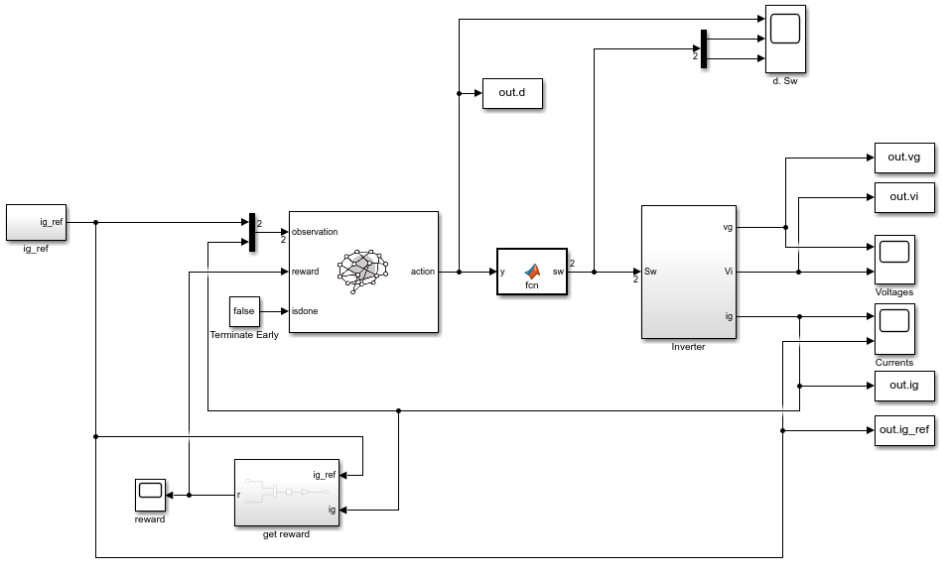

Figure 2: Simulink implementation of the overall model

2. Deep Q-Learning

Reinforcement learning is a semi-supervised learning method where the agent takes an action a_t at time step t by observing the states s_t from the environment and receives a reward r_{t+1} in the next time step, which is a measure of how desirable the action is given the same state, and the environment transitions into a new state s_{t+1}. The objective of the agent is to choose the optimal actions to maximise the overall rewards received. The Q-value $Q^{\pi}(s,a)$ is the measure of the overall expected reward assuming the agent is in state s and performs action a, and then thereafter continues to take actions following the policy π. The Q-value is defined as follows:

$$Q^{\pi}(s,a) = E_{\pi}\{\sum_{k=0}^{\infty} \gamma^{k} r_{t+k+1} \,|s_t = s, a_t = a\}$$

where, r is the reward per step and $\gamma \in (0,1)$ is the discount factor, which represents the relative importance of future reward to the current state. The optimal policy π can be obtained by choosing the action with the maximum Q-value for the given state, which obtains the maximum overall reward. Therefore, the main objective of the learning algorithm is to approximate the Q-value to obtain policy π.

In Q-learning, a finite Q-table can be constructed to store Q-values corresponding to each state-action, which the agent updates at each time step by the following update rule:

$$Q(s,a) \leftarrow Q(s,a) + \alpha\left[r_{t+1} + \gamma \max_{a} Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)\right]$$

where $\alpha \in (0,1)$ is the learning rate. However, for high dimensional state-action space, the Q-value is approximated with a deep neural network to capture the non-linear relationship. The Q-network is trained by minimising a sequence of loss functions $L_i(\theta)$ that changes at each iteration i,

$$L_i(\theta_i) = E_{s,a\sim\rho(\cdot)}\left[\left(y_i - Q(s,a;\theta_i)\right)^2\right],$$

where $y_i = E_{s'\sim\varepsilon}[r + \gamma \max_{a'} Q(s',a';\theta_{i-1})|s,a]$ is the target and θ_i are the weights of the networks for iteration i, and $\rho(s,a)$ is the probability distribution over sequences s and actions a (Minh et al., 2013).

Experience replay (Lin 1993) is used to train the agent where the agent's experiences at each time step is stored in a dataset. An experience is defined as a tuple of four elements, $e_t = (s_t, a_t, r_t, s_{t+1})$, at time step t. The experiences are pooled over many episodes into a replay memory. Samples of randomly drawn experiences from the dataset is used to update the Q-network during the Q-learning. After performing experience replay, the agent selects an action according to an ε-greedy policy that chooses between exploring random actions or performing the action with current maximum Q-value depending on the probability threshold ε.

Furthermore, two deep networks with the same parameters are created. The first one retrieves the Q-values, while the second one gives the target Q-value. The target Q-network is periodically updated that fixes the target Q-value and stabilises the training process.

The deep Q-learning algorithm for HB inverter is summarised in Figure 3.

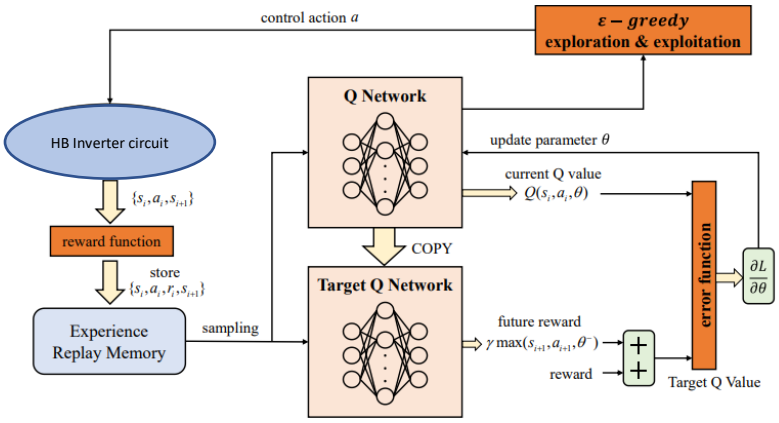

Figure 3: DQN learning algorithm (Cui, Yan & Zhang, 2020)

3. Observation Space

Defining the appropriate states in a Markov decision process is an important step in the algorithm. The state space has to be designed such that it includes sufficient information to fully describe the environment. But including redundant states adversely affects the performance of the network as the parameter search space increases exponentially making it difficult for the agent to learn.

The control target is to track the reference grid current, therefore, the output grid current $i_g(t)$ and the reference current $i_g\_ref(t)$ are considered as the underlying signals to determine the states. The two signals are multiplexed into an array with dimension $[2 \times 1]$ and thus, this is the observation vector size. The observations states at timestep t are defined as follows:

$$S_t = \{i_g\_ref(t), i_g(t)\}$$

4. Action Space

Designing the appropriate action space is a complicated task due to the lack of established rules. Deep Q-networks are used for discrete action spaces and fewer action combinations reduces the complexity of the task, making it easier to learn. Originally duty cycle is used to generate the switching sequences for the HB inverter. Because the duty cycle is a continuous signal ranging between 0 to 1, discretising the duty cycle signal reduces the performance of the controller. To achieve stable performance with duty cycle as the control signal, the signal needs to be discretised into infinitesimally small steps and these increases the action space exponentially, making it near impossible for the network to learn.

However, the duty cycle is modulated with a triangular carrier wave which generates discrete finite switching sequences as input to the HB. So, a discrete finite action space is constructed as $Sw = \{\{0,0\},\{0,1\},\{1,0\}\}$ and there is no modulation with the duty cycle required. The agent outputs an action index $Sw_i = \{0,1,3\}$, which corresponds to the switching signals $Sw = \{\{0,0\},\{0,1\},\{1,0\}\}$ respectively. A MATLAB function takes the action index Sw_i (the output of the agent) and generates the appropriate switching signal Sw, which is then used as the input to the HB.

5. Reward Function

The reward function is a mapping of each state-action pair of the environment to a scalar number that represents the desirability of the state. When an agent interacts with the environment, it can observe the changes in the state and reward signal through the actions and uses the reward signal as feedback to conclude how the action performs given the current and previous states. If taking an action a_t in state s_t receives a high reward, the probability of taking that action a_t in state s_t will increase; otherwise, the probability of taking action a_t in state s_t will be lowered.

The main objective of the agent is to reduce the error $e(t) = i_g\_ref - i_g$ to the minimum, ideally to zero at limit $t \to \infty$ (steady state), so that the controller tracks the sinusoidal reference current perfectly. The reward function is designed to penalise the agent more if the error is large i.e., the gap between the inverter and reference current is large. The agent tries to maximise the reward, hence, minimise the error as much as possible. The minimum error is zero, hence, this is also the maximum reward. The reward function is designed as the negative absolute of the error function so that if the error is reduced, the reward is large. The reward function r is designed as follows:

$$r = -0.1|e(t)|$$

6. Critic Deep Neural Network

A deep neural network is used to approximate the Q-value representation. The input to the DNN is the state representation (the observations) and the outputs correspond to the predicted Q-values of the individual action for the input state. The DNN architecture summary is shown in Figure 4. The input layer is the feature input layer with number of neurons equal to the number of observations. The first and second hidden layers have 32 neurons, and each hidden layer is followed by ReLu activation function. The final layer is a fully connected layer with 2 neurons equal to the number of actions.

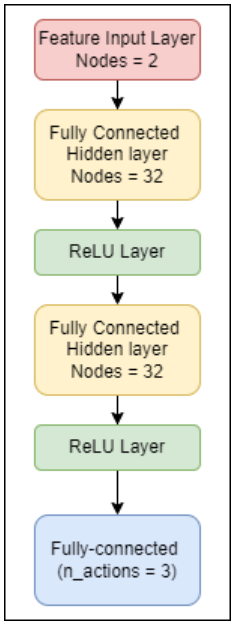

Figure 4: DNN architecture

7. Hyperparameters and Training

The hyperparameters for the DQN agent is given in Table 1. The agent is training with the hyperparameters given in Table 2.

Table 1: Hyperparameters of the agent

Hyperparameter	Value	Definition
Optimiser	ADAM	The optimiser used to train the network of the critic (the Q-value approximator)
Learning rate	1×10^{-4}	The learning rate for the ADAM optimiser
L2 Regularisation Factor	1×10^{-4}	Factor for L_2 regularisation (weight decay)
Gradient Threshold	1	Threshold value for the critic gradient
Sample Time	$2 \times 10^{-5}\ s$	The sampling time of the agent
Experience Buffer Size	1×10^{6}	NN updates are sampled from this number of most recent frames
Use Double DQN	$False$	Don't use double DQN for value function target updates
Target Smooth Factor	1×10^{-3}	Smoothing factor for target critic updates
Target Update Frequency	100	Number of steps between target critic updates
Gamma γ	0.90	The discount factor used in the Q-learning update
Mini Batch-size	256	Number of training samples over which each weights update is computed
Minimum Epsilon ε	0.01	The ε is the probability threshold to either randomly select an action or select the action that maximises the state-action value function. At the end of each training time step, if ε is greater than minimum ε, it is update as: *Epsilon = Epsilon*(1-EpsilonDecayRate)*
Final Epsilon ε	1.00	
ε decay rate	1×10^{-4}	

Table 2: Training hyperparameters

Hyperparameter	Value	Definition
Maximum episodes	1000	The maximum number of episodes to train the agent
Maximum steps per episode	800	The maximum of number of timesteps to run per episode. It is calculated as $ceiling\left(\frac{T_{SIM}}{T_{SAMPLE}}\right)$
Stop Training Criteria	*Average Reward*	The training is stop when the average reward equals or exceeds the critical value
Stop Training Value	-21	The training terminates if the average reward equals or exceeds this critical value

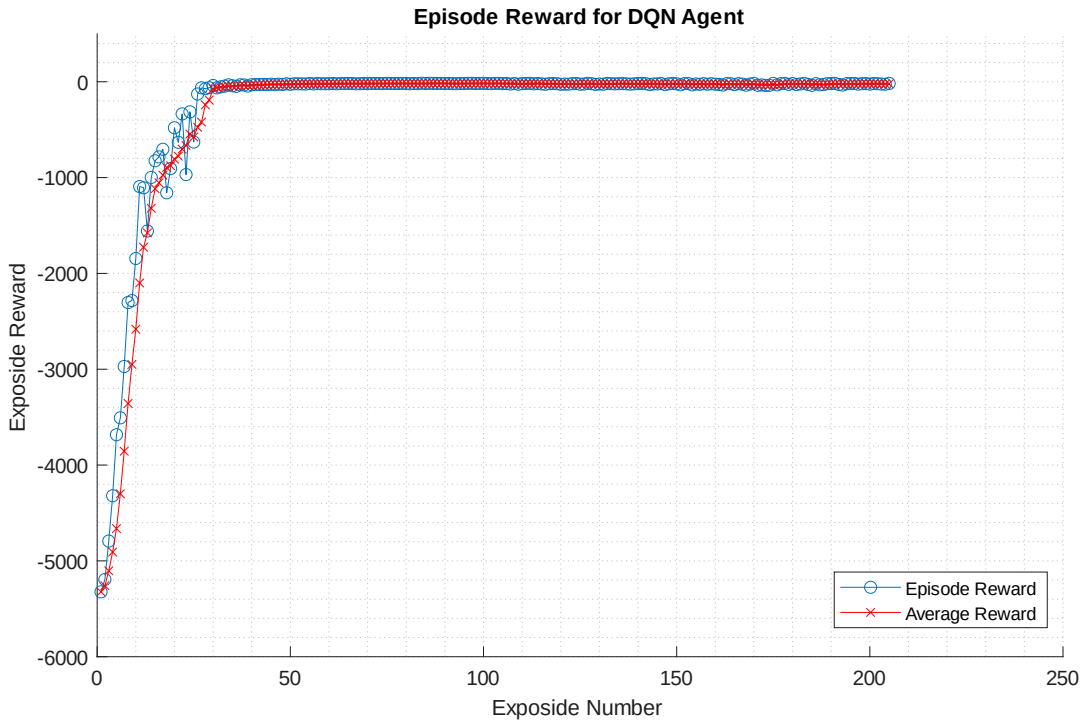

Figure 5: Episode Reward curves for DQN Agent

Figure 5 shows the episode reward curve during the training. The training took about 49 minutes, and the maximum saturated reward was achieved after 50 episodes. The average reward at the end of the training was -20.7841, which is more than the critical value defined, and hence the training stopped after only 205 episodes. As seen from the figure, the reward saturates and reaches maximum many episodes earlier than the defined maximum episodes (1000). The agent is trained on UTS iHPC cluster using Intel Xeon Gold 6238R 2.2GHz 28 cores CPU with 180GiB of RAM; no GPU was required for the training as the training time was already sufficiently small. Because the many experiences were simulated during the training, it was suitable to use CPU to achieve greater performance rather than a GPU. No hardware parallelisation was used in this task.

8. Results and Discussion

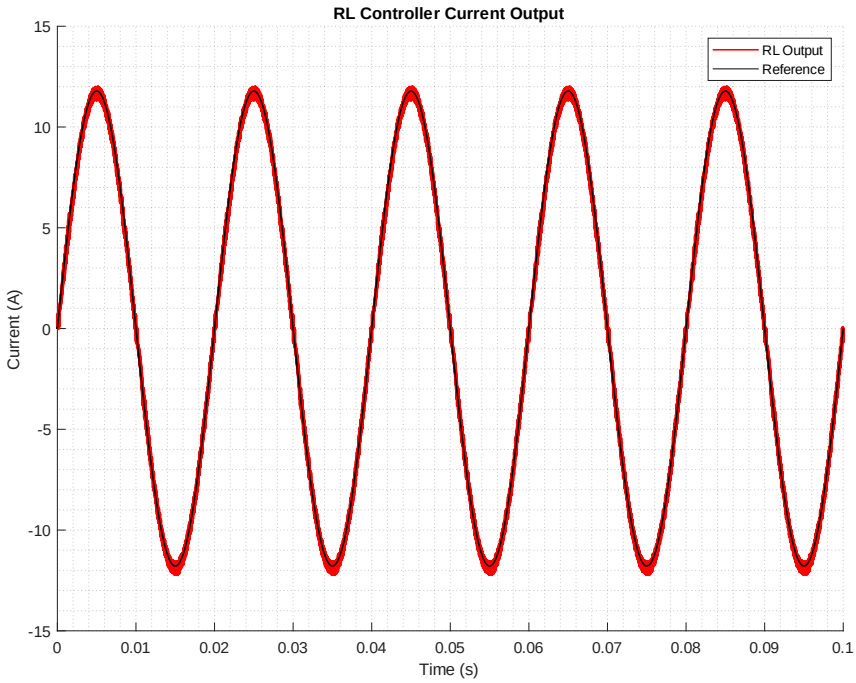

Figure 6: Output Grid Current of RL agent

Figure 6 shows the reference sinusoidal grid current $i_g\_ref(t)$ (black) and the output of the controlled current $i_g(t)$ (red). **As seen in the figure, the agent is able to perfectly track the reference sinusoidal current and thus this satisfies the control object of this task very accurately.** Tracking a non-stationary reference is a difficult control task, however, the reinforcement learning is able to track the sinusoidal current of frequency $f_o = 50\ Hz$ with high accuracy. There is some noise in the output current, but this is expected noise because of the high frequency switching harmonics and cannot be eliminated without using additional high frequency filters. The high frequency harmonics can be reduced by increasing the switching frequency or using multi-stage inverters. The HB inverter used in this task is a single stage inverter and there is always some high frequency harmonics in the output. Therefore, under these conditions the controller performs exceptionally well in tracking the sinusoidal reference current.

The performance of the DQN controller is directly compared with a more classical single-loop digital PI controller that is tuned using the parameters given in Table 3. The tuning steps of the digital PI controller is not discussed in this report as this is beyond the scope of this report and covered in a separate study.

Table 3: Digital PI Controller Parameters

Parameters	**Value**	**Definition**
PWM frequency	$20\ kHz$	The frequency of the modulating triangular carrier wave
Sampling Time	$2 \times 10^{-5}\ s$	The sampling time of the digital controller
k_P_d	-0.05866	The **discrete** proportional
k_I_d	0.049031	

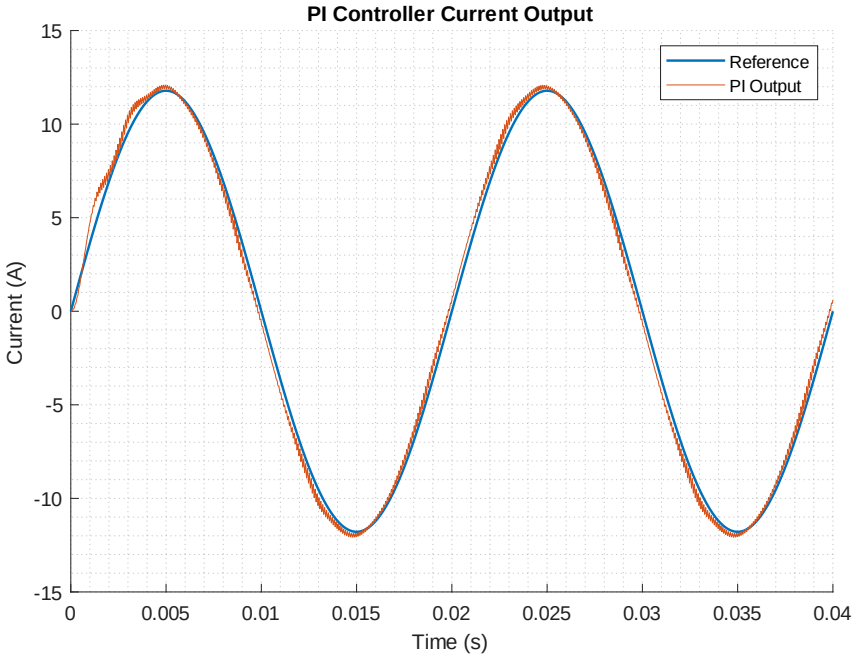

Figure 7: Output Grid Current of PI Controller

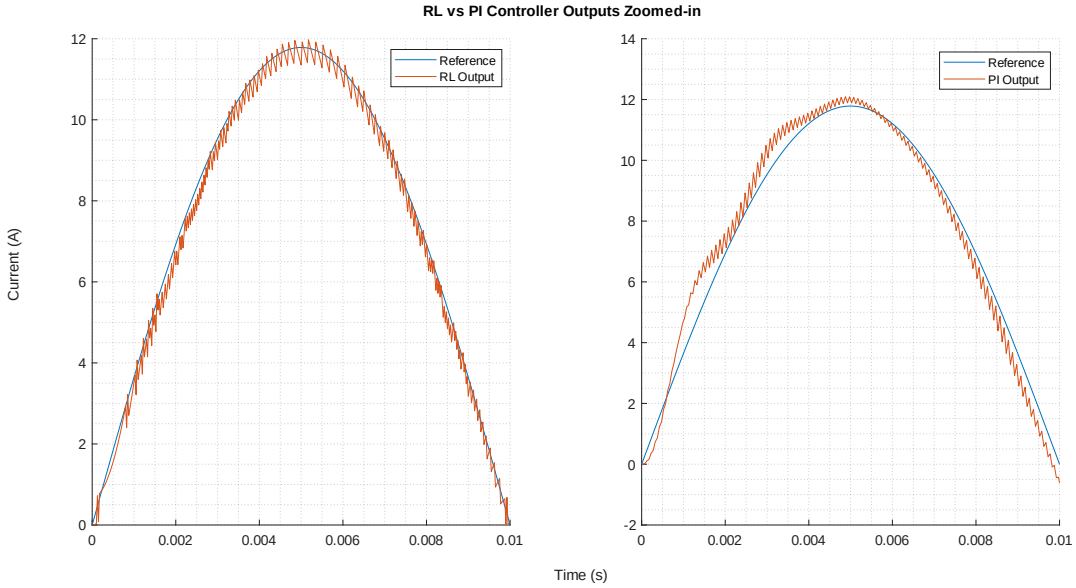

Figure 8: Output Current Comparison of RL (left) and PI (right) Controllers

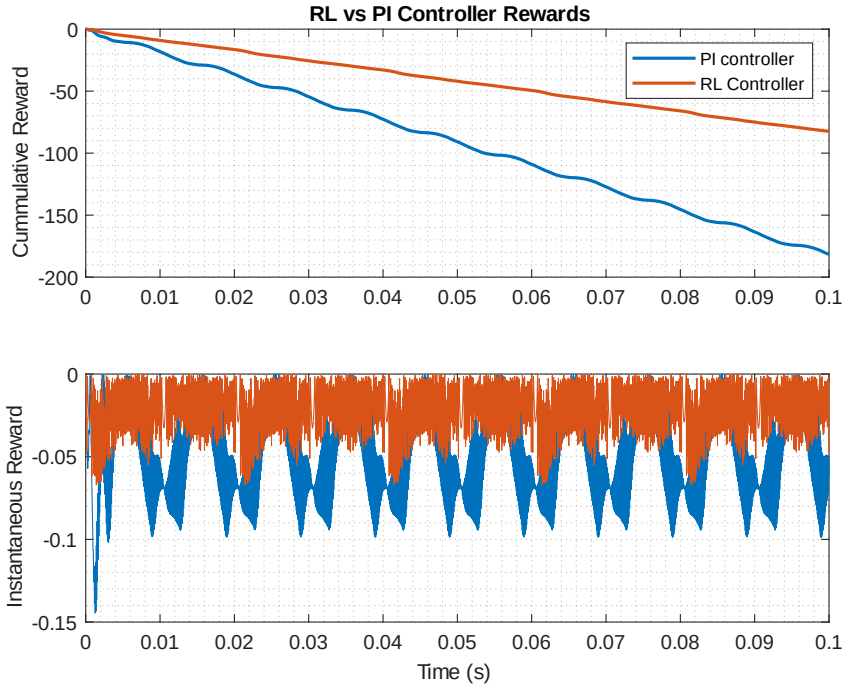

Figure 9: Cumulative Reward Comparison of RL and PI Controllers

Figure 8 shows the current output with the digital PI controller, Figure 8Figure 9 compares the outputs with the RL and PI controllers in a zoomed-in view, and Figure 9 shows the cumulative reward and rewards at each time step for both the controllers. The digital PI controller is able to follow the sinusoidal reference as well, however, the output doesn't perfectly track the reference. This is further confirmed in Figure 8 which shows that the output with the PI controller (right) doesn't perfectly overlap with the reference current. On the contrary, the RL controller does not show such behaviour and is always able to track the reference perfectly. The right plot in Figure 8 further confirms this by showing that the red graph always overlaps the reference in blue. This validates that the RL controller performs better than the digital PI controller.

The reward curves in Figure 9 provides a similar conclusion. The cumulative reward is calculated by summing the rewards at each time step. As defined earlier, the reward is the negative absolute error function. The high the reward, the lower the error and hence, the better the tracking. The cumulative reward for the PI controller (blue) is always smaller than the cumulative reward of the RL controller. This means that the overall error of the PI controller is always larger than the overall error of the RL controller. **Therefore, it can be concluded that the RL controller performs better than the digital PI controller in tracking the reference with smaller error.**

Test with Power Variance

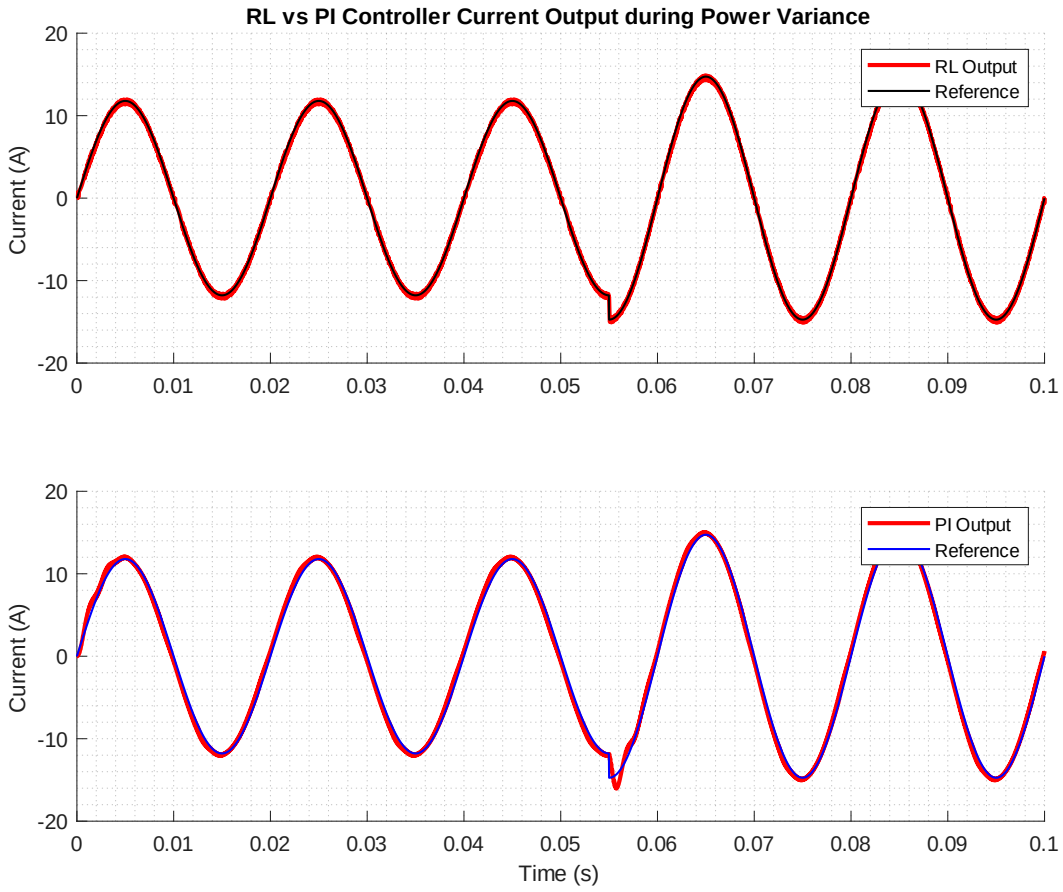

Figure 10: Comparison of RL and PI controllers during power variance

The robustness of the controllers is tested by simulating a scenario where the grid load changes instantaneous and thus the reference current also changes suddenly. The power reference suddenly increases at $t = 0.055\ s$, which increases the amplitude of the current instantaneously. This can be observed in Figure 10 where there is a sharp downward peak at $t = 0.055s$ and the amplitude of the current is larger from that time onwards.

In the sudden power variance scenario, the RL controller is able to perfectly track the new reference current with increased amplitude. The output always overlaps with the reference, even during the transition period at $t = 0.055\ s$, and continues to track with minimum error. On the contrary, even though the PI controller is eventually able to track the current with increased amplitude from $t = 0.055\ s$ onwards, but the output current does not follow the reference during the transition period at exactly $t = 0.055\ s$. **This further shows that the RL controller is more robust than the digital PI controller and is able to quickly recover during sudden transients without failing to track the reference.**

9. Conclusion and Future Works

In this report, a deep Q-learning based controller for H-Bridge grid-connected inverter is proposed. The control process is described as a Markov decision process, and the observation space, action space and the reward function are designed. A deep neural network is designed to estimate the Q-value function which represents the relation between state and action pairs. The agent explores the action space while learning the optimal control law by looking into states and rewards. A controller is successfully designed using deep Q-learning and verified by numerical simulations. Furthermore, the performance of the RL controller is compared and validated against a traditional digital PI controller.

There are scopes of further improvements to this study. Firstly, deep Q-learning is for discrete action spaces, but usually a continuous control is used along with PWM to control the inverter. Agents, such as DDPG and TD3, will be explored in the future which are able to work with continuous actions. The robustness of the RL controller can be further validated in the presence of sensor noises and observers. Methods of deploying the trained agent on microcontrollers will be explore so that it can be used with physical systems. The performance of the controller can also be validated using physical systems once implemented on microcontrollers or digital computers. Moreover, the method will be extended for more complex systems such 3-level NPC inverters, DC-DC buck converters, etc.

References

Cui, C., Yan, N., & Zhang, C. (2020). An Intelligent Control Strategy for buck DC-DC Converter via Deep Reinforcement Learning. https://arxiv.org/abs/2008.04542

Lin, L. (1993). Reinforcement Learning for Robots Using Neural Networks. https://apps.dtic.mil/sti/citations/ADA261434

MATLAB and Reinforcement Learning Toolbox Release 2021a, The MathWorks, Inc., Natick, Massachusetts, United States.

Mnih, V., Kavukcuoglu, K., Silver, D., Graves, A., Antonoglou, I., Wierstra, D., & Riedmiller, M. (2013). Playing Atari with Deep Reinforcement Learning. https://arxiv.org/abs/1312.5602