Reinforcement Learning based Non-Linear Flight Control

A project report submitted

in partial fulfillment for the award of the degree of

Bachelor of Technology

in

Aerospace Engineering

by

Subrahmanya V Bhide

Department of Aerospace

Indian Institute of Space Science and Technology

Thiruvananthapuram, India

May 2022

Certificate

This is to certify that the project report titled ***Reinforcement Learning based Non-Linear Flight Control*** submitted by **Subrahmanya V Bhide**, to the Indian Institute of Space Science and Technology, Thiruvananthapuram, in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Aerospace Engineering** is a bonafide record of the original work carried out by him/her under my supervision. The contents of this project report, in full or in parts, have not been submitted to any other Institute or University for the award of any degree or diploma.

Dr. Dhayalan R
Assistant Professor

Dr. Aravind Vaidyanathan
HOD, Aerospace Department

Place: Thiruvananthapuram
Date: May 2022

Declaration

I declare that this project report titled ***Reinforcement Learning based Non-Linear Flight Control*** submitted in partial fulfillment for the award of the degree of **Bachelor of Technology** in **Aerospace Engineering** is a record of the original work carried out by me under the supervision of **Dr. Dhayalan R**, and has not formed the basis for the award of any degree, diploma, associateship, fellowship, or other titles in this or any other Institution or University of higher learning. In keeping with the ethical practice in reporting scientific information, due acknowledgments have been made wherever the findings of others have been cited.

Place: Thiruvananthapuram
Date: May 2022

Subrahmanya V Bhide
(SC18B030)

This project report is dedicated to AI and ML enthusiasts, Nihharranjan Pradhan an alumnus of IIST, and the whole IIST family . . .

Acknowledgements

I would love to take this opportunity to acknowledge everyone who has helped me during this project. First I would like to express my gratitude to my guide Dr. Dhayalan R for his constant support and motivation throughout the project to work and try out newer things. I must also thank Dr. Vineeth B S for helping me in understanding reinforcement learning better and related concepts about the problem. I also would like to extend my heartfelt gratitude to Late Mr. Nihharanjan Pradhan for helping me during project and for his guidance. I must even thank my friends Sri Aditya Deevi, Aditya Venkateshwaran, and Asish Kumar Mishra for helping me out with remote GPU connections and jupyter notebook management. I cannot leave out my friends and family for being a constant source of motivation and being there for me when required. ...

Subrahmanya V Bhide

Abstract

In this project, a reinforcement learning based non-linear flight controller is developed and studied. For the start, pure pitching motion is chosen and it is formulated as a reinforcement learning problem. Different RL algorithms are implemented, various neural network architectures are tried, hyper-parameters selected, and reward functions explored. The different agents explored are DDPG, A3C and SAC agents. After probing these different agents, reward engineering was done for the DDPG agent where different mathematical operations were used as part of reward functions. Also at the start, the state vector was $(q, \Delta\theta, \theta_{goal})$, improvising this at different stages into $(q, \Delta\theta, \dot{\theta})$ and then $(q, \Delta\theta, \theta)$. The best result was obtained for the DDPG agent using the state vector $(q, \Delta\theta, \theta)$. The results obtained are not conclusive to permit longitudinal motion controller development. There exist steady state errors up to 0.3 degrees and some high frequency oscillations that need to be handled.

Contents

List of Figures **xiii**

List of Tables **xvii**

List of Algorithms **xix**

Abbreviations **xxi**

Nomenclature **xxiii**

1 Introduction **1**

1.1 Flight Control and Controllers 1

1.2 Reinforcement Learning 2

1.3 Literature Review 2

2 Problem Description **5**

2.1 Equations of Motion 6

2.2 Pure Pitching Motion 8

2.3 Aircraft Details 9

3 Reinforcement Learning **10**

3.1 Basic ideas 10

3.2 Deep Deterministic Policy Gradients 12

3.3 Advantage Methods 15

4 Training Setup **21**

4.1 Device and platforms 21

4.2 Hyper parameters 22

4.3 Reward Functions 22

4.4 Neural Network Architecture 26
4.5 Training methodology 27
4.6 State Vector Initialization 27

5 Results and Discussions **28**
5.1 Results from the first neural network configuration 28
5.2 Change of States 38
5.3 Reward Engineering 44
5.4 Visiting DDPG again 45

6 Conclusion and Future Works **71**
6.1 Conclusion 71
6.2 Future Works 72

Bibliography **73**

Appendices **77**

A Packages used for the implementation **77**
A.1 Gym-AI 77
A.2 RL Keras 77

List of Figures

2.1 Different axes system defined for a conventional aircraft [9] 5
2.2 Flow diagram showing the sequence of operations for the controller to be developed 7
2.3 The Euler Angles defined for a conventional aircraft 7

3.1 A general RL model 10
3.2 A representation of DQN 13
3.3 A representation of DDPG 13
3.4 A schematic of the A3C method 17

4.1 Quadratic reward function variation 24
4.2 Linear and ReLu activation functions 26

5.1 Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 4° 29
5.2 Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 0° 30
5.3 Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 3° 31
5.4 Results obtained for A3C agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 4° 33
5.5 Results obtained for A3C agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 2° 34
5.6 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 3° 35

5.7 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 1° 36
5.8 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 1° 37
5.9 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 0^\circ$ 39
5.10 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 2^\circ$ 40
5.11 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 4^\circ$ 41
5.12 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = -3^\circ$ 42
5.13 Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = -6^\circ$ 43
5.14 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1^\circ$ 46
5.15 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 3^\circ$ 47
5.16 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 4^\circ$ 48
5.17 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1^\circ$ 49
5.18 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -2^\circ$ 50
5.19 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -3^\circ$ 51
5.20 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -5^\circ$ 52
5.21 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 0^\circ$ 54
5.22 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 0.5^\circ$ 55

5.23 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1^\circ$ 56
5.24 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1.5^\circ$ 57
5.25 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 2^\circ$ 58
5.26 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 2.5^\circ$ 59
5.27 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 3^\circ$ 60
5.28 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 4^\circ$ 61
5.29 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -0.5^\circ$ 62
5.30 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1^\circ$ 63
5.31 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1.5^\circ$ 64
5.32 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -2^\circ$ 65
5.33 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -3^\circ$ 66
5.34 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -4^\circ$ 67
5.35 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -7^\circ$ 68
5.36 Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -8^\circ$ 69

List of Tables

2.1 Mass, Inertia and Geometric properties of the aircraft. 9
2.2 Aerodynamic properties of the given aircraft. 9

5.1 Typical values and Scaling factors for the terms in the new reward function 39

List of Algorithms

3.1 DDPG 14
3.2 A3C 18
3.3 SAC 20

Abbreviations

DDPG	Deep Deterministic Policy Gradient
SAC	Soft Actor Critic
RL	Reinforcement Learning
A3C	Asynchronous Advantage Actor Critic
DRL	Deep Reinforcement Learning
PID	Proportional Integral Differential
CPU	Central Processing Systems
GPU	Graphical Processing Systems
ANN	Artificial Neural Network
ReLu	Rectified Linear Unit
PPO	Proximal Policy Optimization
TRPO	Trust Region Policy Optimization

Nomenclature

m	Mass of the Aircraft
g	Acceleration due to gravity
I_{xx}	Moment of Inertia about the X axis
I_{yy}	Moment of Inertia about the Y axis
I_{zz}	Moment of Inertia about the Z axis
I_{zy}	Moment of Inertia about the X-Z plane
T_x	Thrust along the X axis in the body reference frame
T_y	Thrust along the Y axis in the body reference frame
T_z	Thrust along the Z axis in the body reference frame
X	Aerodynamic force along the X axis in the body reference frame
Y	Aerodynamic force along the X axis in the body reference frame
Z	Aerodynamic force along the X axis in the body reference frame
L	Lift Force
D	Drag Force
ℓ	Aerodynamic Moment along the X axis in the body reference frame
M	Aerodynamic Moment along the Y axis in the body reference frame
N	Aerodynamic Moment along the X axis in the body reference frame
u	Velocity along the X axis in the body reference frame
v	Velocity along the Y axis in the body reference frame
w	Velocity along the Z axis in the body reference frame
ψ	Angle between the Local North and the X axis in the body reference frame
θ	Angle between the Local Horizon and the X axis in the body reference frame
ϕ	Angle between the Local Horizon and the Y axis in the body reference frame
p	Angular velocity about the X axis in the body reference frame
q	Angular velocity about the Y axis in the body reference frame
r	Angular velocity about the Z axis in the body reference frame
δ_e	Elevator Input

δ_a Aileron Input

δ_r Rudder Input

R Reward

a Action

s State

γ Discount Factor

V Value

e_x Error in the quantity 'x'

Chapter 1

Introduction

1.1 Flight Control and Controllers

An Aircraft or UAV flying in the air, does so by managing a balance between the forces and moments of gravity, lift, drag and thrust. Apart from this there is a delicate balance between lateral and directional forces and moments that keep the aircraft stable during flight. Additionally there exists coupling between lateral and directional motions of the aircraft, all this put together makes aircraft control as well as maneuvering a hefty and challenging task.

Presently all aircrafts use PID controllers/autopilots, which are designed based on the *linearized* equations of motion of the aircraft. Thus these controllers would only be valid in the linear regime. Commercial and passenger flight operations are very well handled by these controllers but they require routine checks and parameter updations to account for the changes in aircraft body, engine and flight performance. Highly non linearized maneuvers for military applications are taken care of by making the aircraft inherently less stale and giving the pilot more authority and control over the aircraft.

In recent times UAVs have gained immense popularity due to their advantages in size and the impact they can achieve. A recent example of this is when COVID-19 vaccines were delivered to inaccessible places using drones and UAVs. Automation and incorporation of non linear governing equations of motion for the controller/autopilot will boost the capabilities of the sector multifold.

1.2 Reinforcement Learning

Reinforcement learning is a subset of Artificial intelligence and machine learning, which are techniques used to design predictors and/or classifiers and can do wonders such as speech/text recognition, computer vision-based applications, and whatnot. All these algorithms require is data which are some sort of examples and previously verified information based on which we can train the machine learning algorithms/agents.

Hence we could until now train and create agents to do tasks that are already done, with scope for generalization. After the advent of reinforcement learning, which is a framework that generates its own data and uses it to train agents computers have been able to play games such as Go, Chess, Atari games, and so on. This is possible due to the increased computational capabilities developed by humans which the reinforcement learning framework uses to simulate the games/situations and learn from them. It must be mentioned however that humans can also generate the training data but it is practically impossible and would require a lot of time to do it physically, whereas they can be easily simulated and the job gets quicker and easier.

The basic requirements of a reinforcement learning framework are a simulated environment which sets the rules of the game, a reward function that conveys the message of the effectiveness of actions taken by the RL agent to itself and an agent which learns and updates itself from the past experiences based on rewards received.

The flight control problem is also simulated as an environment with the non linear governing equations of motion of flight. A subset of reinforcement learning known as deep reinforcement learning is used where neural networks are incorporated in the algorithm which are well suited to handle non linearities since they make use of activation functions such as *tanh*, *relu*, *sigmoid* and many more.

1.3 Literature Review

Becker-Ehmck and Karl [1] have applied deep reinforcement learning to train an agent for the stabilization and control of a quad-copter. In a quad-copter the control input is the rpm for each rotor, by varying the rpm of the four rotors in different combinations, pitch roll and yaw can be achieved. Increasing the RPM of two consecutive rotors would lead to a pitch motion with that side of the vehicle moving up. Yaw stability is achieved by altering the direction of the spin of the consecutive rotors. For movements in either direction, 3

rotors have to increase their rpms and this would give a tilt to the drone and then increasing the velocity uniformly would move the drone in a particular direction. The training was hardware based with the model having no knowledge of the underlying physics. The input parameters were the measurements made using lidars, rate gyros, and a visual observation system. They have also used a reparametrization technique called the variational auto-encoder to improve the agent's learning and performance. Here the states of the quad-copter are converted into a latent space vector with fewer elements. The training data is provided to the agent employing a PID controller which flies the quad-copter for a specific duration and paths. The framework of the agent is an actor critic model and the learning happens via policy gradients. The reward function used was a simple quadratic function of the errors in p, u, v and θ.

D. Kroezen [2] has developed a reinforcement learning based controller for a conventional aircraft for certain maneuvers with a fixed trajectory. This is done using Dual Heuristic Dynamic Programming as the RL algorithm. The states are split up into longitudinal and lateral directional components as they are independent of each other's dynamics. They have used an actor critic design, with the critic predicting the derivative of the value function with respect to the states. The state vector comprises the aircraft states followed by the corresponding goal states.

Feng Liu [3] has used Deep Deterministic Policy Gradient (DDPG) algorithm to train agents to takeoff a fixed winged aircraft during cross-winds. They have used a mix of neural networks and CNNs since the data input is multi-modal with numerical inputs of center line deviation, true airspeed, wind speed, position using GPS, velocity information and visual input of the view from the main pilot's position in the cockpit. The controls are throttle, aileron, elevator and rudder. They have compared the predictions using the DDPG agent with and without visual input. They have concluded that DDPG provided good control and the objectives were met. Also, they have found out that the DDPG agent trained outperformed a PID based controller for the same application.

Emil Andreas [4] has used Asynchronous Advantage Actor Critic (A3C) methods to develop a path following controller. The vehicle that is controlled is a Manta 4-wheeled vehicle available inbuilt in the V-Rep environment. A gaussian reward function is used with the path error as the variable. Along with this another term of the rate of action is also incorporated in the reward function. The agent was trained on different straight and curved paths and has shown acceptable performance.

Koch [5] has implemented Proximal Policy Optimization (PPO), DDPG, and Trust Region Policy Optimization (TRPO) for different phases of flight for a drone. It was found that all

the three algorithms perform well with DDPG performing well for certain functions like hover policy where it was required to maintain stability and level flight. PPO on the other hand performed better for some other functions such as the landing policy. The performance of different algorithms was on the parameters of how quickly the reward function achieves its maximum value, the maximum value attained and the amount of training required. They have found that the PPO agent performs better than a PID controller on the metrics of steady state error, rise time and overshoot.

Google Deepmind [6] in 2016 introduced the deep deterministic policy gradient method which is a actor critic based neural network. This was introduced to extend DQN to the continuous action spaces. This could solve over 20 simulated physics tasks like the cartpole swing-up, mountain car, locomotion, and many more. And this was using the same model i.e. hyper-parameters, and the architecture of the network.

Google Deepmind [7] proposed a new concept for reinforcement learning which used asynchronous gradient descent for weight updates in the deep neural network control problem. This method was stable during training and the asynchronous actor critic performed the best of the four different standard RL algorithms they had implemented. This could be achieved while training for half the time on a single multi-core CPU instead of a GPU.

Tuomas [8] introduced the soft actor-critic, which along with maximizing the expected reward, it also maximizes the entropy of policy. This method achieves state-of-the-art performance on a range of continuous control benchmark tasks, outperforming prior on-policy and off-policy methods by combining off-policy updates with a stable stochastic actor-critic formulation. Furthermore, they have demonstrated that, in comparision to other off-policy algorithms, soft actor critic approach is very stable, achieving very similar performance across different random initializations.

Chapter 2

Problem Description

The dynamics of a conventional aircraft can be split up into longitudinal and lateral-directional components which are independant of each other. The three axes of motion along which the components are split are shown in Fig. 2.1. Here pitch motion corresponds to longitudinal, lateral to roll and directional to yaw.

The bird's eye view with which reinforcement learning is being brought into flight control

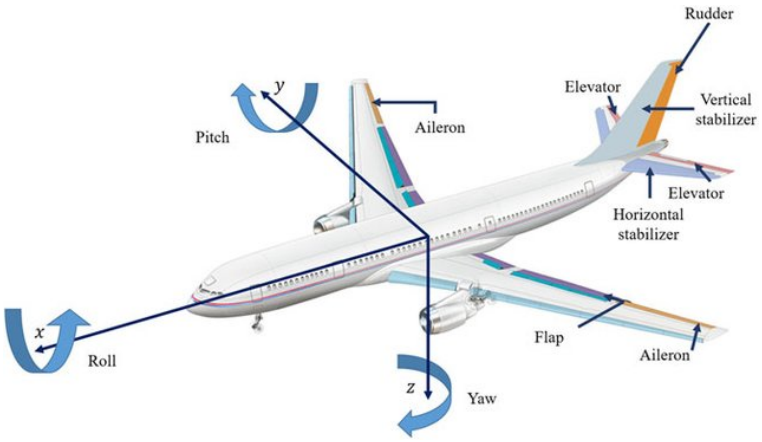

Figure 2.1: Different axes system defined for a conventional aircraft [9]

is for the following reasons and advantages.

1. Ability to factor in the non-linearities of the aircraft dynamics in the controller.
2. Based on some studies neural network based controllers have been shown to display better performance than PID controllers.
3. Developing a unified controller that can take in all the states and provide control inputs to the control surfaces. Here the 2 sets of dynamics mentioned earlier can be separated as they do not correlate.

4. When information of all states is presented to the neural networks, it gives the *big picture* to the neural network which may develop simpler and elegant correlations owing to its capabilities.

The eventual goal is thus set, the first building block for the whole controller is studied under this project.
The longitudinal mode consists of 2 modes of oscillations namely the phugoid or the long period oscillations and short period oscillations. The phugoid mode involves a slow but large-amplitude variations of air-speed, pitch angle, and altitude. The short period motion involves a rapid change in the angle of attack essentially it is rapid pitching motion of the aircraft.
The control surfaces in action for the long period mode are elevator and throttle since both pitch angle and airspeed keep changing. For the phugoid mode the control surface in action is the elevator alone.
Hence, initially, reinforcement learning is used to try and develop a controller for the pure-pitching/phugoid mode where there exists only one control surface that needs to be controlled.
The flow diagram of the controller is shown in the Fig. 2.2. The states of q and θ are fed into a reinforcement learning model, this model must be trained to predict the necessary δ_e at that particular instant. Applying this control to the system dynamics the states q and θ for the next time step can be obtained and this is repeated over a period of time. Here for the reinforcement learning model to make predictions an essential piece of information is the end state or the expectation from the controller. This information must be either incorporated into the reinforcement learning model or provided separately as need be.

2.1 Equations of Motion

The equations of motion can be derived by applying a balance of linear and angular momentum. The euler angles are defined as shown in Fig. 2.3. The equations hence obtained for longitudinal dynamics are given as Eqns. 2.1 to 2.3.

$$m\dot{u} + m(qw - rv) = T_x + X - mgsin\theta \tag{2.1}$$

$$m\dot{w} + m(pv - qu) = T_z + Z + mgcos\phi cos\theta \tag{2.2}$$

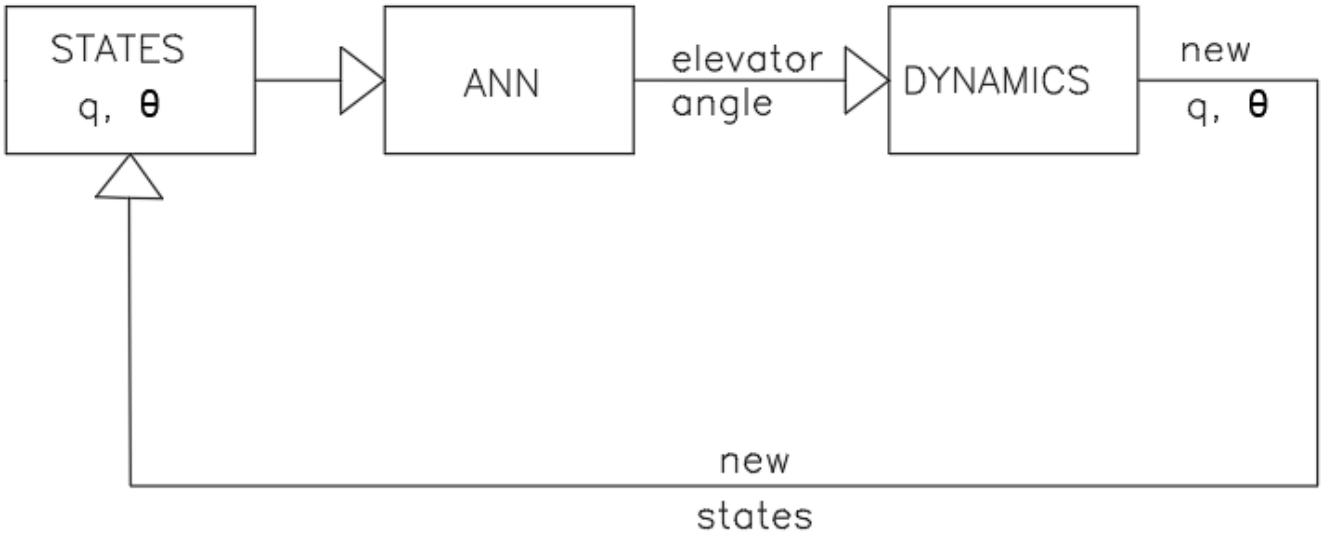

Figure 2.2: Flow diagram showing the sequence of operations for the controller to be developed

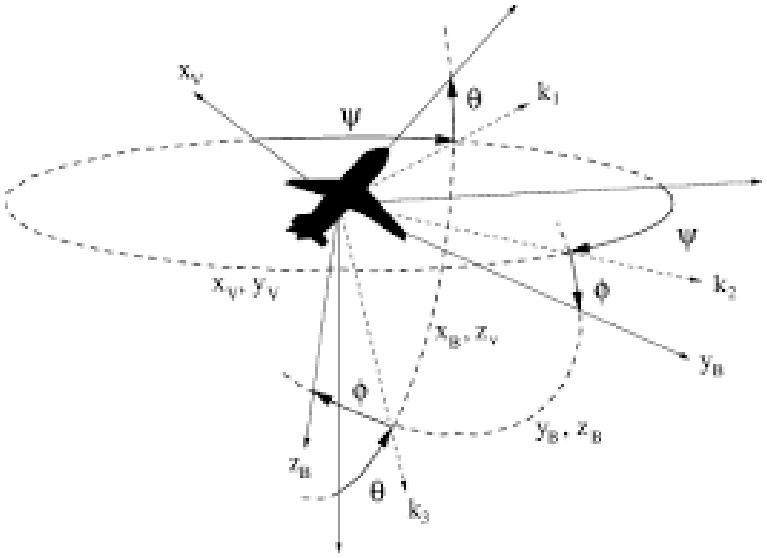

Figure 2.3: The Euler Angles defined for a conventional aircraft

$$I_{yy}\dot{q} = M + I_{xz}(r^2 - p^2) + (I_{zz} - I_{xx})pr \tag{2.3}$$

Along with these equations, we also have the attitude equations. 2.4.

$$\begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix} = \begin{bmatrix} 1 & tan\theta sin\phi & cos\phi tan\theta \\ 0 & cos\phi & -sin\phi \\ 0 & sec\theta sin\phi & sec\theta cos\phi \end{bmatrix} \begin{bmatrix} p \\ q \\ r \end{bmatrix} \tag{2.4}$$

The Aerodynamic model used for the simulation is presented as Eqns. 2.5 to 2.10.

$$L = q_\infty S C_L \tag{2.5}$$

$$D = q_\infty S C_D \tag{2.6}$$

$$M = q_\infty S \bar{c} C_m \tag{2.7}$$

$$C_L = C_{L_0} + C_{L_\alpha}\alpha + C_{L_{\delta_e}}\delta_e \tag{2.8}$$

$$C_D = C_{D_0} + KC_L^2 \tag{2.9}$$

$$C_m = C_{m_0} + C_{m_\alpha}\alpha + C_{m_{\delta_e}}\delta_e \tag{2.10}$$

2.2 Pure Pitching Motion

Pure Pitching motion is a subset of longitudinal dynamics hence v, p, r and ϕ are set to zero. The equations of motion hence get modified to the Eqns. 2.11 and 2.12. It is assumed that the thrust is constant and the only control input possible is the elevator. When some control input is given it would result in a change in the moment in the Eqn. 2.11 which would cause a change in q which would be translated into a change in θ. The reinforcement learning based agent would have to learn to provide the correct δ_e input so as to bring changes in the moment which would finally result in achieving our required θ and q.

$$\dot{q} = \frac{M}{I_{yy}} \tag{2.11}$$

$$\dot{\theta} = q \tag{2.12}$$

2.3 Aircraft Details

The mass, inertia and geometric properties of the aircraft used for simulations in this project are tabulated in the Tab. 2.1.

Table 2.2 summarizes the aerodynamic properties of the aircraft.

Table 2.1: Mass, Inertia and Geometric properties of the aircraft.

Mean Aerodynamic Chord, c	$1.211m$
Wing Span, b	$15.47m$
Aspect Ration, AR	19.9
Wing Area, S	$12.47m^2$
Mass, m	$700kg$
Moment of Inertia, Ixx	$1073kg/m^2$
Moment of Inertia, Iyy	$907kg/m^2$
Moment of Inertia, Izz	$1680kg/m^2$
Moment of Inertia, Ixz	$1144kg/m^2$

Table 2.2: Aerodynamic properties of the given aircraft.

C_{D_0} = 0.036	C_{L_0} = 0.365	C_{m_0} = 0.05
C_{D_α} = 0.061	C_{L_α} = 5.2	C_{m_α} = -0.529
e = 0.9	C_{L_q} = 17.3	C_{m_q} = -5.8
$C_{D_{\delta_e}}$ = 0.026	$C_{L_{\delta_e}}$ = 0.5	$C_{m_{\delta_e}}$ = -1.28
C_{Y_0} = 0	C_{l_0} = 0	C_{n_0} = 0
C_{Y_β} = -0.531	C_{l_β} = -0.031	C_{n_β} = 0.061
C_{Y_p} = 0.2	C_{l_p} = -0.3251	C_{n_p} = -0.015
C_{Y_r} = 0.633	C_{l_r} = 0.1836	C_{n_r} = -0.091
$C_{Y_{\delta_r}}$ = 0.15	$C_{l_{\delta_r}}$ = 0.005	$C_{n_{\delta_r}}$ = -0.049
$C_{Y_{\delta_a}}$ = 0	$C_{l_{\delta_a}}$ = -0.153	$C_{n_{\delta_a}}$ = 0

Chapter 3

Reinforcement Learning

This chapter includes basic concepts of reinforcement learning and the reinforcement learning models used throughout the project.

3.1 Basic ideas

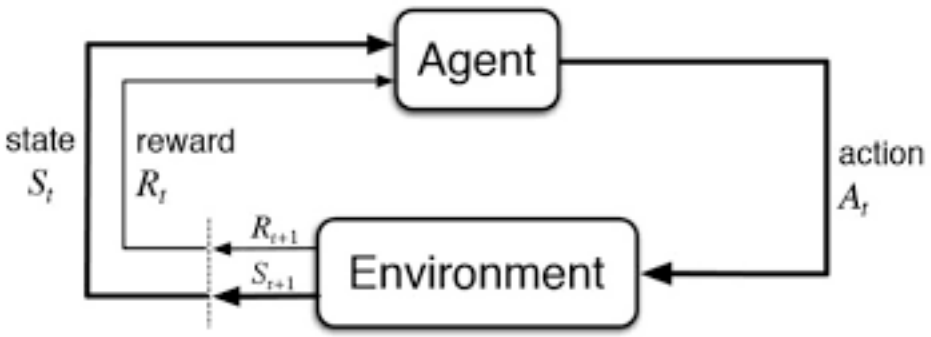

Figure 3.1: A general RL model

Reinforcement learning is a relatively new development in AI and ML. RL agents can now beat humans in games such as chess and Go. As the word suggests it is a way of learning similar to that of humans who learn by taking actions and analyzing the kind of response he/she gets for the action taken. The major components and their functions in a RL model are:

1. ENVIRONMENT: It is the virtual simulation of any system which acts similar to the real world.

2. STATE (s): Is a representation of the current situation in the environment.

3. AGENT: Agent is the one which observes the state and decides what action to take.

4. ACTION (a): It is decided by the agent and once an action is taken the agent is presented with the new state as a consequence of the action taken according to the rules of the environment and a reward that represents the correctness or some kind of appreciation or criticism about the action taken.

5. REWARD (r): It is rewarded to the agent which then corrects the way in which it calculates the action for a given state based on the reward received.

6. DISCOUNTED SUM OF REWARDS (G) = It can be given by the formula $G_t = \sum_{i=0}^{n} \gamma^i R_{t+i}$. γ is the discount factor which is how much we give importance to the future reward than to the current reward.

7. VALUE FUNCTION: Future rewards from being in a particular state and following a certain policy. It is the maximum of the Expectation of the discounted sum of rewards. It in a way represents the value of a state i.e. moving from a state of lower value to a higher value is beneficial and moving the other way round would not be beneficial to achieve the goal.

8. POLICY: It is the function that the agent uses to map states to actions. Policies can be deterministic or stochastic (probabilistic).

Based on the problem the agent can be modeled in different ways, using ML based feature extractors such as support vector machines belonging to the classical reinforcement learning. The class of reinforcement learning where neural networks are used is known as deep reinforcement learning. CNNs can be used for situations where the state is represented as an image. The problem of flight control agent can be modeled as a neural network itself.
A general flow of the reinforcement learning algorithm is

1. Randomly initialize the policy network.

2. Obtain the state from the environment and take an action. The next action is taken using the policy network and random exploration.

3. Obtain the reward and the next state.

4. Based on the obtained information make relevant changes in the policy network.

The reinforcement learning problems are modelled as a markov decision process, which assumes that the current state is a sufficient statistic of all the history of states and actions. It means that knowing the current state is enough to predict the next state and the information that could be conveyed by the history of all the previous states and actions is conveyed by the current state itself. The value function is given by the bellman equation 3.1.

$$V(s) = max_a \left(R(s,a) + \gamma \sum s' P(s,a,s') V(s') \right) \tag{3.1}$$

Here s' is the state which is reached when the agent takes the action a when at a state s. $P(s,a,s')$ is the probability of moving to the state s' from s by performing the action a. Iteratively substituting $V(s')$ we obtain $V(s)$ as the expected sum of future rewards.
The value defines the maximum of the expected sum of rewards over different actions, thus removing the max part another function for a state and action pair can be defined known as the quality of the state in Eqn. 3.2

$$Q(s,a) = R(s,a) + \gamma \sum s' P(s,a,s') V(s')$$

$$Q(s,a) = R(s,a) + \gamma P(s,a,s') max_{a'} Q(s',a') \tag{3.2}$$

We use this equation to get an estimate of the Q values for the next timestep from the Q values of the previous timestep as in Eqn. 3.3

$$Q_t(s,a) = Q_{t-1}(s,a) + \alpha \left(R(s,a) + \gamma P(s,a,s') max_{a'} Q(s',a') - Q_{t-1}(s,a) \right) \tag{3.3}$$

3.2 Deep Deterministic Policy Gradients

Maintaining a table for the Q values and updating them after each timestep is easy when the size of the problem is small. For large problems with many possible states, and actions a neural network is used to map the state to the action Q value pair. An analogy is shown in Fig. 3.2.

The input to the network would be the state and the output would be the Q value for the corresponding action node. The weights of the network are updated after a certain number of steps using the Bellman equation. This method would be well suited for problems that have a discrete action space. For problems with continuous action space, a method known as the deep deterministic policy gradient is used. Here the algorithm uses two networks

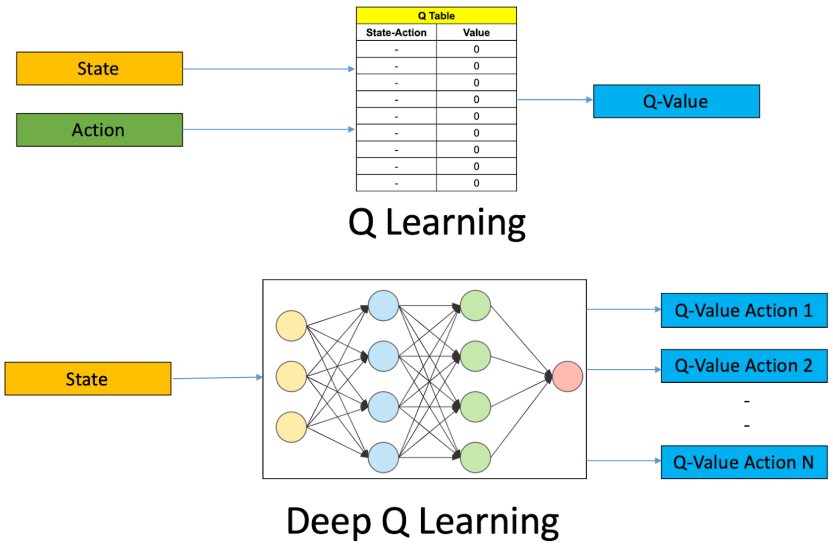

Figure 3.2: A representation of DQN

called the actor and the critic. Apart from this two more networks called the target networks for the main networks which have the same set of weights are used. After a certain fixed number of steps, the weights of the main networks are transferred to the target networks. This is done to ensure stability. The actor network takes in the state and predicts the action to be taken. The critic network takes in the state and the action predicted by the actor and produces the Q value corresponding to the state and action. Corrections are made in the networks based on the Bellman equation using stochastic gradient descent. Deep deterministic policy gradients also use what is known as an experience replay which is a sort of memory of some of the previous state action state pair. Using this memory the training happens in mini batches to ensure that the training process is smooth. The pseudocode for DDPG is given as Algorithm 3.1. A representation of the DDPG method is shown in Fig. 3.3

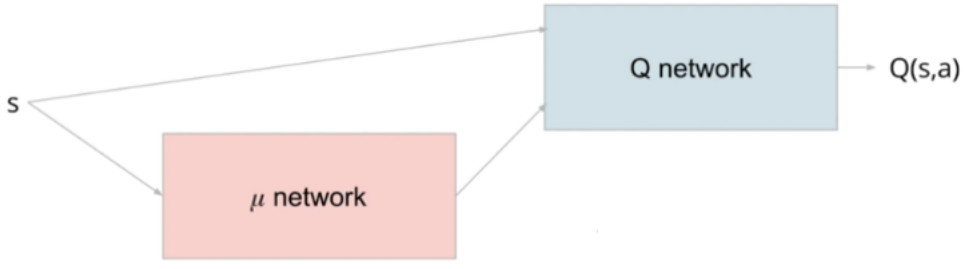

Figure 3.3: A representation of DDPG

Algorithm 3.1: DDPG

Initialize the critic network Q and actor network μ
Initialize the target critic network $\hat{Q}$ and target actor network $\hat{\mu}$
Initialize the experience replay memory D
Initialize environment
for *episode 1,M* **do**
    Initialize the random process N for action exploration
    Receive initial observation s
    for *t = 1,T* **do**
        Select action a_t using $\mu(s_t)$ + N_t according to current policy and exploration noise
        Execute a_t receive R_t and s_{t+1}
        Store the transition $(s_t, a_t, R_t, s_{t+1}, done_i)$ in D
        Sample a random mini batch from D of size H
        if $done_i$ **then**
            $y_i = R_i$
        end
        else
            $y_i = R_i + \gamma max_a \hat{Q}(s'_{i+1}, \hat{\mu}(s_{i+1}))$
        end
        Calculate the Loss $L = \frac{1}{N}\sum_{i=0}^{N-1}(Q(s_i, a_i) - y_i)^2$
        Update Q using Stochastic Gradient Descent by minimizing the Loss
        Update the Actor policy using the sampled policy gradient
        $\nabla_\mu J \approx \frac{1}{H}\sum_i \nabla_a Q|_{s=s_i, a=\mu(s_i)} \nabla_\mu \mu|_{s_i}$
        Update the target networks,
        $\hat{Q} \leftarrow \tau Q + (1-\tau)\hat{Q}$
        $\hat{\mu} \leftarrow \tau\mu + (1-\tau)\hat{mu}$
    end
end

During action selection, random noise is added to the action specified by the actor network.

3.2.0.1 Ornstein–Uhlenbeck process

In mathematics, the Ornstein–Uhlenbeck process named after Leonard Ornstein and George Eugene Uhlenbeck is stochastic. In physics this process was used as a model for velocities of large Brownian particles under the influence of friction. This process is temporally homogeneous, a gaussian and a Markov process, the only non-trivial process to have all three properties. Over time it converges towards its mean. It is sort of a random walk in continuous time.

3.3 Advantage Methods

The Reinforcement learning problems are modeled as a Markov decision process, which assumes that the current state is a sufficient statistic of all the history of states and actions. It means that knowing the current state is enough to predict the next state and the information that could be conveyed by the history of all the previous states and actions is conveyed by the current state itself. The value function is given by the Bellman Equation 3.1.
Where s' is the state which is reached when the agent takes the action a when at a state s. $P(s, a, s')$ is the probability of moving to the state s' from s by performing the action a. Iteratively substituting $V(s')$ we obtain $V(s)$ as the expected sum of future rewards.
The value defines the maximum of the expected sum of rewards over different actions, thus removing the max part another function for a state and action pair can be defined known as the quality of the state in Eqn. 3.4

$$Q(s, a) = R(s, a) + \gamma \sum s' P(s, a, s') V(s')$$

$$Q(s, a) = R(s, a) + \gamma P(s, a, s') max_{a'} Q(s', a') \tag{3.4}$$

We use this equation to get an estimate of the Q values for the next timestep from the Q values of the previous timestep as in Eqn. 3.5

$$Q_t(s, a) = Q_{t-1}(s, a) + \alpha \left(R(s, a) + \gamma P(s, a, s') max_{a'} Q(s', a') - Q_{t-1}(s, a)\right) \tag{3.5}$$

For computational purposes probabilities are converted into log probabilities. As in the above algorithm which is a basic algorithm also called the 'Reinforce' algorithm, the policy parameter are updated through Monte Carlo updates (i.e. taking random samples). Due to this the training trajectories can deviate from each other at greater levels and thus this introduces inherent high variability in log probabilities (log(policy distribution)) and cumulative reward values.
Consequently, the gradients obtained will be noisy, lead to unstable learning, and the policy can get skewed to a non optimal direction.
Another problem with policy gradients, is when the cumulative reward is zero. Policy gradient requires positive and negative responses to be able to learn, a cumulative zero won't lead to any learning.
Overall, these issues lead to the instability and slow convergence of vanilla policy gradient methods. One way to counter this of course taking mini batches of episodes and averaging the gradient over all the episodes. Another way is to use different baselines in the sense. In the equation 3.4, in place of the V(s') we use some other functions. One such functions can be the Advantage function.[34, 10]

$$A(s_t, a_t) = Q(s_t, a_t) - V(s_t) \tag{3.6}$$

The advantage function is nothing but the difference between the Quality function of a state-action pair and the Value of the particular state. This in a way calculates or tries to capture the additional Quality over the Value of the state. Thus this also decreases the magnitude of the Reward/Loss and hence we get better gradient flow and convergence. From the expression 3.6 it may seem that two neural network function approximators will be required one for Q and another one for V to calculate the advantage function but by using the expansion of Eqn. 3.4 we can write Q in terms of V and we obtain the Advantage function as

$$A(s_t, a_t) = R_{t+1} + \gamma V(s_{t+1}) - V(s_t)$$

Now we need only one neural network which will be the Critic network that will approximate the Advantage function. This method is generally known as the Advantage Actor Critic method or in short the A2C method.
Up to this point we have only one environment in which one actor is working. A variation of the A2C is known as the A3C or the Asynchronous Advantage Actor Critic method where we have multiple actors working on multiple copies of the same environment. By doing so there is more variability and also it speeds up the process of arriving at an opti-

mal control policy. A schematic of the A3C is shown in Fig. 3.4. The multiple actors are

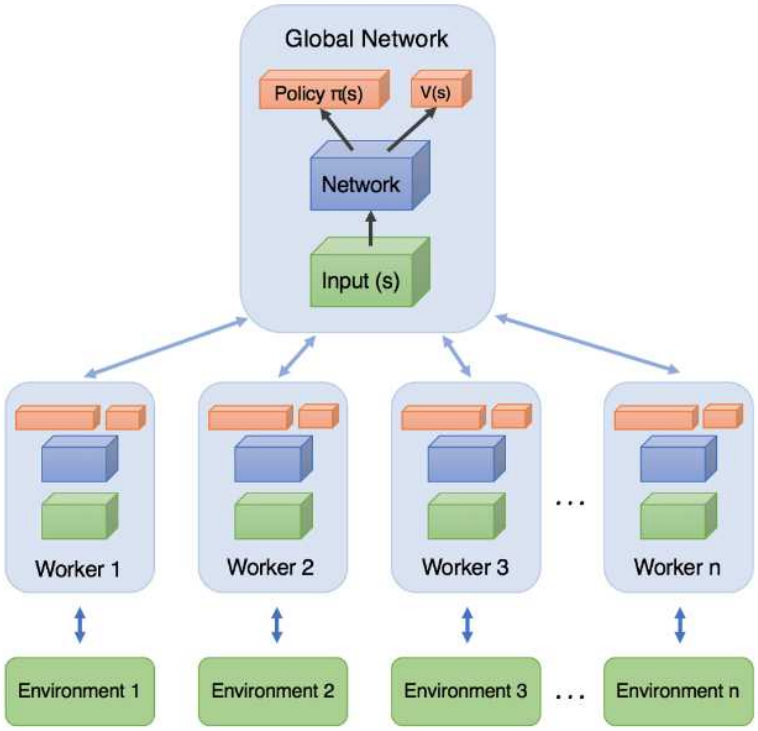

Figure 3.4: A schematic of the A3C method

referred to as workers in the context of the algorithm and the number of workers possible parallelly at any given point of time depends upon the computational capability of the system on which the Reinforcement Learning is done. On a CPU generally, 2 workers can be obtained whereas if a GPU is used as many as 60 workers can operate simultaneously.
The pseudocode for the asynchronous actor critic method is given as Algorithm 3.2 The training process followed for the A3C algorithm is

1. Initialize the global network with random weights.

2. These weights are copied to each available worker.

3. Each worker then runs episodes and collects the information of several runs.

4. The gradient information from all the workers is then used for updating the global network.[35]

Algorithm 3.2: A3C

Initialize the critic and the actor network with random weights
for *episode 1,M* **do**
    Download weights from each headquarters to the Actor
    for *each Actor* **do**
        Initialize the environment with state s_0
        for *t 1,K* **do**
            Select action a_t, from the actor network
            Execute action a_t, observe reward r_t, and get the state s_{t+1}.
            Update the actor network parameters
        end
        Update the critic network parameters
    end
    Upload the weight of each actor network to headquarters
end

Soft Actor Critic Method

Soft actor-critic (SAC) can be described as an off-policy model-free deep reinforcement learning algorithm. The major features of this method are summarized below.

1. SAC makes use of an Experience replay or a replay buffer which can store transitions of (s,a,s',r) and uses it to learn optimal policies.

2. This method can be visualized as applying DQN for continuous spaces but also along with the Value of a state the Maximization of the Entropy is also done.

3. The Maximization of entropy is the idea that the agent explores more actions and thus it doesn't restrict itself very early to only a subset of the action space.

4. Unlike DDPG the action chosen in SAC is a stochastic one, meaning that for each action, there exist 2 variables that are predicted by the neural network which are the mean and variance of the particular action variable. Later, from this distribution, an action is sampled.

5. Due to the use of the maximization of entropy along with the expected summation of rewards, this method is more robust than some other methods such as A3C, DDPG and PPO which are on policy methods.

6. This method being a off policy method uses target networks and also uses soft updates in place of hard updates.

The loss function which is used is shown as Exp. 3.7.

$$J_\pi = E_\pi \left[\sum_t r(s_t, a_t) - \alpha \cdot log(\pi(a_t|s_t)) \right] \tag{3.7}$$

Here α is a constant often called the temperature parameter associated with the entropy which decided the level of importance to be given to the entropy. Here the π refers to the policy from which the action is chosen. The second term in the Exp. 2.11 is the entropy term and it is called so due to its similarity to the thermodynamic entropy equation proposed by Ludwig Boltzmann.
The pseudocode for the implementation of soft actor critic is given as Algorithm 3.3.

Algorithm 3.3: SAC

Initialize policy parameters θ, Q function parameters ϕ_1 and ϕ_2, empty the replay buffer D

Set target parameters equal to the main parameters $\phi_{targ,1} \leftarrow \phi_1$, $\phi_{targ,2} \leftarrow \phi_2$

Observe state s and select action a $\pi_\theta(\cdot|s)$ Execute a in the environment and observe the next state s', reward r, done signal d

Store (s,a,r,s',d) in the replay buffer D

If s' is terminal reset environment state

if *it's time to update* **then**

  for *j in range* **do**

    Randomly sample a batch of transitions B from D

    Compute targets for the Q functions

$$y(r, s', d) = r + \gamma(1 - d)\left(minQ_{\phi_{targ,i}}(s', \bar{a}') - \alpha log\pi_\theta(\bar{a}'|s')\right), \bar{a}' \; \pi_\theta(\cdot|s')$$

    Update Q functions by one step of gradient ascent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum (Q_{\phi_i}(s, a) - y(r, s', d))^2$$

    Update policy by one step of gradient ascent using

$$\nabla_{\phi_i} \frac{1}{|B|} \sum \left(minQ_{\phi_i}(s, a_\theta\bar{(s)}) - \alpha log\pi_\theta(\bar{a_\theta}(s)|s)\right)$$

    where $\bar{a_\theta}$ is a sample from $\pi_\theta(\cdot|s)$ which is differentiable wrt θ via the reparametrization trick.

    Update target networks with

$$\phi_{targ,i} \leftarrow \rho\phi_{targ,i} + (1 - \rho)\phi_i$$

  end

end

until convergence

Chapter 4

Training Setup

4.1 Device and platforms

The different reinforcement learning models were trained on 2 different platforms

1. Google Colab GPU/CPU engine

2. Intel Xeon Silver 4114 Processor (10 Core, 2.20 GHz, 13.75 MB L3 Cache), Tesla V100 PCI-E GPU card

Generally, for all deep learning applications, GPUs are preferred since, with more memory and computing speed available GPUs can complete parallel operations with increased speed. Hence even large amounts of data can be analyzed easily within feasible time frames.
But for deep reinforcement learning, there is a restriction due to the non existence or very minimal parallel tasks existing in the reinforcement learning pipeline. Thus there is almost no difference in the process speeds for CPU and GPU execution. This necessitated the use of multiple devices and multiple programs.
Usage of a GPU makes a difference only when the network size is large thus requiring more memory. The network sizes used in this project can be put into the category of small nets and hence CPU implementation is sufficient and appropriate.

4.2 Hyper parameters

4.2.1 Learning rate

The learning rate is a constant used for the stochastic gradient descent method. It decides the factor by which we need to move in the direction of the gradient. Typical orders of the lr for deep learning applications are $1E-4$. In the project different learning rates such as $1E-3$, $3E-4$, $1E-4$ are experimented with. The best of the results were obtained by using $1E-4$.

4.2.2 Discount factor

The discount factor is a constant between 0 and 1 which decides the importance given to the future rewards as compared to the reward of the current step. This is used in the Bellman equation and influences the calculation of the Q function and the V function. Based on different literature surveyed the values of γ used for the experiments are 0.44, 0.8, 0.9, and 0.99.

4.2.3 Timestep size

The step size is fixed in the whole project and is taken to be 0.02s i.e. the controller sends control input and changes the elevator angle every 0.02 seconds.

4.3 Reward Functions

The reward function is a very important part of the reinforcement learning algorithm. Out of the two ways of providing information to the agent, this is one of the ways. The other is being the states that are observed as given as inputs to the actor/agent.
The reward must do the following things primarily, take up high values when the current state is favorable and when the agent takes appropriate action. It must take up low values when the current state or the action taken by the agent is unfavorable.
Also there can exist cases where a certain action or state may not be as such favorable but the direction it is moving will lead to favorable conditions, in such cases also the reward must encourage the agent to move in the same direction. Hence the gradient of the reward

function is also an important part to be taken care of while designing the reward function apart from the magnitude itself.
Also during the project, it was found that if the reward values are too high or too low then the incremental bonus rewards or decrements in the value don't make any difference in the training.
Based on the following pointers the reward function has been designed. The different reward blocks used are explained in the following few sections.

4.3.1 Quadratic Variation

The quadratic variation can be implemented in the following way,

$$R = -(x - x_{goal})^2$$

Here x can be either θ, q, $\dot{q}$, or any other relevant variables of the problem. The graph of this function is shown in Fig. 4.1. This is in accordance to the requirement that the function value is highest at the best possible state for the environment to be in and it must decrease as we move away from that particular point. This could also be satisfied by a linear function, $R = |x - x_{goal}|$. But here the second condition about gradient of the reward function won't be followed. For instance the increment in reward for a fixed improvement in x near x_{goal} and far away from it would be the same. Thus by providing the quadratic function we get larger increments in rewards when x moves towards x_{goal} far away from x_{goal}. Based on the range of values x can take during the episodes the above reward can even be scaled and shifted suitably.

Implementing the simple quadratic has one drawback that as x moves towards x_{goal} the relative increment in the reward becomes smaller and smaller. To tackle this another reward block can be added as follows:

$$R = dx_{set}^2 - (x - x_{goal})^2$$

This reward must be added to the original reward only when $dx_{set} > (x - x_{goal})^2$. This value can also be scaled and offset accordingly as required by the problem.

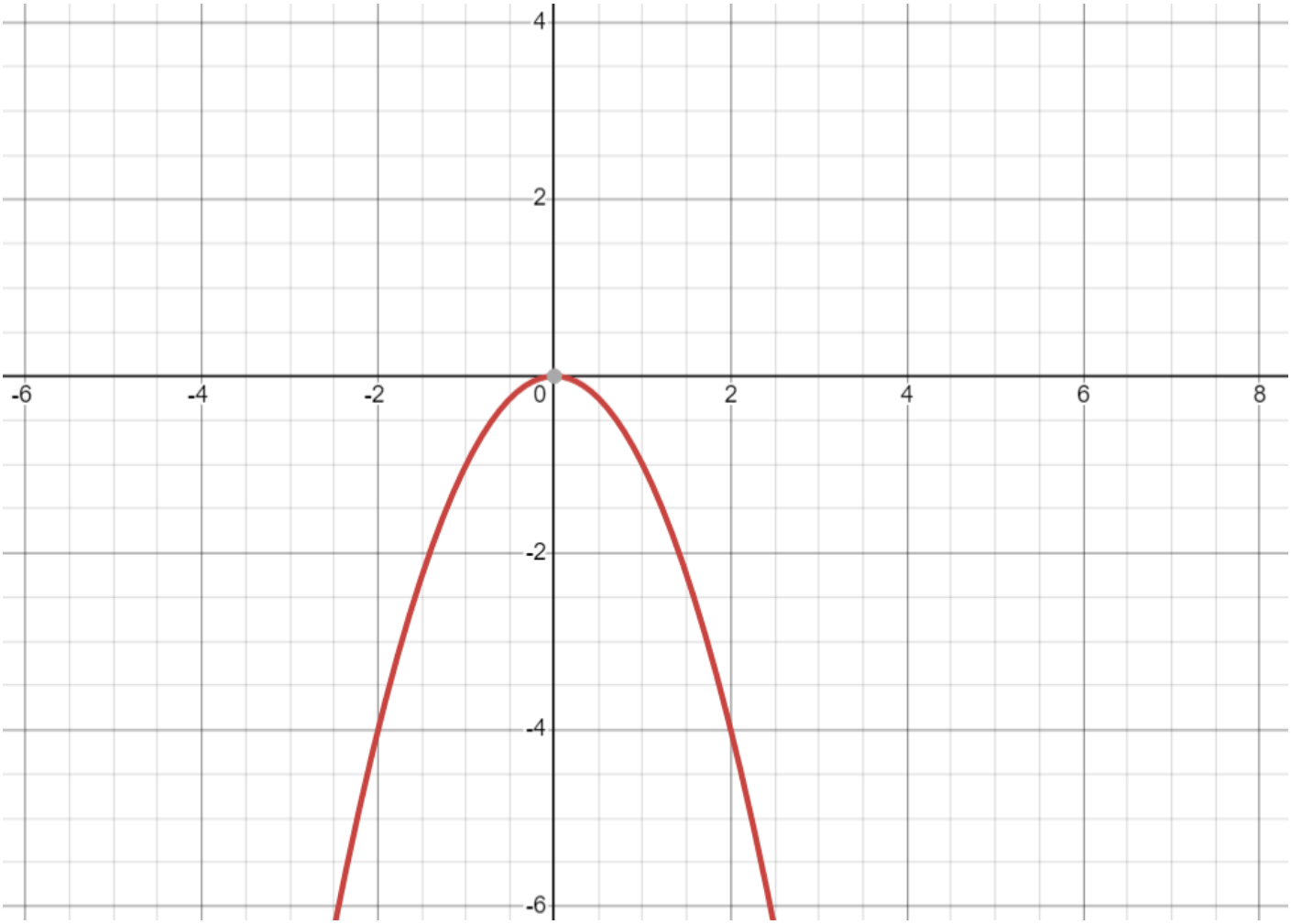

Figure 4.1: Quadratic reward function variation

4.3.2 Elevator reward function

The agent is free to choose any control input value at every time step and there is no restriction provided in the environment. But in reality, huge jumps in the elevator angle is not possible physically, to include this as a reward function

$$R = -|\Delta\delta_e|$$

can be used. Here the difference is taken between the actions taken by the actor/agent in two consecutive time steps. This is one way how large jumps in the elevator input can be discouraged. As in the quadratic case here too scaling and shifting can be done as the need arises.

4.3.3 Ideas from PID controllers

Some ideas from PID can also be borrowed and create meaningful reward functions. The quadratic reward function takes care of the gradient (derivative part) and the proportional part in some sense, but one more element which is helps mitigate any steady state errors is the integral part. This can be converted to a reward function in the following way,

$$R = -\frac{1}{2}(e_x + e_{x-1}) \cdot dt$$

Here e_x is the error in the variable x.

Apart from this the reward function:

$$R = -|PID - \delta_e|$$

can also be used, where PID is an already tuned controller for the required function.
These reward functions can also be scaled and shifted to acquire the required range of values for the reward function.

4.3.4 Gaussian

Unlike the quadratic nature reward function where increment in rewards is high when x is far away from the target variable, and low as it gets closer. A gaussian reward function,

$$R = exp(-e_x^2)$$

has the reverse effect where the increment in rewards is low when x is far away from the target variable, and high as it gets closer. As and when required this function too can be scaled and shifted, also the values of mean and standard deviation can be incorporated. Here e_x is the error in the variable x.

4.4 Neural Network Architecture

The domain of reinforcement learning used to develop pitch controllers is deep reinforcement learning where neural networks are used. The neural network has different parameters that can be varied. In this project, ReLu activation functions are used in all layers except the final layer where linear activation is used.
The activation functions are given in Fig. 4.2 The number of hidden layers is also fixed to

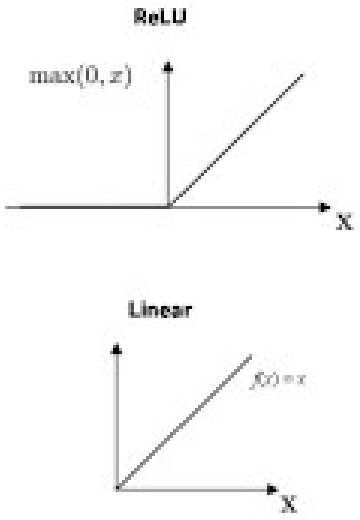

Figure 4.2: Linear and ReLu activation functions

2 based on previous studies. For the actor, two sets of values for the number of neurons are experimented with 32 and 400. For the critic network, the two values of neurons used are 32 and 300. Thus the configuration of the two neural nets used is as follows:

- First Net, actor and critic
  Layer 1 - 32 neurons - ReLu activation
  Layer 2 - 32 neurons - ReLu activation
  Output Layer - 1 neuron - Linear activation

- Second Net, actor
  Layer 1 - 400 neurons - ReLu activation
  Layer 2 - 400 neurons - ReLu activation
  Output Layer - 1 neuron - Linear activation

- Second Net, critic
  Layer 1 - 300 neurons - ReLu activation
  Layer 2 - 300 neurons - ReLu activation
  Output Layer - 1 neuron - Linear activation

Here the output layer has 2 neurons. In the case of the SAC method where reparametrization is done in order to obtain a differentiable function of the policy. The reparametrization has two parameters namely the mean μ and standard deviation σ.

4.5 Training methodology

The course of training a reinforcement learning algorithm involves running a large number of episodes and from the rewards that are obtained the learning happens. For the problem of a pitch controller, the following two training procedures can be adopted.

1. Fix the θ_{goal} to a constant value and train the reinforcement learning model.
2. Vary the value of θ_{goal} for different episodes.

Both these approaches have been tried and implemented in the experiments.

4.6 State Vector Initialization

In the process of training RL models, they require an initial start state from which actions are taken and subsequent states are observed. $\Delta\theta$ is initialized randomly between -5° and 5°. q is initialized randomly between -0.01 rad/s and 0.01 rad/s, doing so provides robustness to the algorithm. For cases where $\dot{q}$ is used as a state, it is initialized to zero. The episode is terminated in cases where, $|\theta|$ goes beyond 10° and the elevator can vary between -15° and 15°.

Chapter 5

Results and Discussions

The pure pitching problem which is to be solved and a controller developed using reinforcement learning is a tracking problem in θ and a reference problem for q maintained to a value of zero. The states present in the governing equations i.e. q and θ are included in the state vector. Since θ is a tracking problem it would be more appropriate to include $\Delta\theta$ rather than θ. In addition, to provide information regarding the goal angle to be achieved a third state is included in the state vector, θ_{goal}. Thus the state vector $(q, \Delta\theta, \theta_{goal})$ is used to train the reinforcement learning agent. In the training methodologies discussed in the previous chapter, ideally the first methodology must be able to train an agent which can track any given reference angle since we use the error in the reward function and so it must always go to zero in the most ideal case irrespective of the θ_{goal}. But from the experiments, it has been found that this is not the case and there exist steady state errors for the angles for which no training is done. Thus the second methodology is employed throughout.

5.1 Results from the first neural network configuration

5.1.1 Deep Deterministic Policy Gradient

The results obtained for the DDPG agent, for different discount factors of 0.44, 0.8, and 0.9 are shown in Figs. 5.1, 5.2, and 5.3. The reward function blocks used here are the quadratic variations with and without the conditional bonus reward for $\Delta\theta < 1^{\circ}$. Also the quadratic term for $\Delta\theta$ is scaled by a factor to give more weightage to it than q. The reward function blocks are $R = -20\Delta\theta^2 - q^2$ and for $\Delta\theta < 1^{\circ}$ $R = R + 100 \cdot (1^{\circ^2} - \Delta\theta^2)$
As it can be seen in the plots there exists a high frequency oscillations in the control input of the elevator and this also is reflected in the values of q. Also the values to which the θ seems to converge are not the θ_{goal} set for the episode.

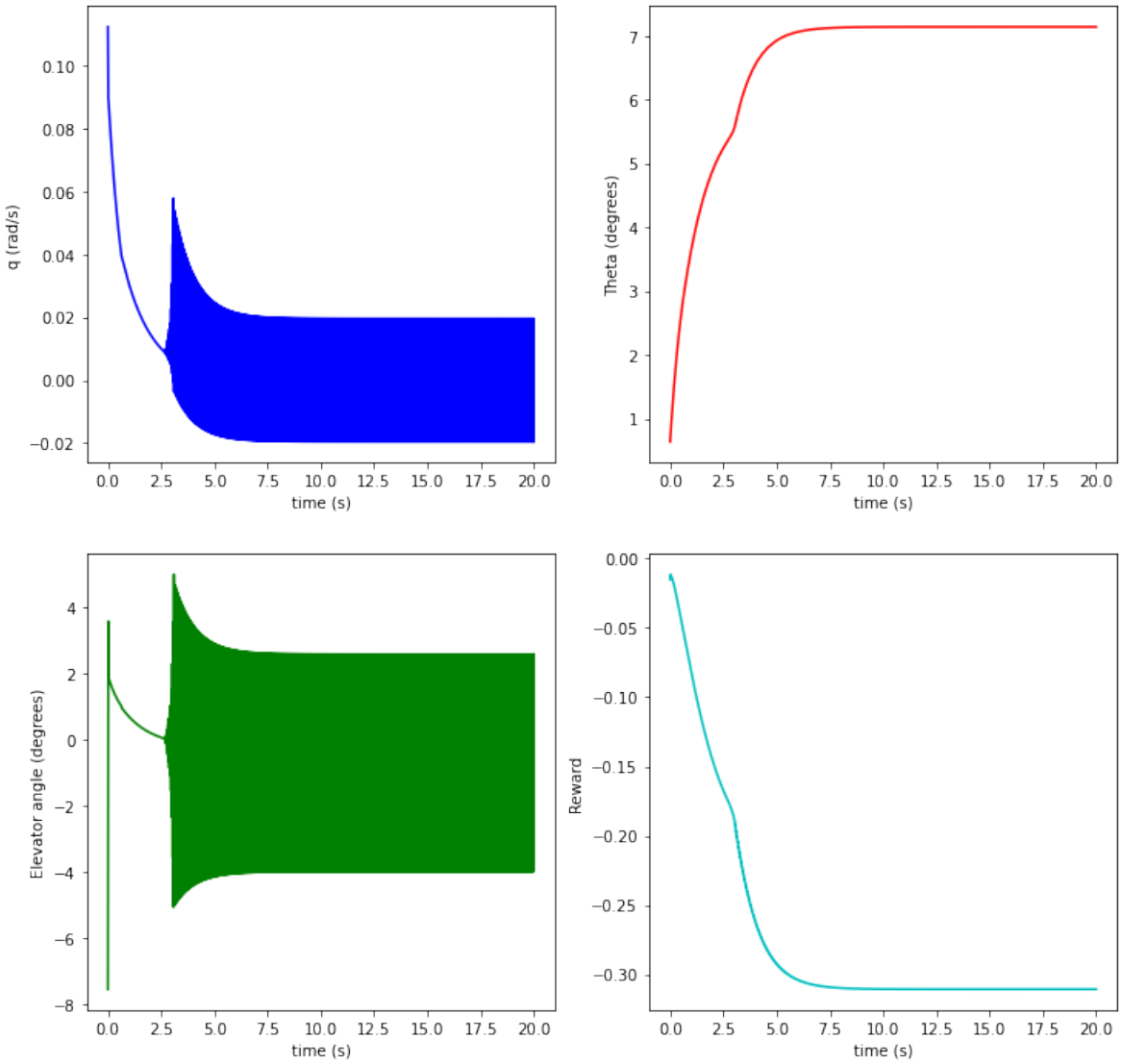

Figure 5.1: Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of $4°$

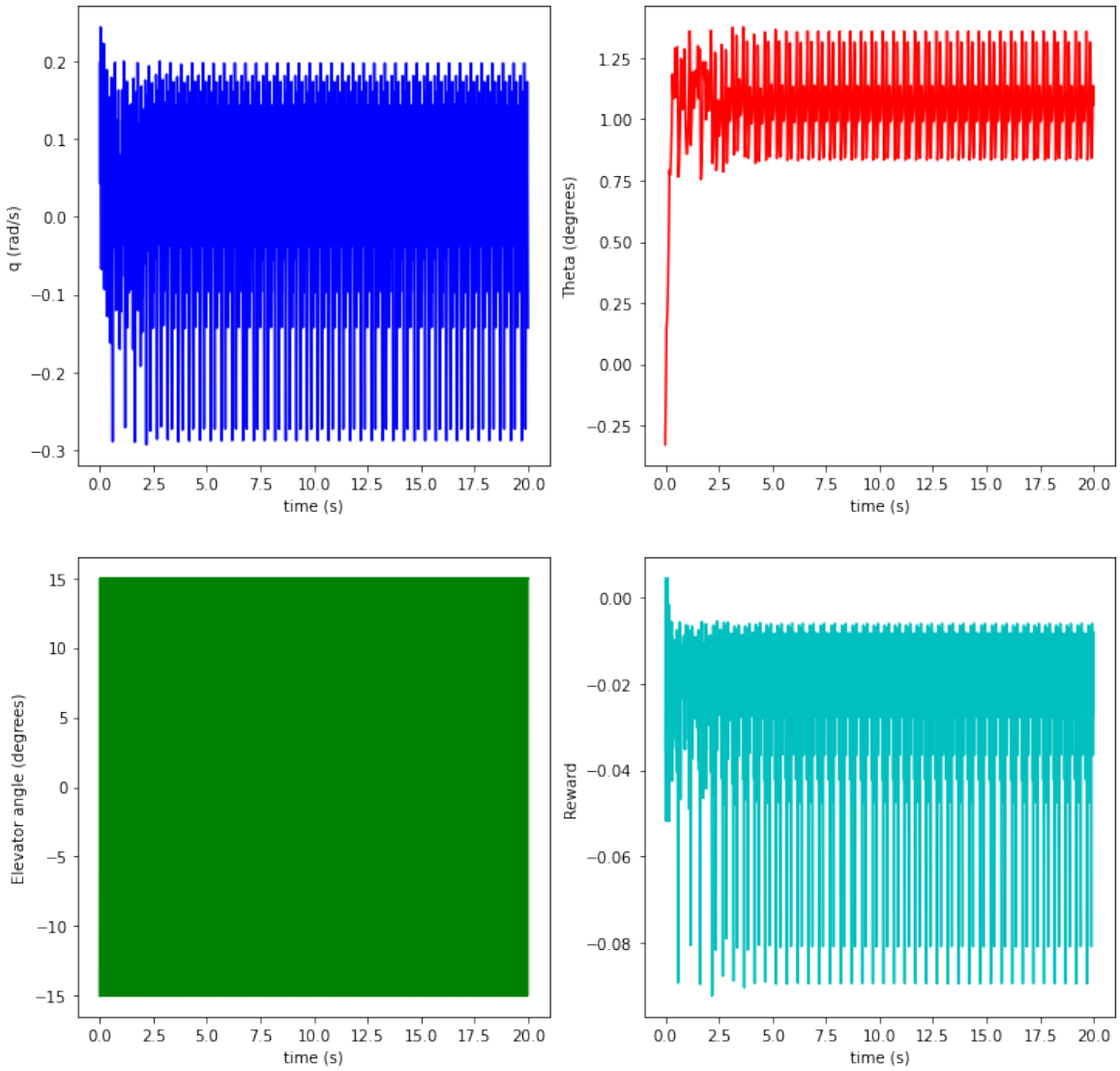

Figure 5.2: Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 0°

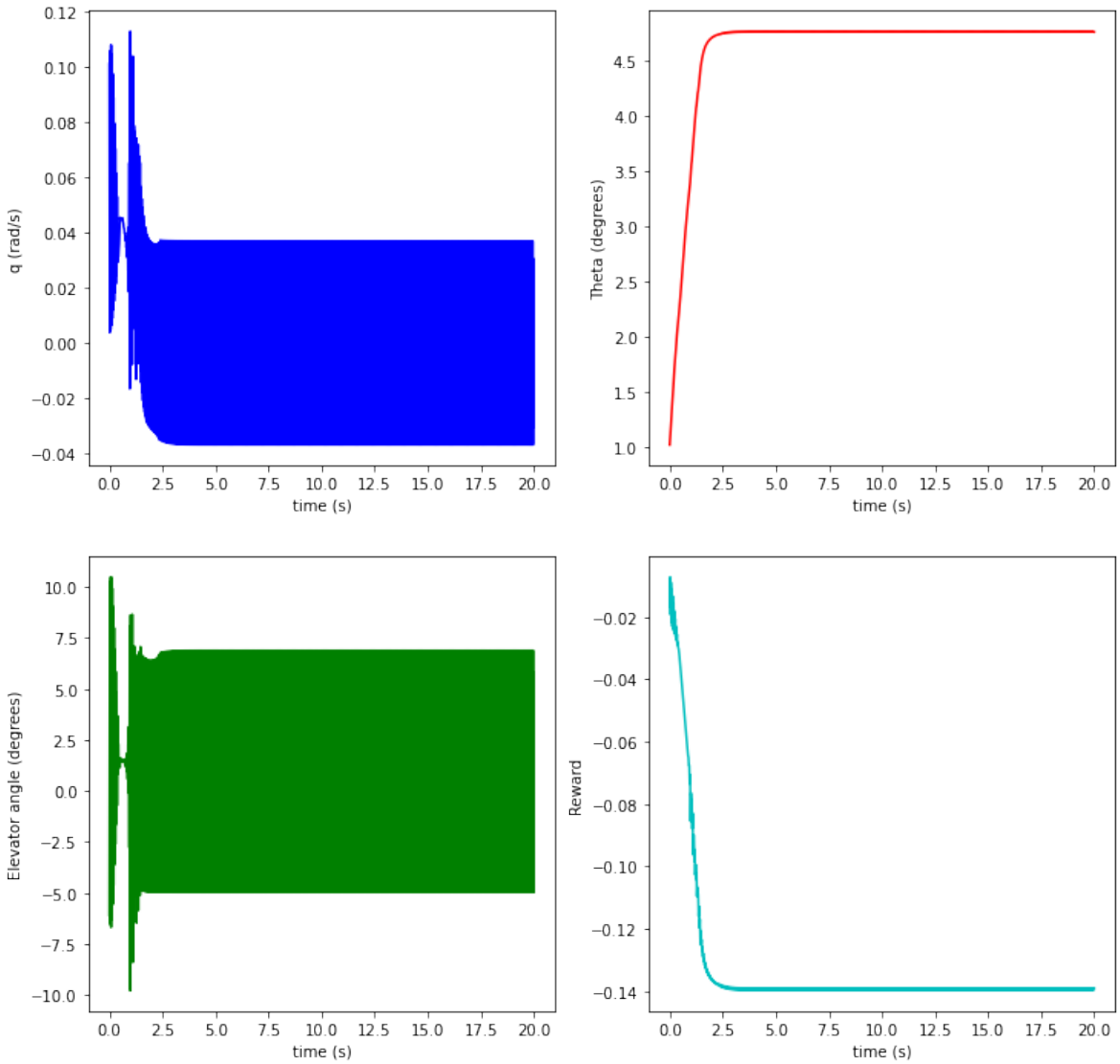

Figure 5.3: Results obtained for a DDPG agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of 3°

The possible reasons for the non-convergence of the DDPG algorithm in the case are:

1. More training is required for the DDPG to converge and provide an optimal solution.
2. The DDPG algorithm somehow doesn't suit this problem.
3. The input state required to make the decision are not good enough and some more information is needed.

The first point is explored by training the DDPG agent for more number of episodes but there is no improvement in the results.
Thus the second point is explored next searching for different reinforcement learning algorithms from literature. Two algorithms appear to suit this problem based on the literature available which are the asynchronous advantage actor critic method and the soft actor critic method.

5.1.2 Asynchronous Advantage Actor Critic method

The results obtained for an A3C agent with a discount factor of 0.99 and a reward function

$$R = -20 \cdot \Delta\theta^2 - q^2$$

If $\Delta\theta < 1^\circ$,

$$R = R + 100 \cdot (1^{\circ^2} - \Delta\theta^2)$$

are shown in Fig. 5.4 and 5.5.
From the plots even though we see an improvement in the variation of θ and q, there still exists oscillations in the elevator input from about 3 degrees to -3 degrees.

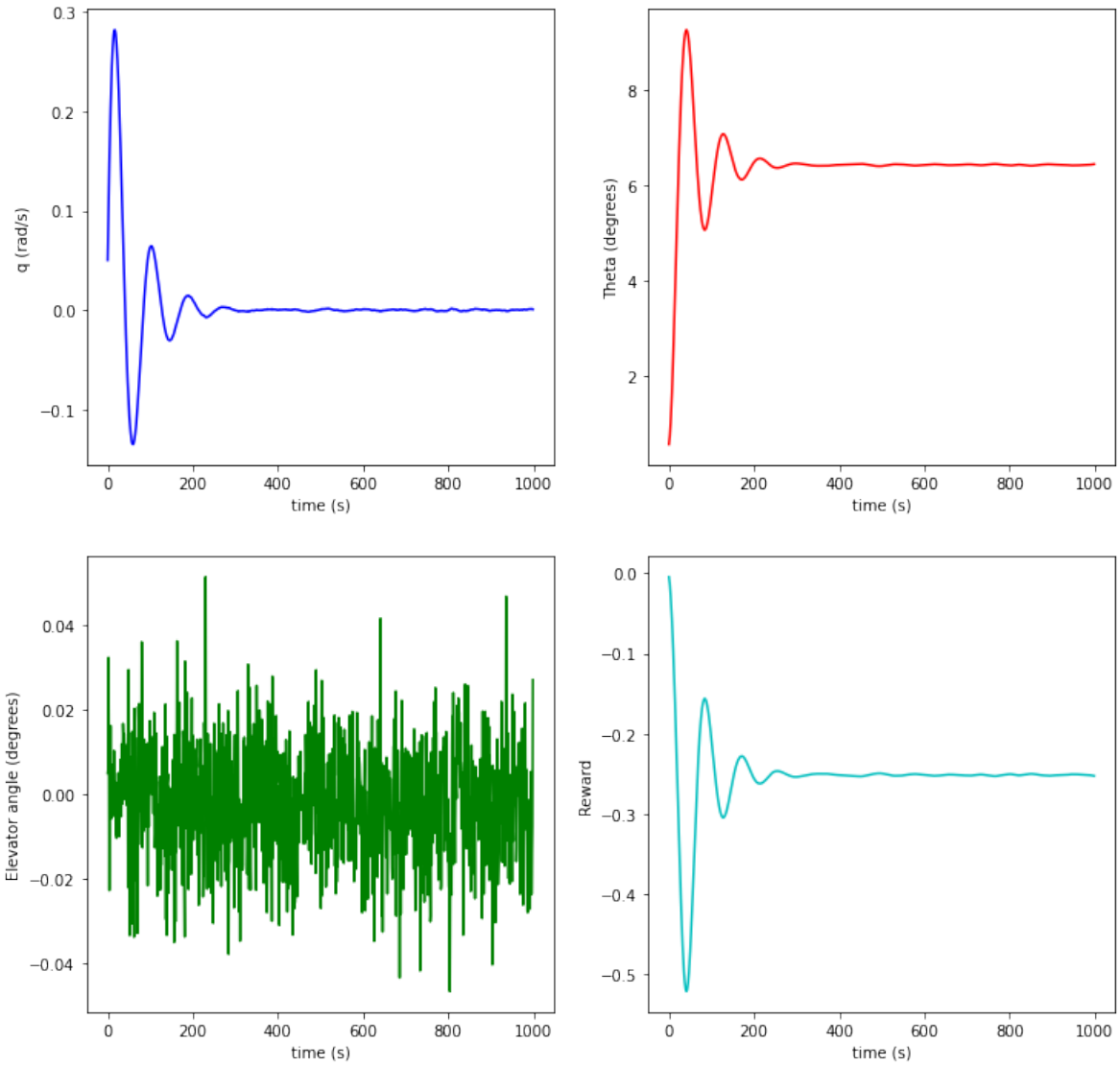

Figure 5.4: Results obtained for A3C agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 4°

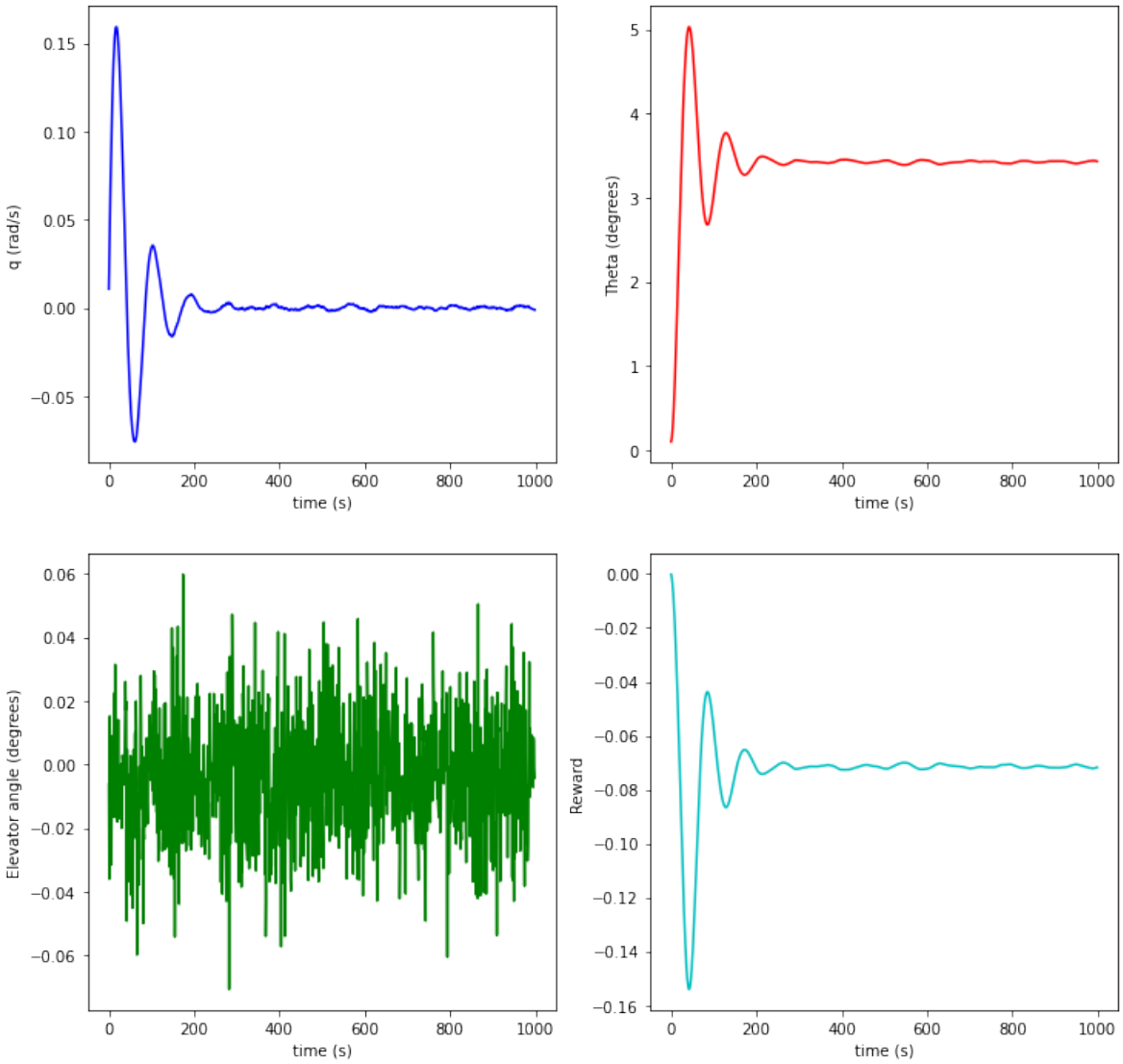

Figure 5.5: Results obtained for A3C agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 2°

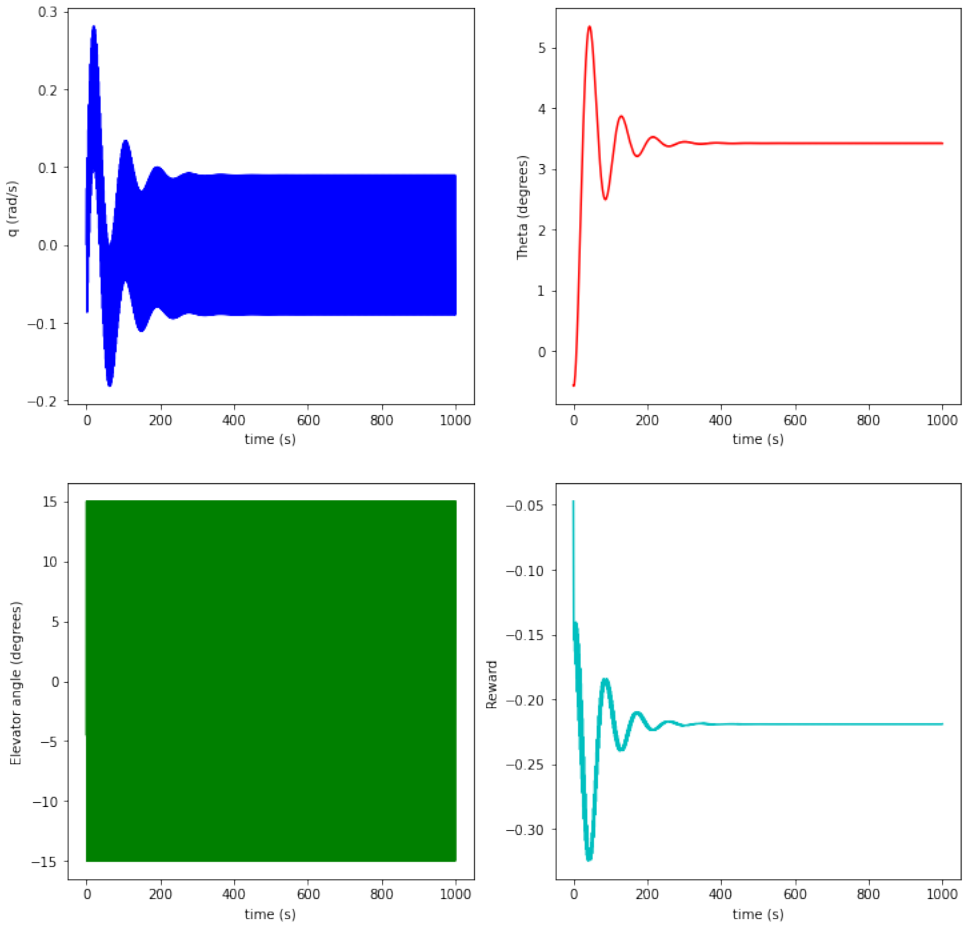

Figure 5.6: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology for a θ_{goal} of $3°$

5.1.3 Soft Actor Critic method

Another promising algorithm found in literature is the soft actor critic method. Results obtained for a SAC agent are shown in Figs. 5.6, 5.7, and 5.8. In plot 5.6 it can be seen that θ shows good convergence but the oscillations for the elevator are very high. Hence the reward function is modified to

$$R = -20 \cdot \Delta\theta^2 - q^2 - |\Delta\delta_e|$$

If $\Delta\theta < 1°$,

$$R = R + 100 \cdot (1^{\circ^2} - \Delta\theta^2)$$

By doing so the high frequency oscillations have decreased but not disappeared completely.

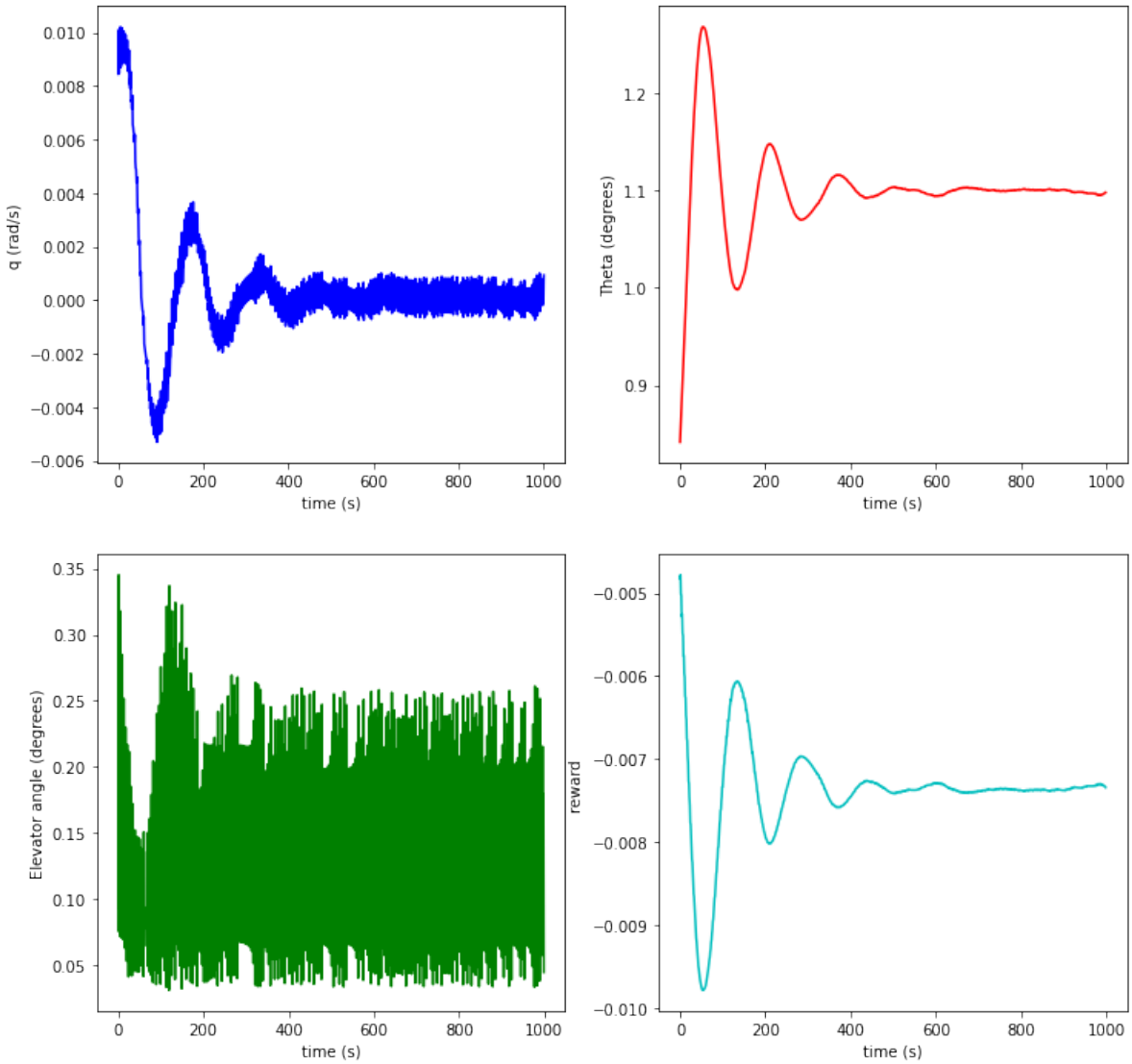

Figure 5.7: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 1°

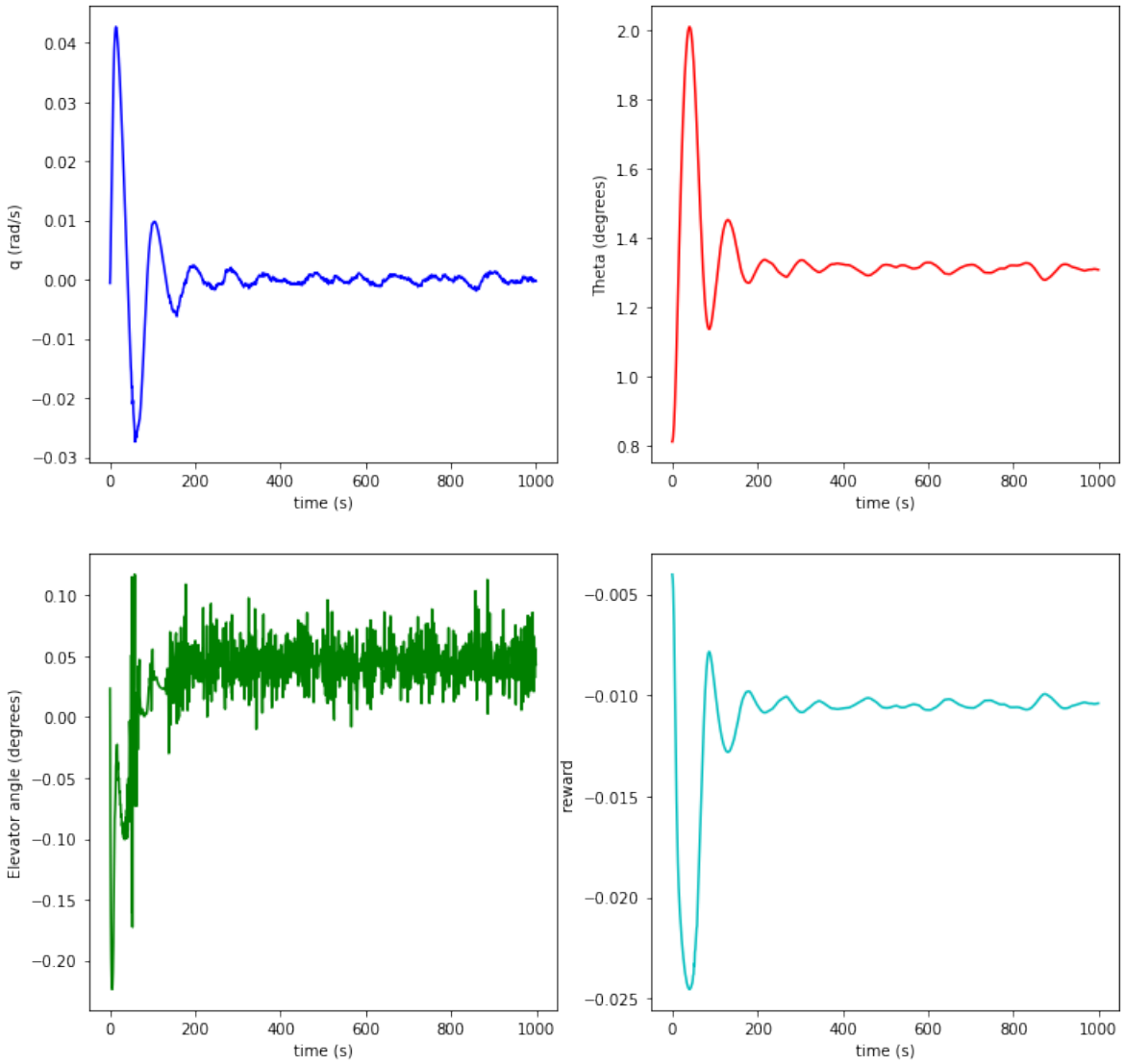

Figure 5.8: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \theta_{goal})$ using the second training methodology with the elevator angle shown in radians for a θ_{goal} of 1°

5.2 Change of States

The next approach adopted to try and solve the problem is looking into the third reason for possible non-convergence of the problem. Considering the state vector the following can be noted
The governing equations for the pure pitching system that is being solved here are

$$\dot{q} = \frac{M}{I_{yy}}$$

$$\dot{\theta} = q$$

The moment can be expressed as,

$$M = C_m \bar{c} Q_\infty$$

The coefficient C_m is

$$C_m = C_{m0} + C_{m\alpha}\alpha + C_{m\delta_e}\delta_e$$

From these above equations, it can be seen that the θ_{goal} doesn't appear anywhere in the governing equations. The only influence it has is on the control input of δ_e which is kind of a very complex dependence. But from the expressions, $\dot{q}$ has a more direct influence and relationship with M and consequently δ_e thus using the state vector (q, $\Delta\theta$, $\dot{q}$) could improve the results.
Up until now the magnitude of the values of the reward were numbers between 0 and 1 since the values of $\Delta\theta$ are taken in radians. Thus improvement in values when the agent performs better may not be evident as such. This requires scaling up the terms in the reward function. The typical values of the different terms in the reward function

$$R = -\Delta\theta^2 - q^2 - \dot{q}^2$$

are tabulated in Tab. 5.1. Since $\dot{q}$ is added to the state vector it is also added to the reward function to provide a more stringent condition.
By implementing the change of states with the reward function of

$$R = -10000 \cdot \Delta\theta^2 - 10000 \cdot q^2 - \dot{q}^2$$

Table 5.1: Typical values and Scaling factors for the terms in the new reward function

	q	$\Delta\theta$	$\dot{q}$
Typical Value	0.01	0.017	6.25
Scaling factor	10000	10000	1

If $\Delta\theta < 1^{\circ}$,

$$R = R + 10 \cdot (1^{\circ^2} - \Delta\theta^2)$$

and a discount factor of γ = 0.9 the plots 5.9 to 5.13.

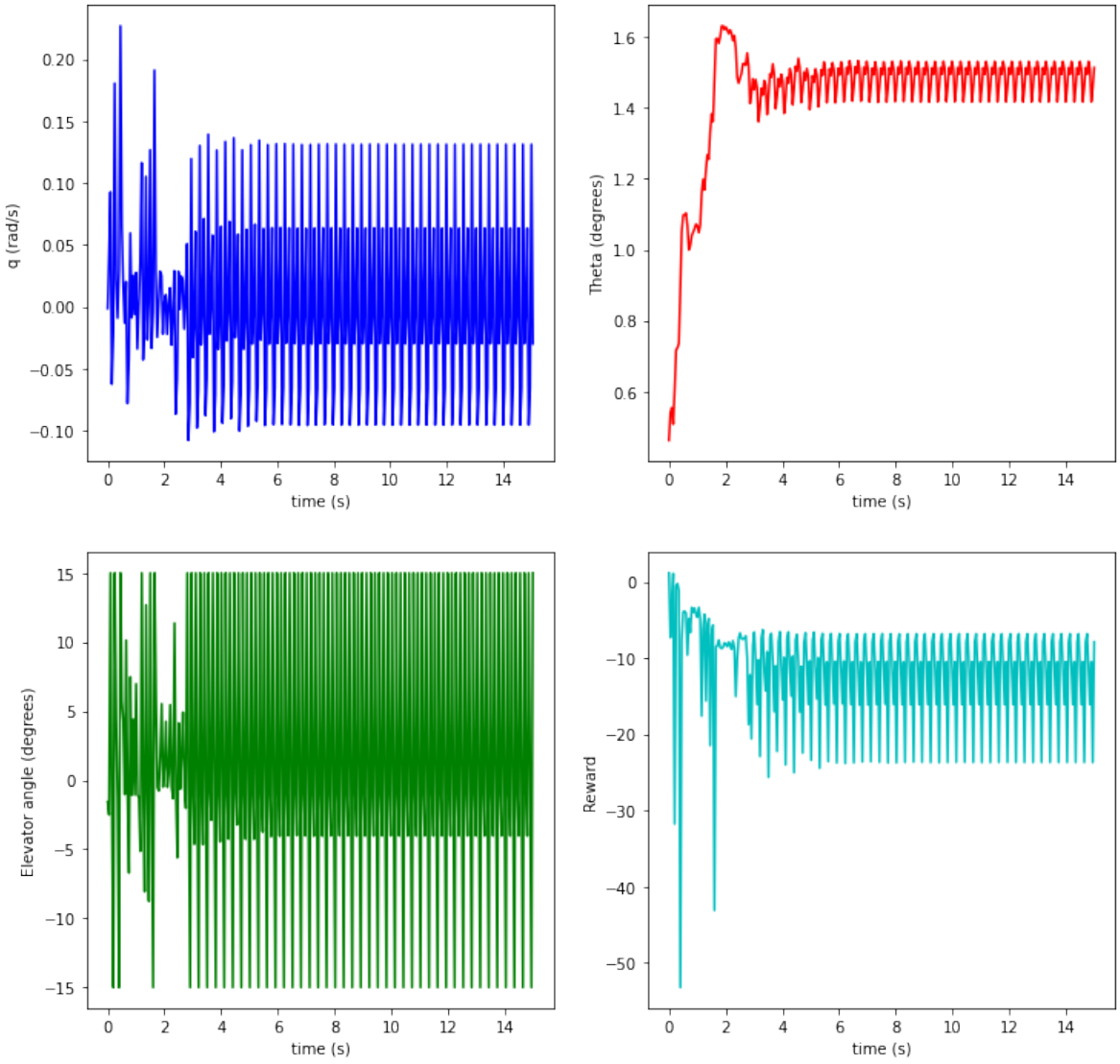

Figure 5.9: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 0^{\circ}$

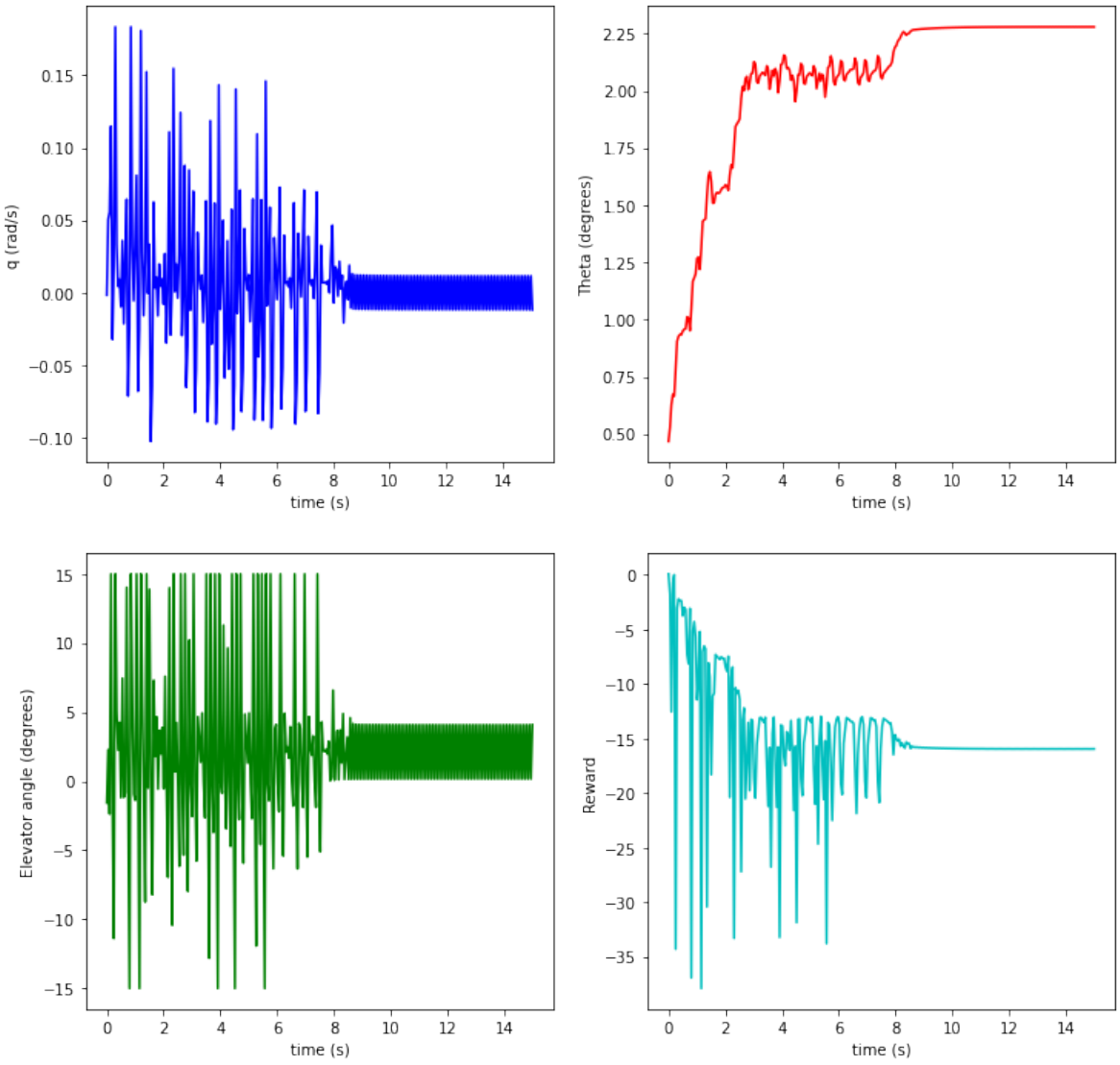

Figure 5.10: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 2°$

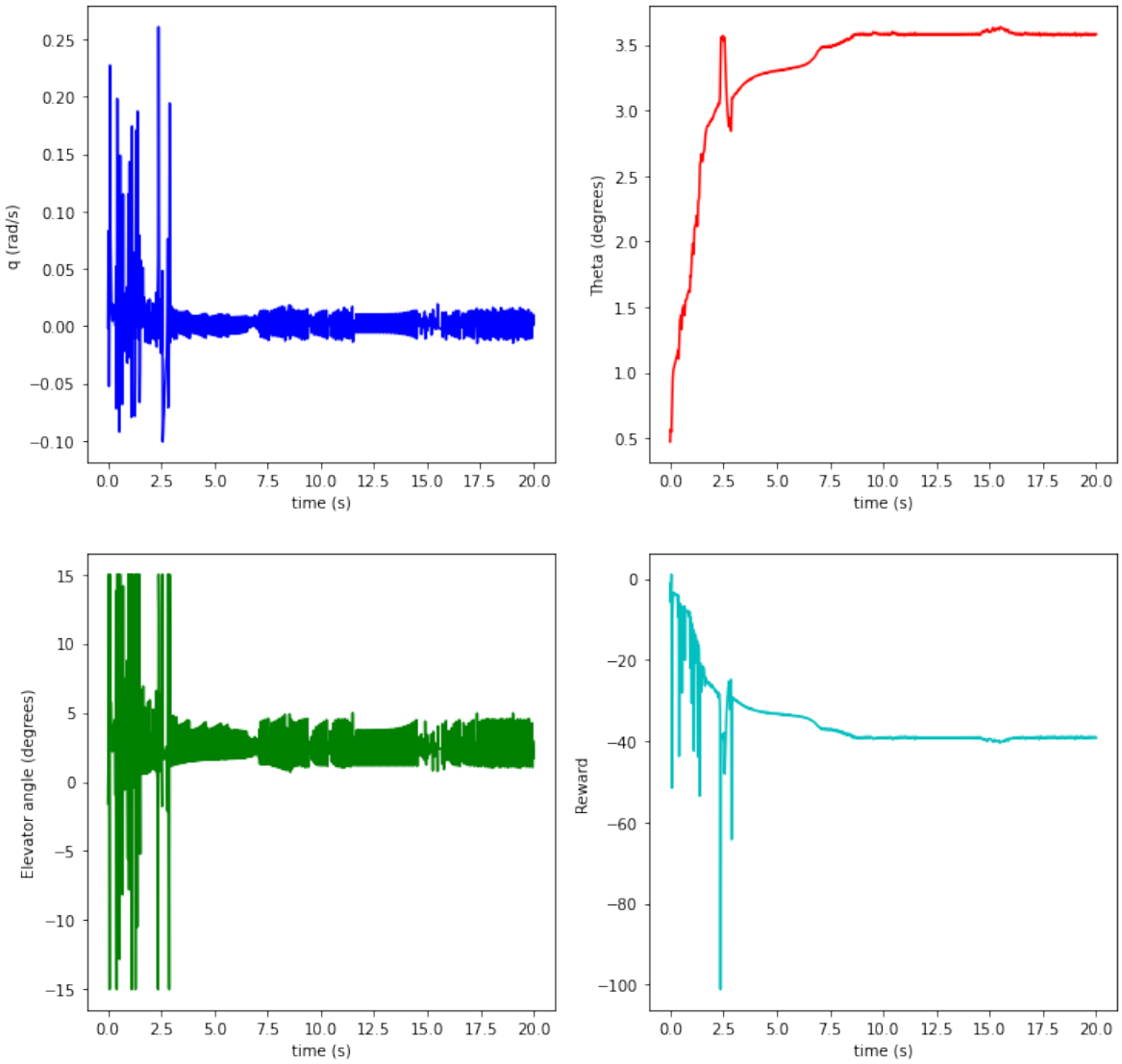

Figure 5.11: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = 4°$

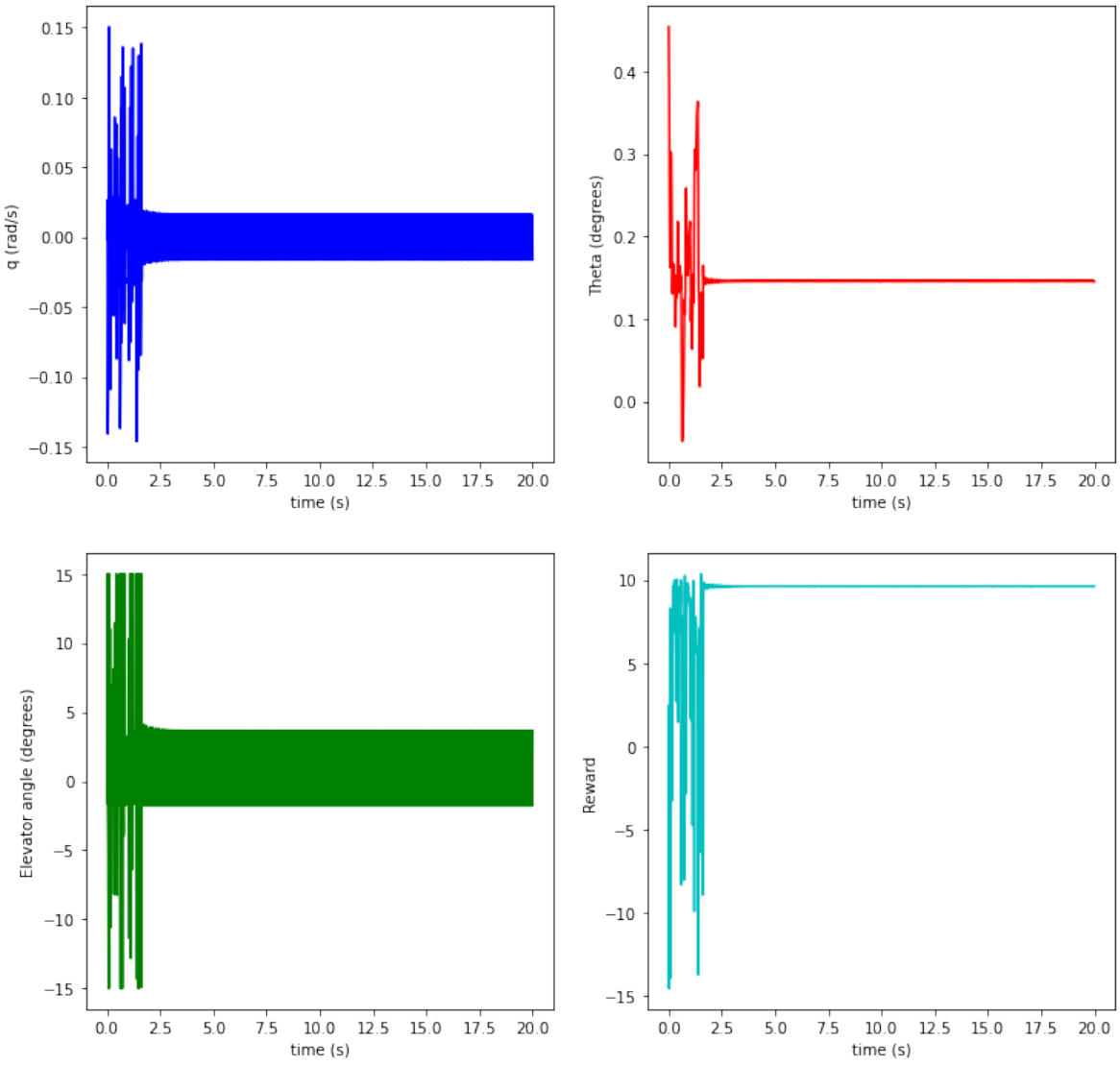

Figure 5.12: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = -3°$

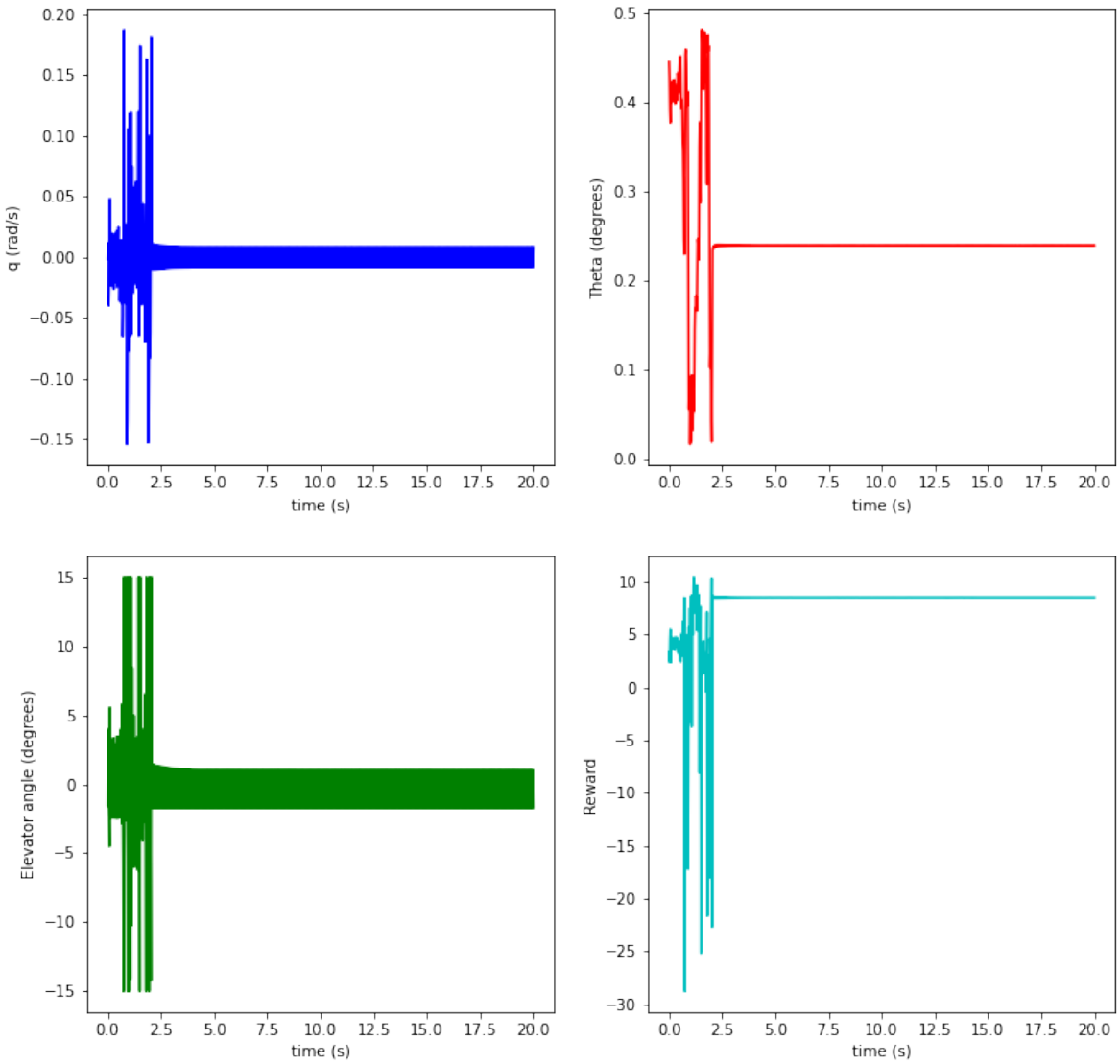

Figure 5.13: Results obtained for SAC agent trained using the state $(q, \Delta\theta, \dot{q})$ using the second training methodology for a $\theta_{goal} = -6°$

The following observations can be made from the above 5 figures.

1. The high frequency oscillations still exist but the range of variation decreases after some time for all the plots except 5.9.

2. For the positive θ_{goal}, the convergence is close to the required goal angle but there exists steady state error, a maximum of 0.5° for a θ_{goal} of 4°.

3. But for negative angles, the θ doesn't go into the negative side too and converges to a small positive value close to zero.

5.3 Reward Engineering

After exhausting all the 3 possible reasons identified for non-convergence in the beginning the next possible route is to make use of other reward functions and engineer the reward function appropriately to improve the learning of the agent.
The reward function is changed to a PID based reward function,

$$R = -|0.9059 \cdot e_\theta + 0.2738 \cdot \int e_\theta + 0.4407 \cdot \frac{de_\theta}{dt} - \delta_e|$$

The K_p, K_d, and K_i values are obtained using MATLAB's PID tuner for the pitch controller for the given aircraft.
But this didn't improve the convergence of θ and the high frequency oscillations of δ_e and q.
Along with this the other reward function opposite to the quadratic function in terms of gradients, the gaussian function is incorporated into the reward function and is implemented but this too doesn't improve the performance of the agent.
To improve the SSE the integral of the error is also incorporated into the reward function and combined with the two reward functions on top but this too yields no improvement.
The reward function

$$R = -\Delta\theta^2 - q^2 - |\Delta\delta_e|$$

If $\Delta\theta < 1^\circ$,

$$R = R + 10 \cdot (1^{\circ^2} - \Delta\theta^2)$$

If $\Delta\theta > 3^\circ$,

$$R = R - 10 \cdot \Delta\theta^2$$

is implemented. Here along with a boost in the reward for better performance, there is a dip in the reward when the $\Delta\delta_e$ goes above a certain fixed value, taken to be 3°. This too doesn't provide any improvement in the performance.

5.4 Visiting DDPG again

The following changes are made to the deep deterministic policy gradient algorithm and the agent is trained.

1. The size of the neural network is changed with 300 neurons in each layer for the actor and 400 neurons in each of the two hidden layers in the critic network.
2. The discount factor is fixed to a value of 0.99
3. The size of the replay buffer is fixed to 10^6
4. The reward function implemented is

   $$R = -\Delta\theta^2 - q^2 - |\Delta\delta_e|$$

   but in this case, θ is taken in degrees.
5. The state vector is changed to $(q, \Delta\theta, \theta)$
6. Providing both $\Delta\theta$ and θ may help improve the performance

The results obtained by implementing the above DDPG agent are shown in Figs. 5.14 to 5.20.

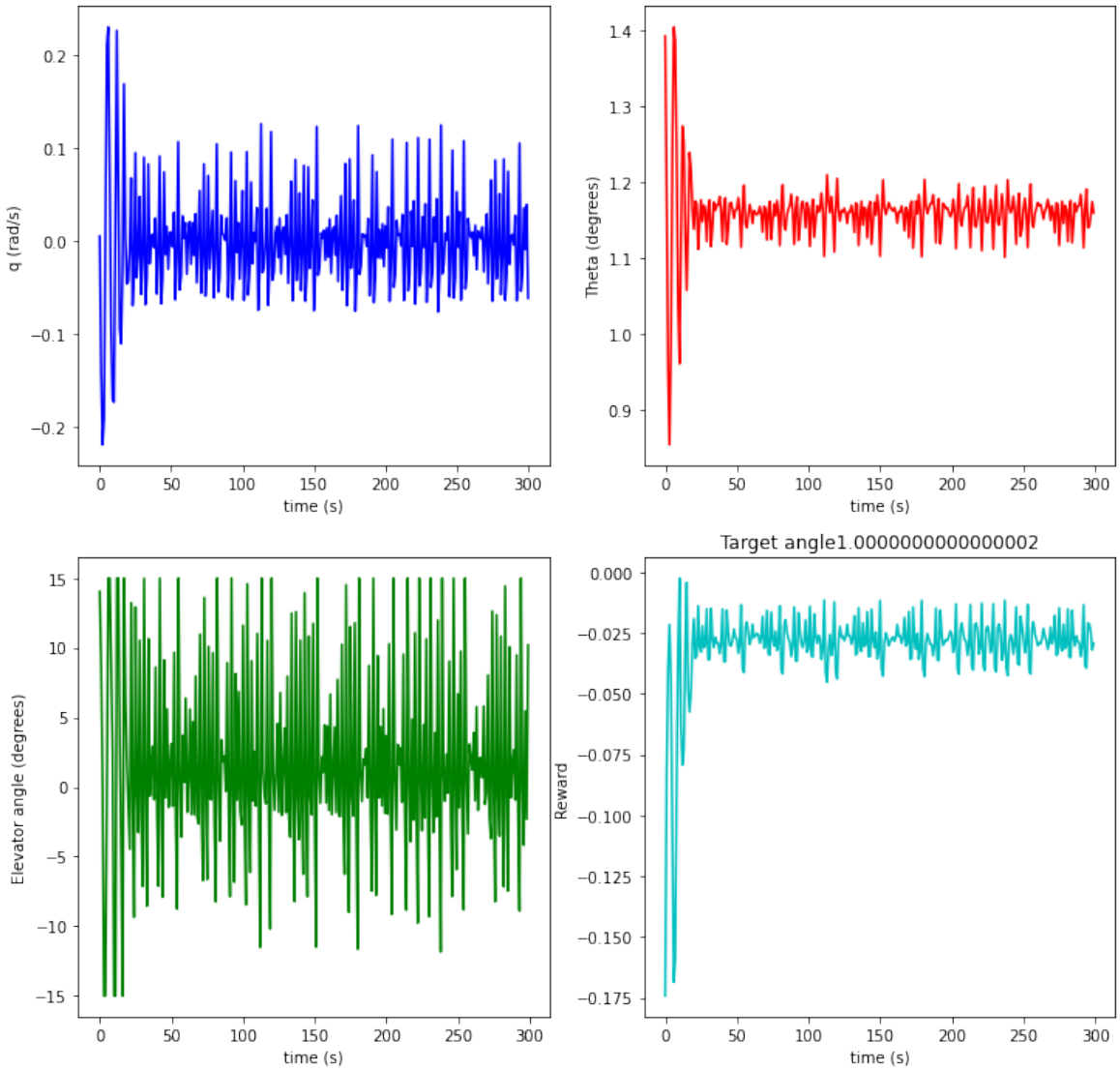

Figure 5.14: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1^{\circ}$

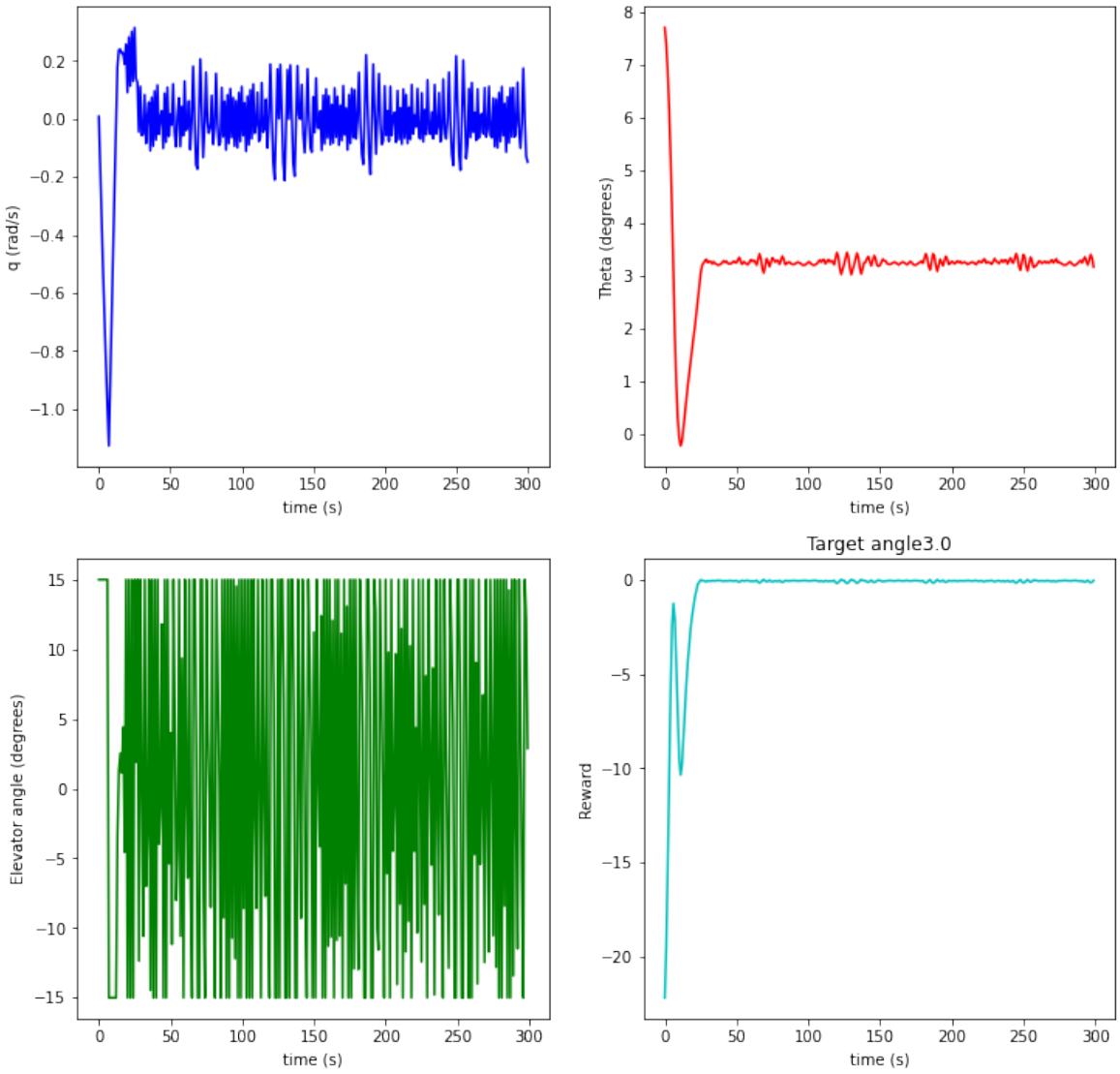

Figure 5.15: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 3°$

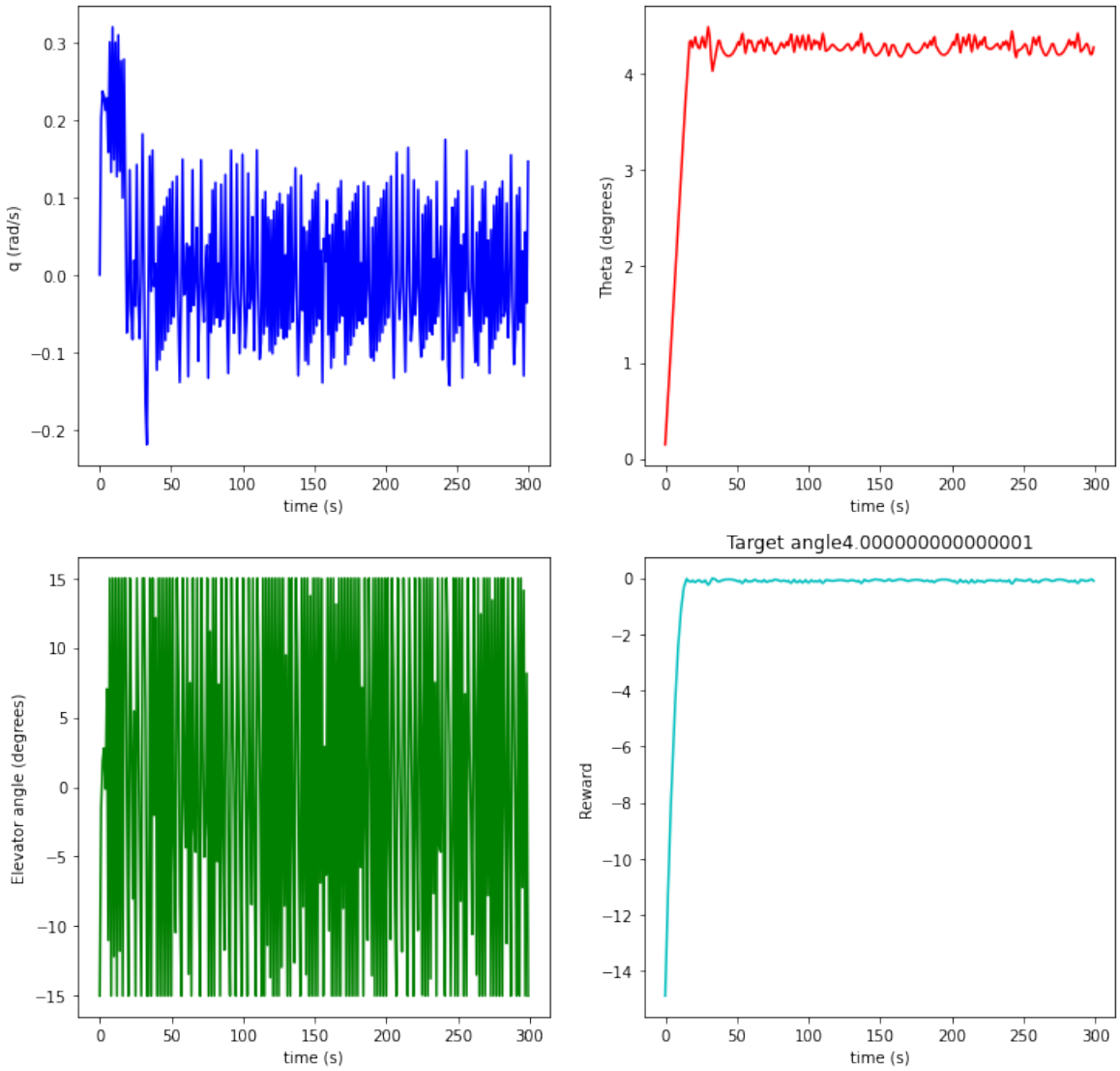

Figure 5.16: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 4°$

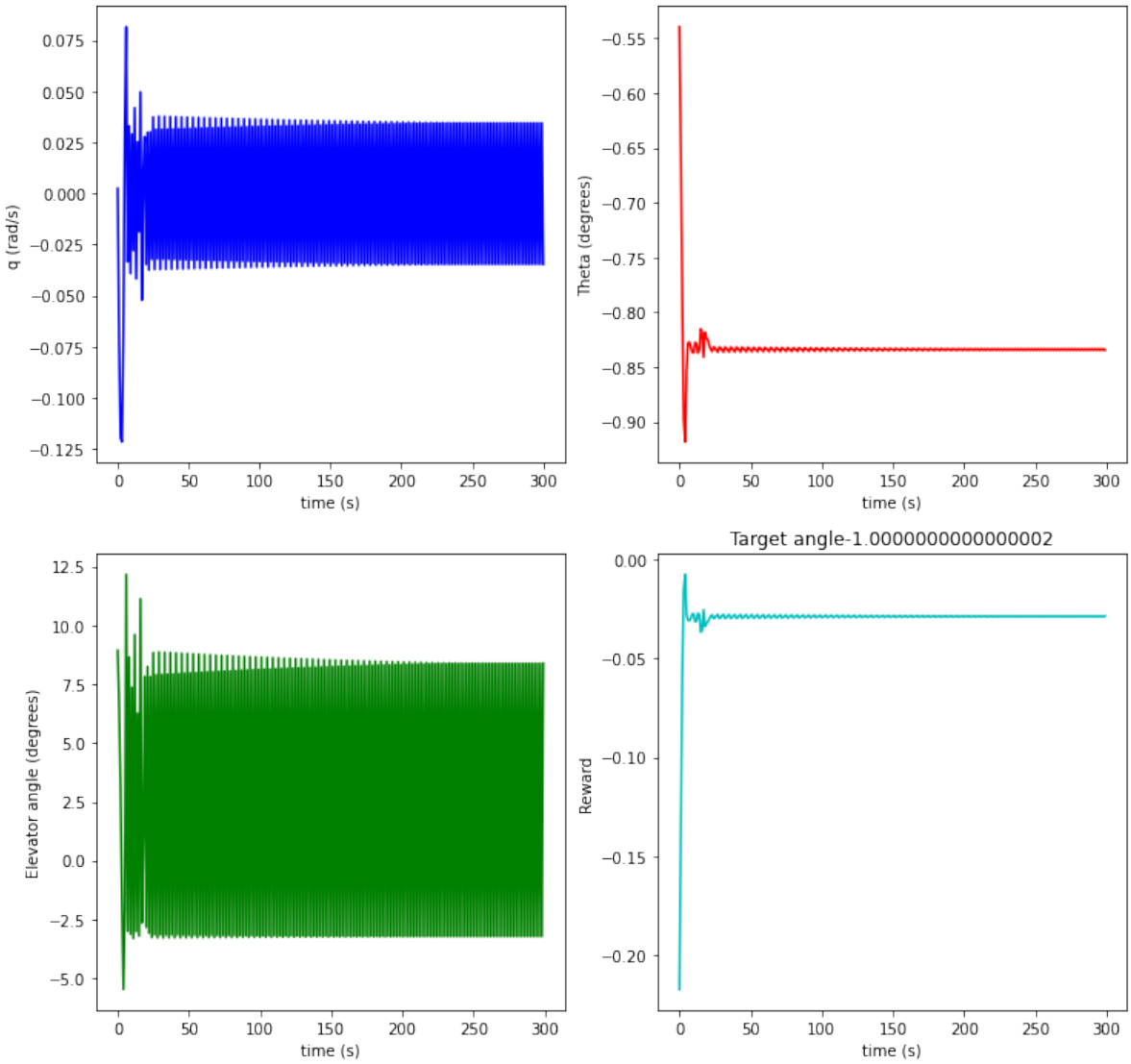

Figure 5.17: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1°$

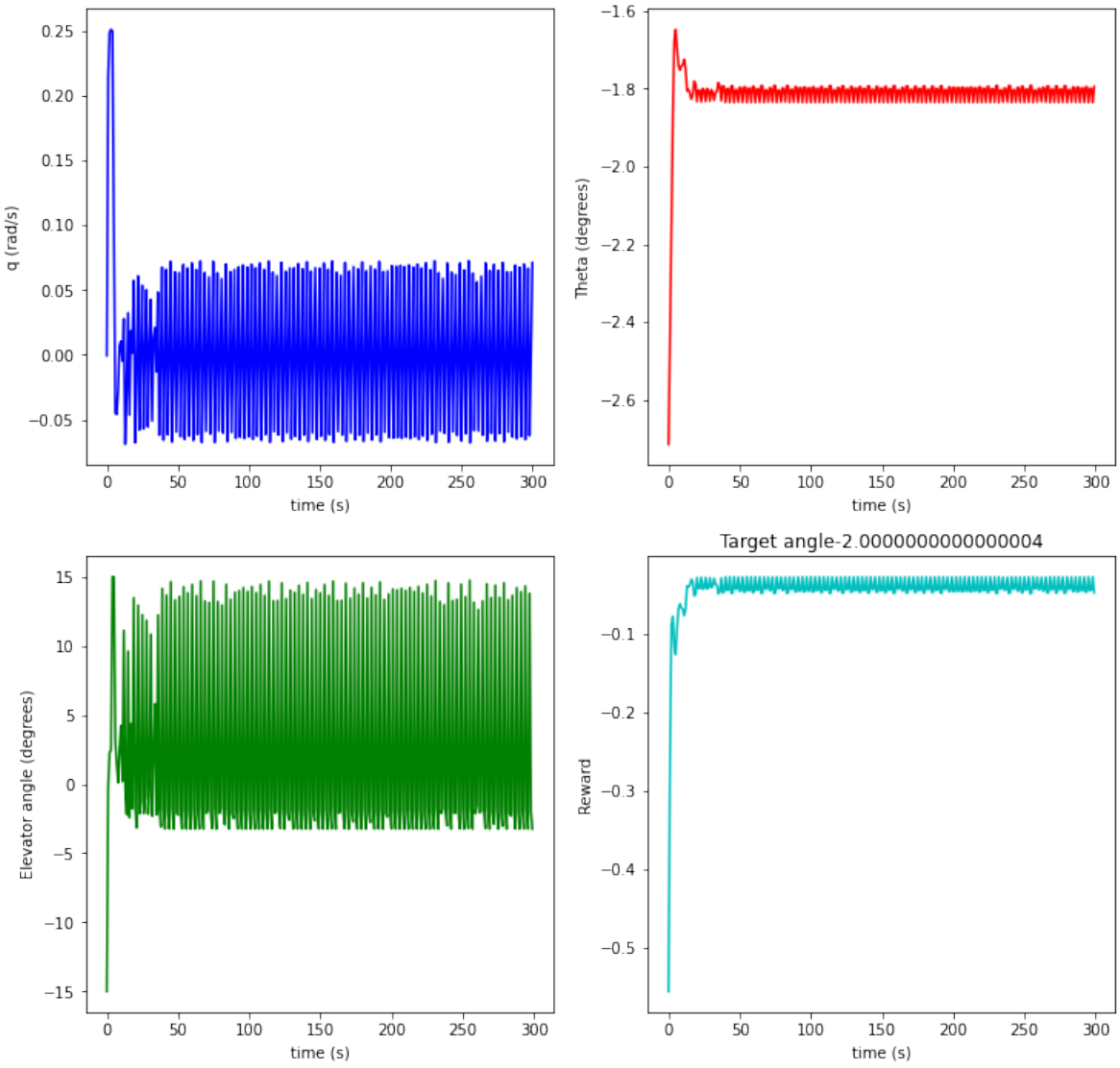

Figure 5.18: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -2°$

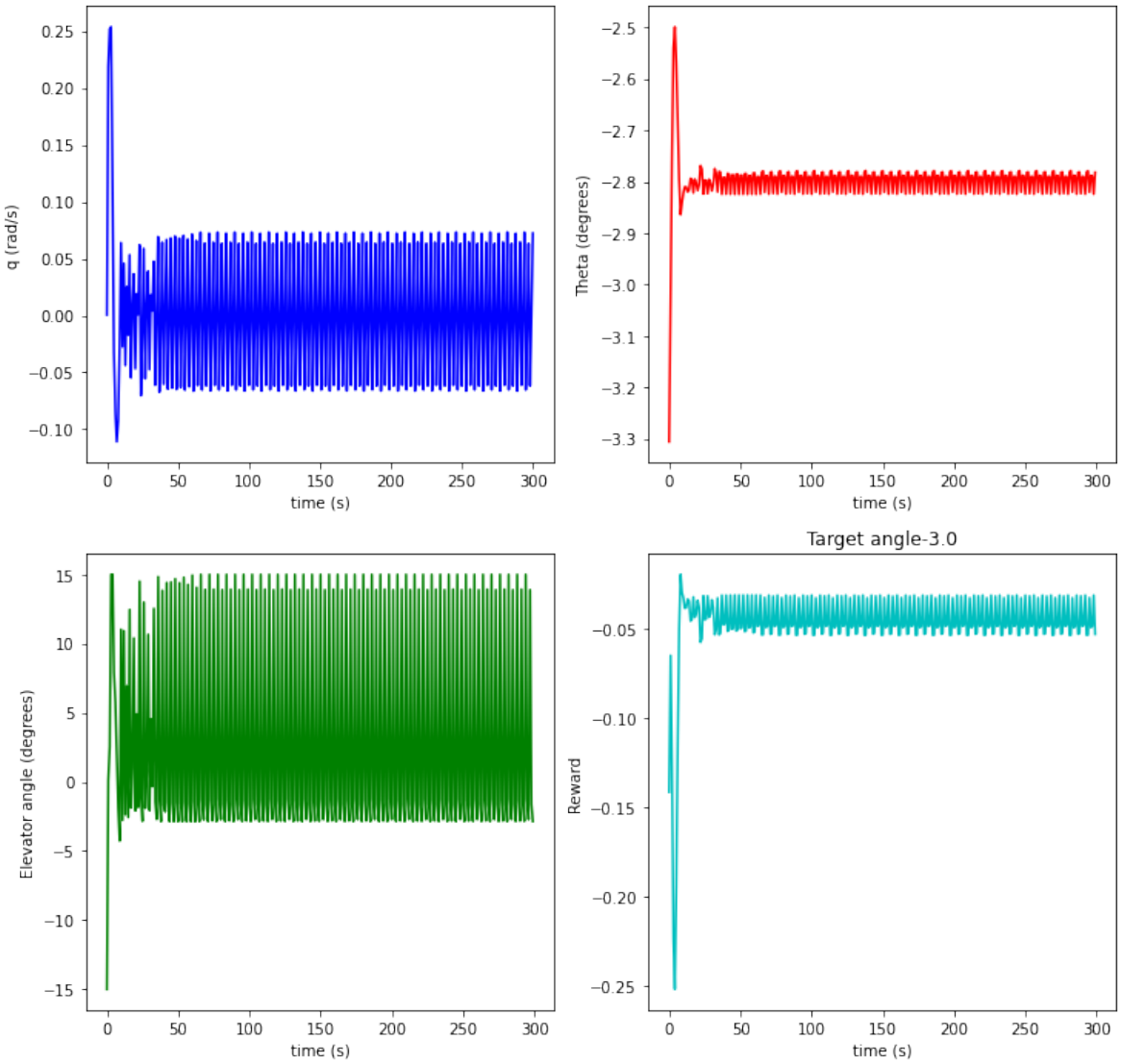

Figure 5.19: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -3°$

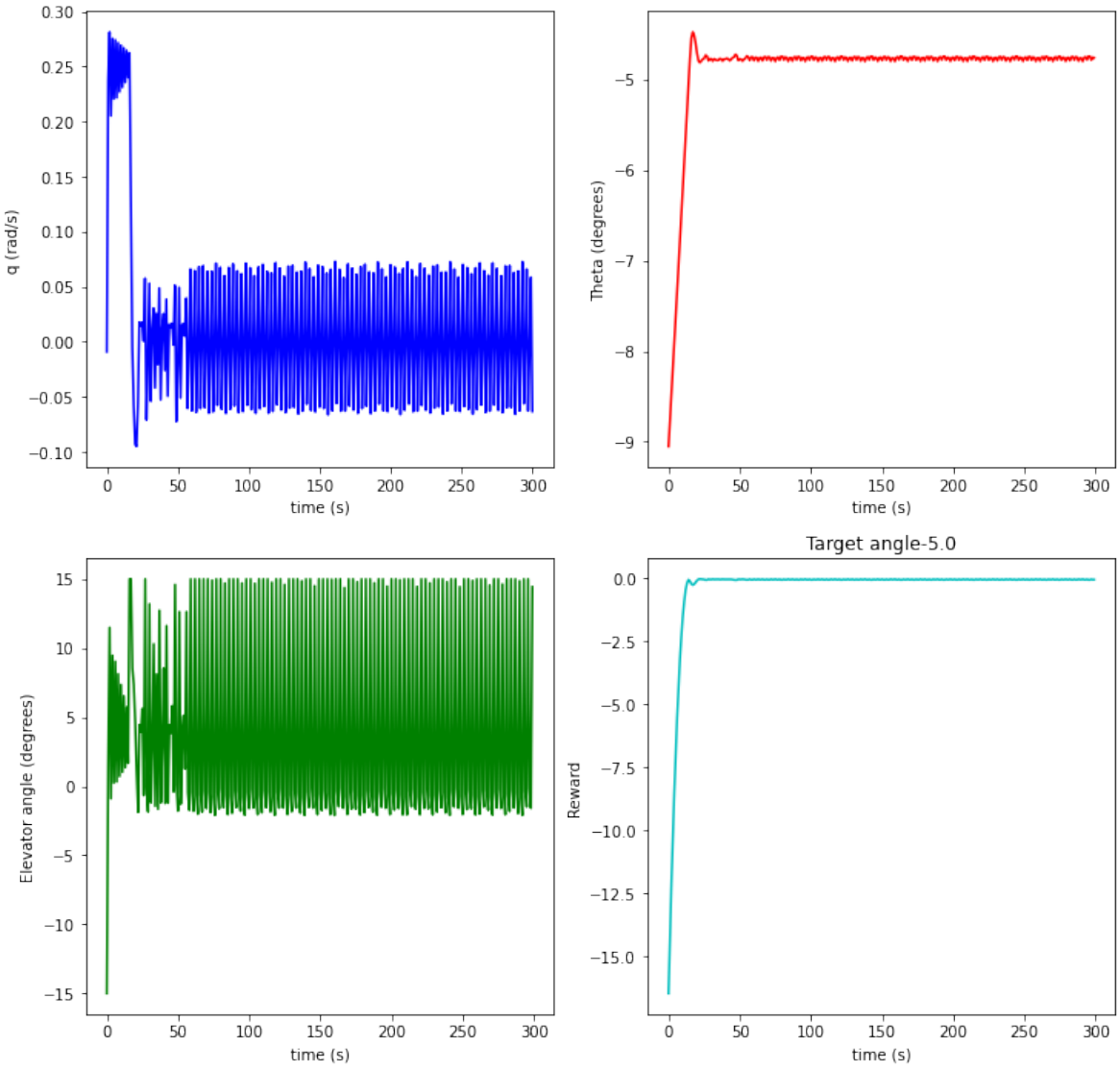

Figure 5.20: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -5°$

The following observations can be made from the above plots,

1. There exist high frequency oscillations in q and δ_e variations.
2. But along with this, there is a high degree of convergence for different values of θ_{goal}
3. This has not been observed in any previous trials.
4. Also, the convergence is for both positive and negative values of θ_{goal}

This is certainly an improvement from the previous trials, taking it further the reward function is modified to

$$R = -\Delta\theta^2 - q^2 - |\Delta\delta_e|$$

If $\Delta\theta < 1^\circ$,

$$R = R + 10 \cdot (1^{\circ^2} - \Delta\theta^2)$$

and the results obtained by this model are shown in Figs. 5.21 to 5.36.

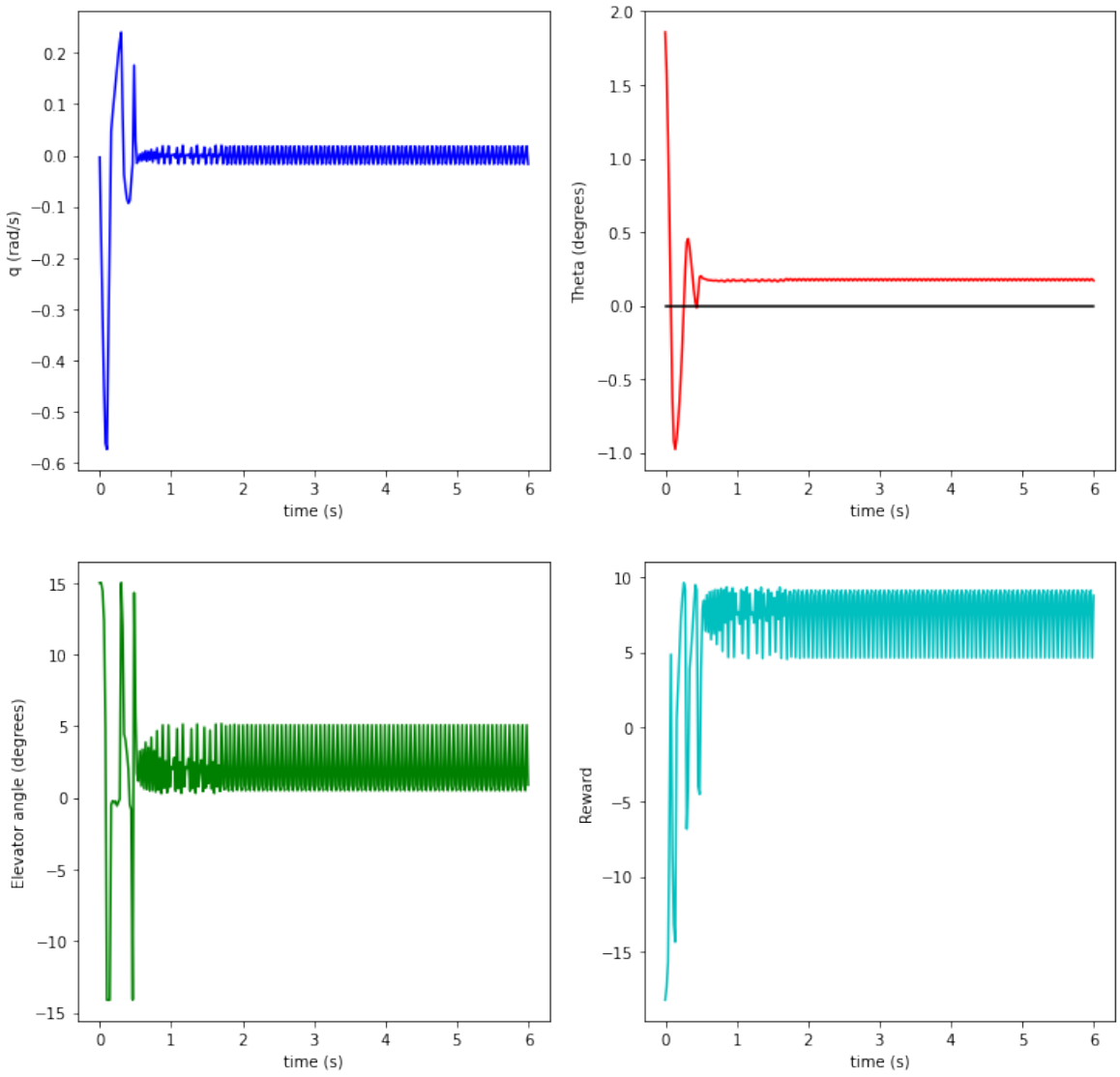

Figure 5.21: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 0°$

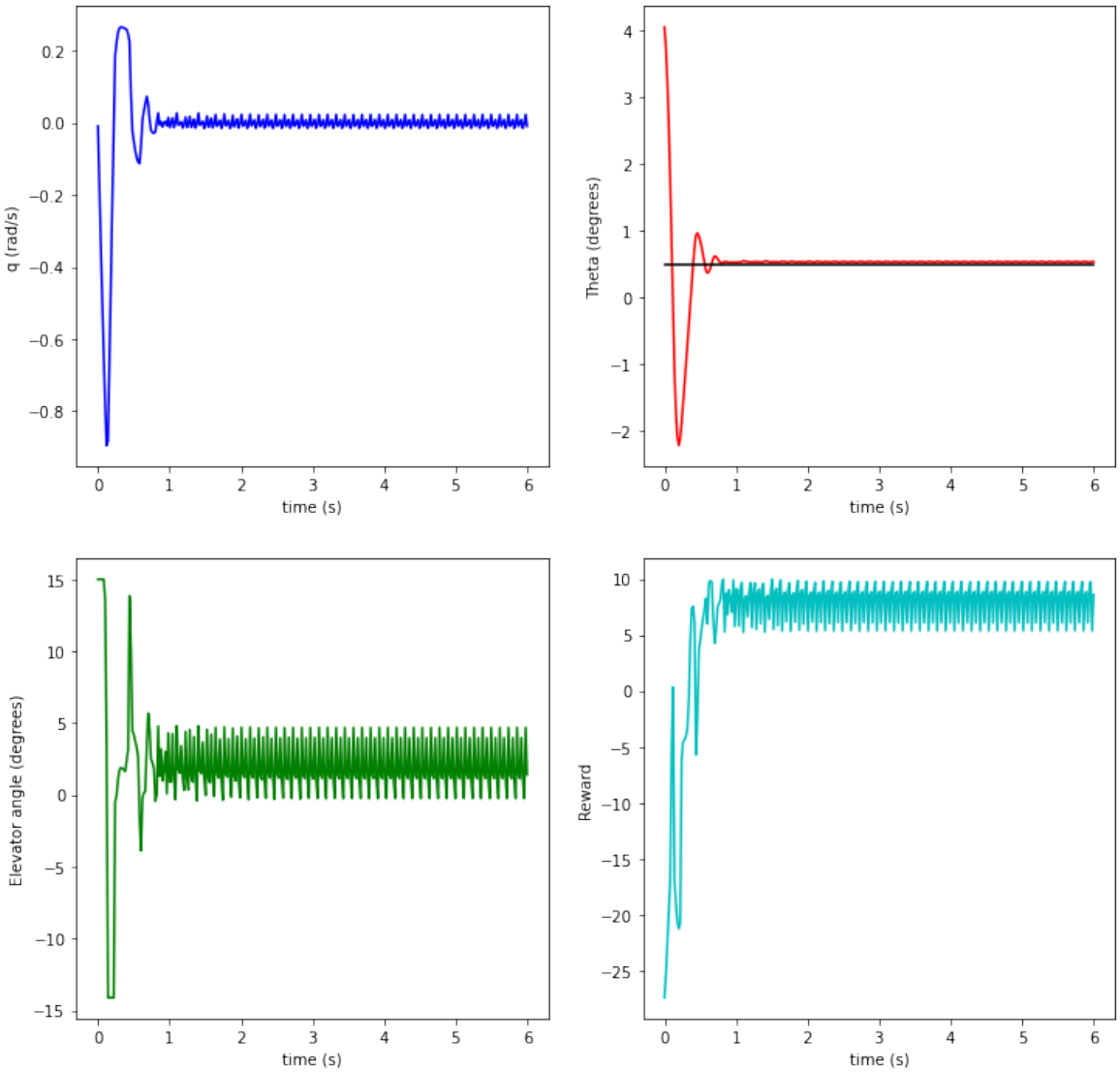

Figure 5.22: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 0.5°$

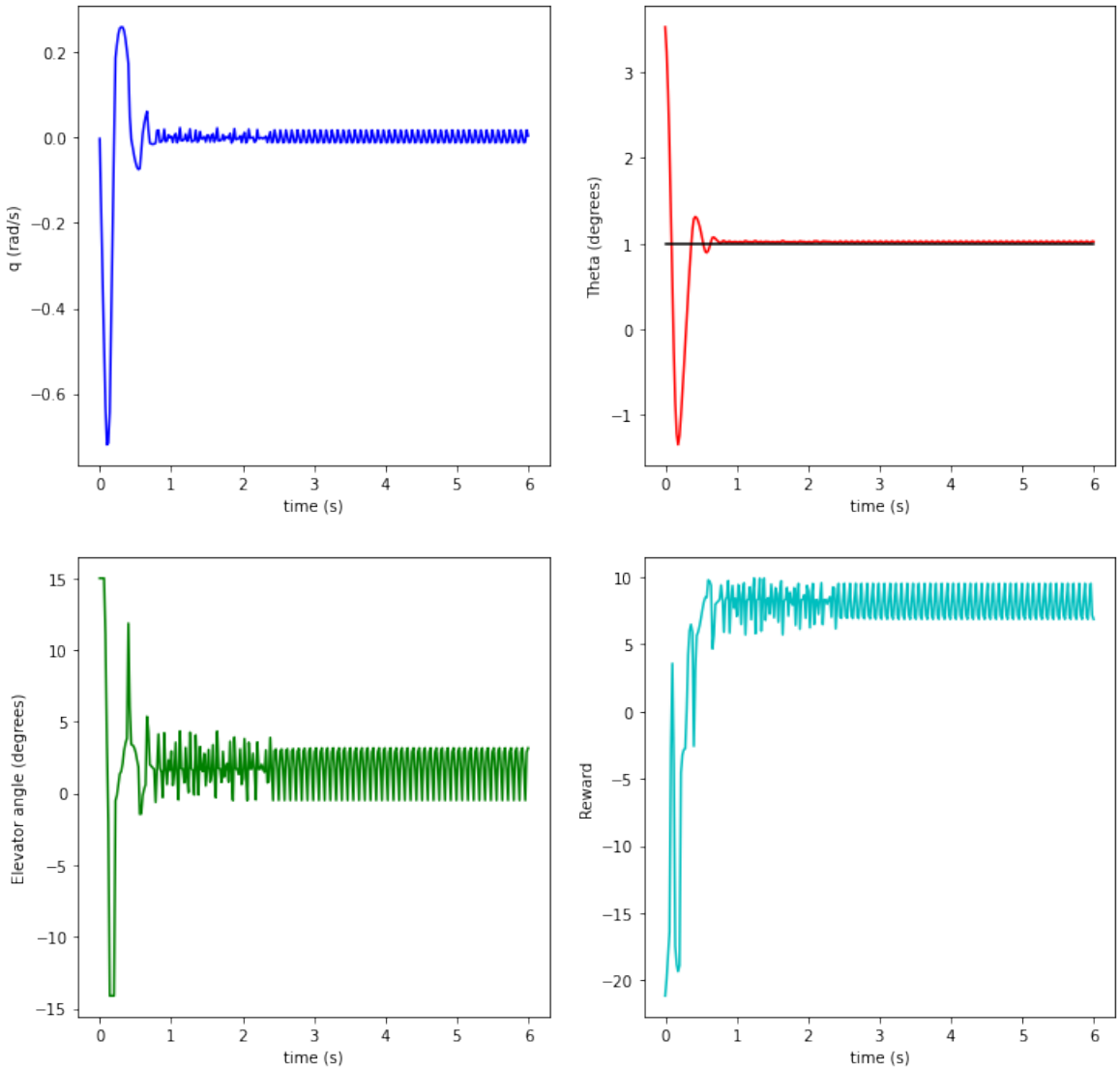

Figure 5.23: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1^{\circ}$

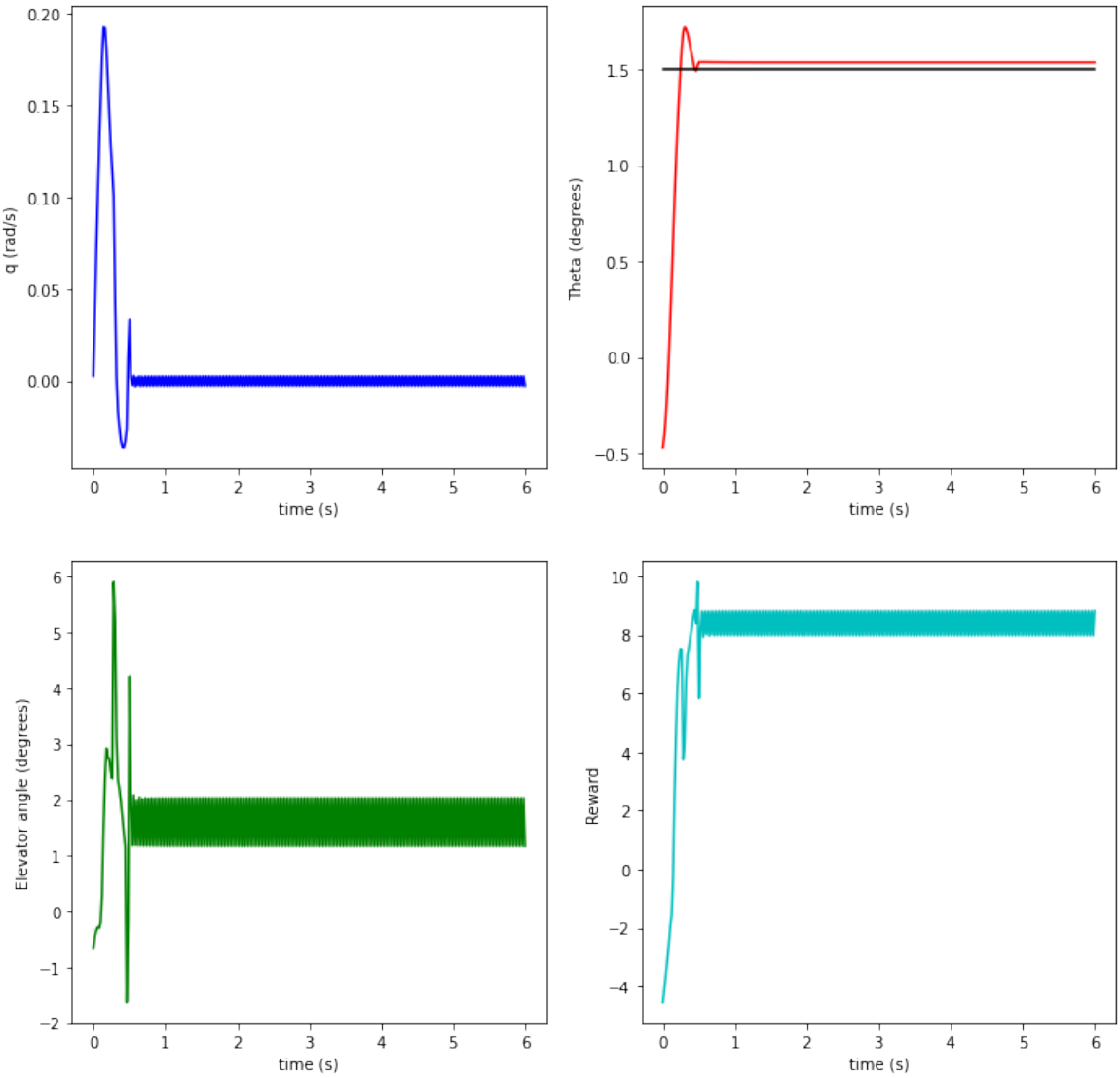

Figure 5.24: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 1.5°$

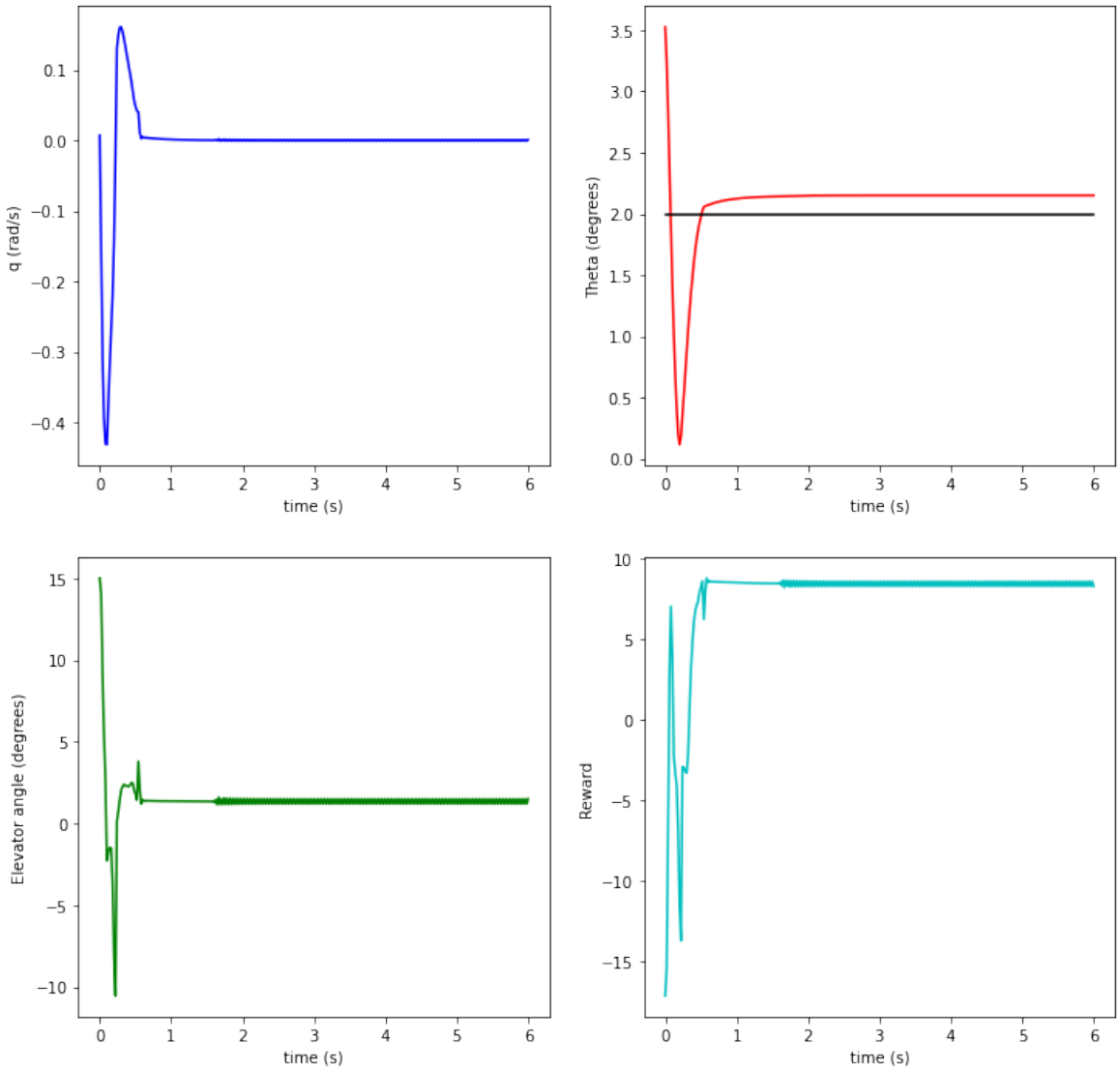

Figure 5.25: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 2^{\circ}$

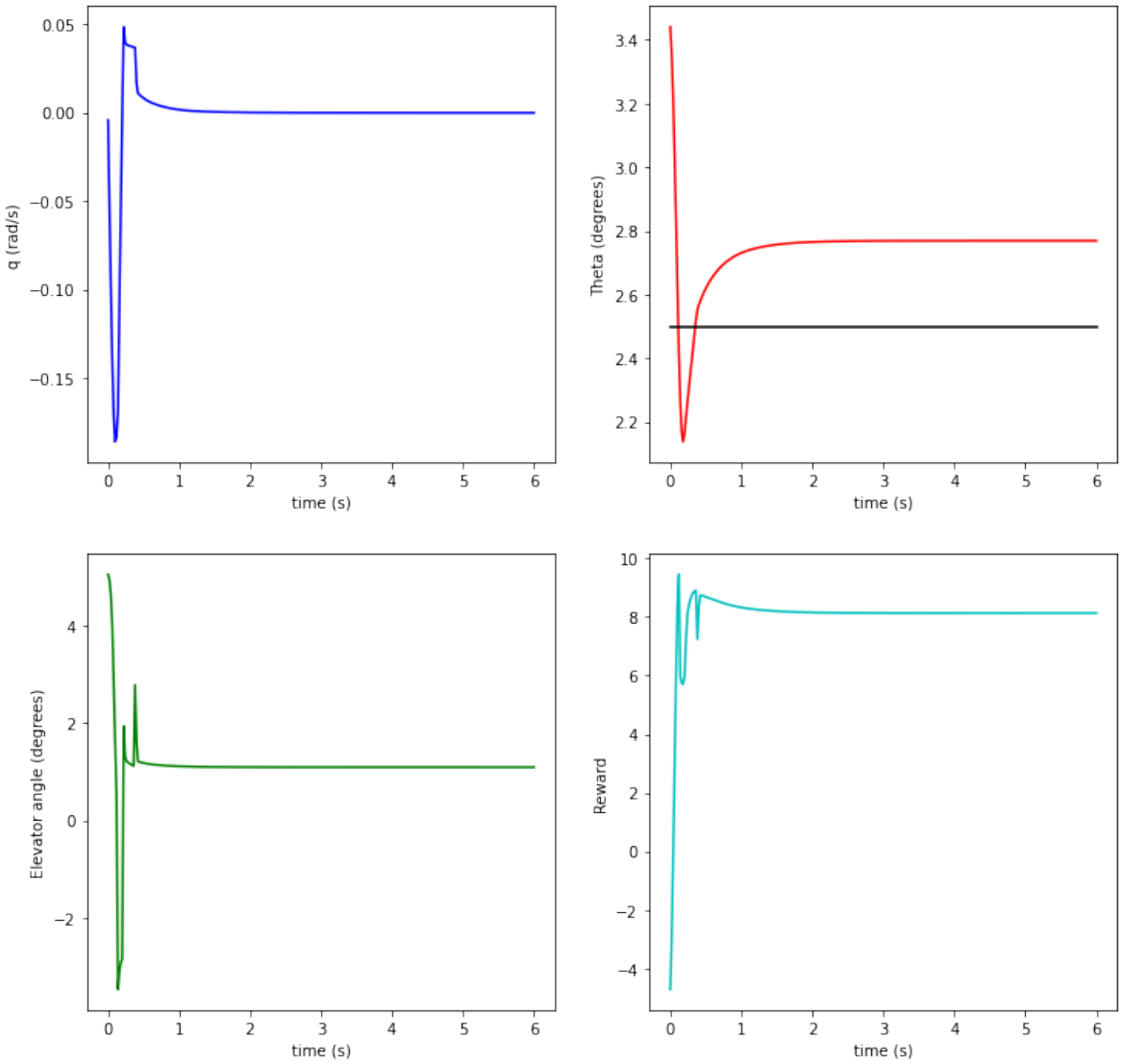

Figure 5.26: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 2.5°$

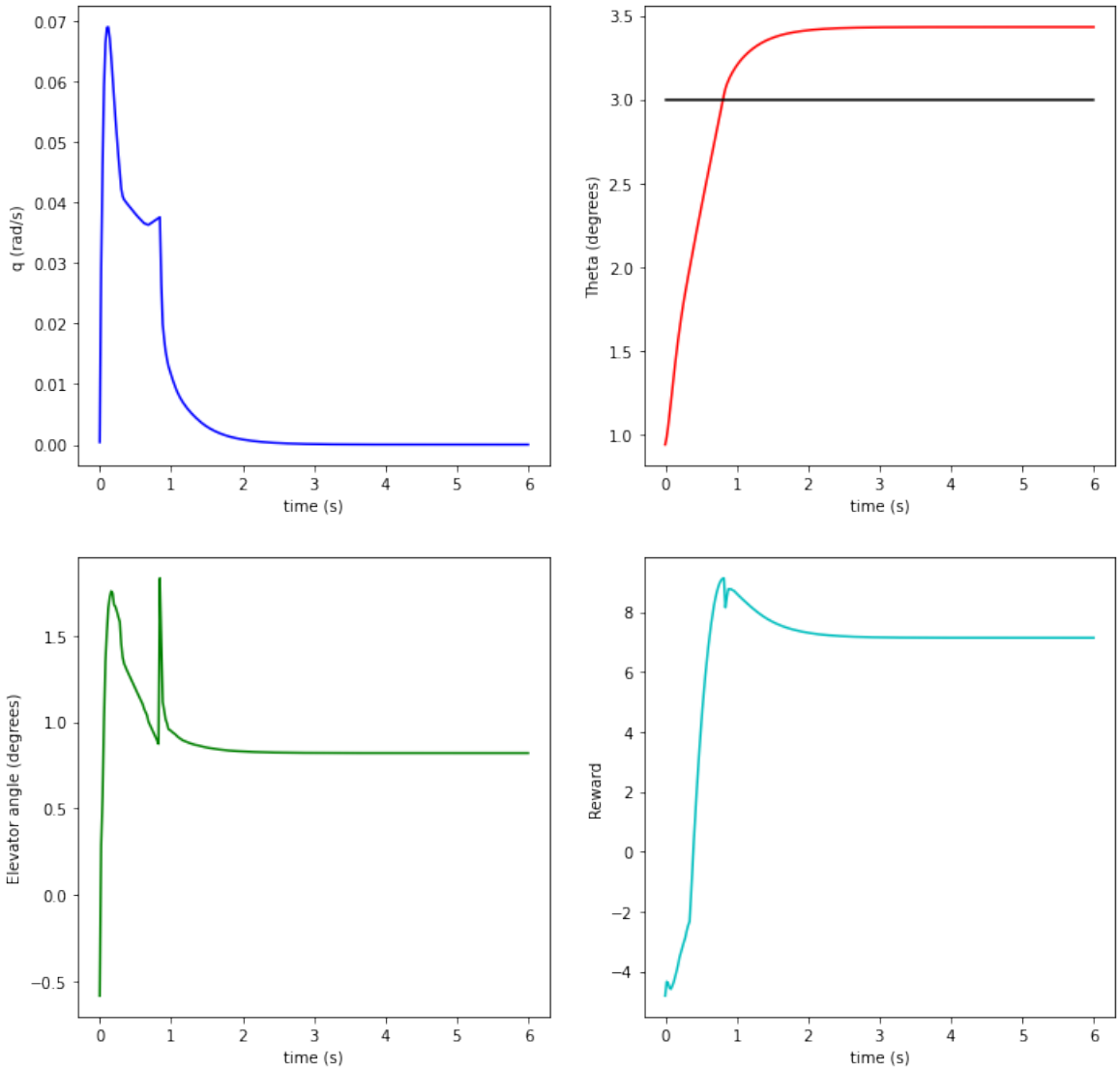

Figure 5.27: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 3°$

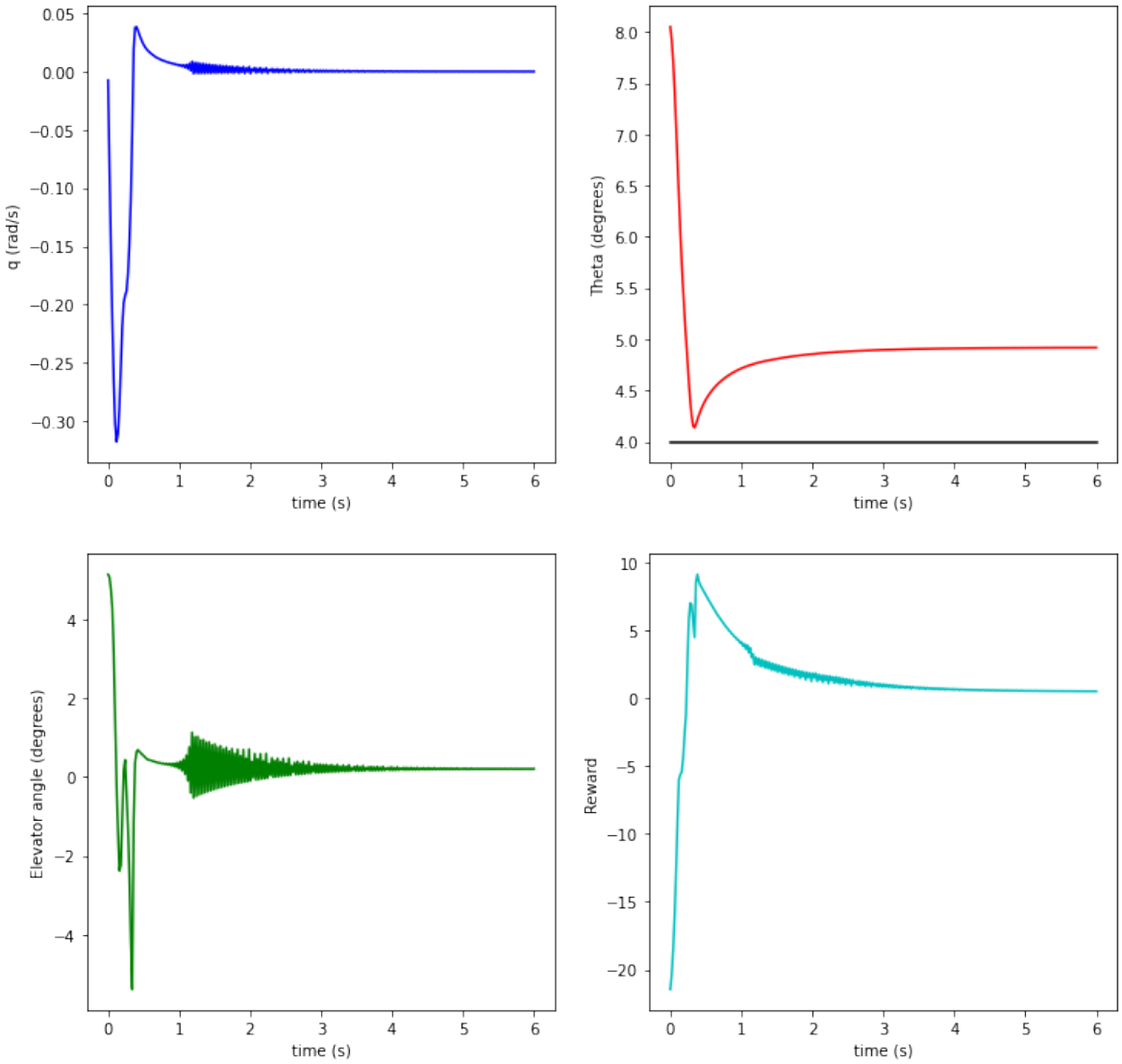

Figure 5.28: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = 4°$

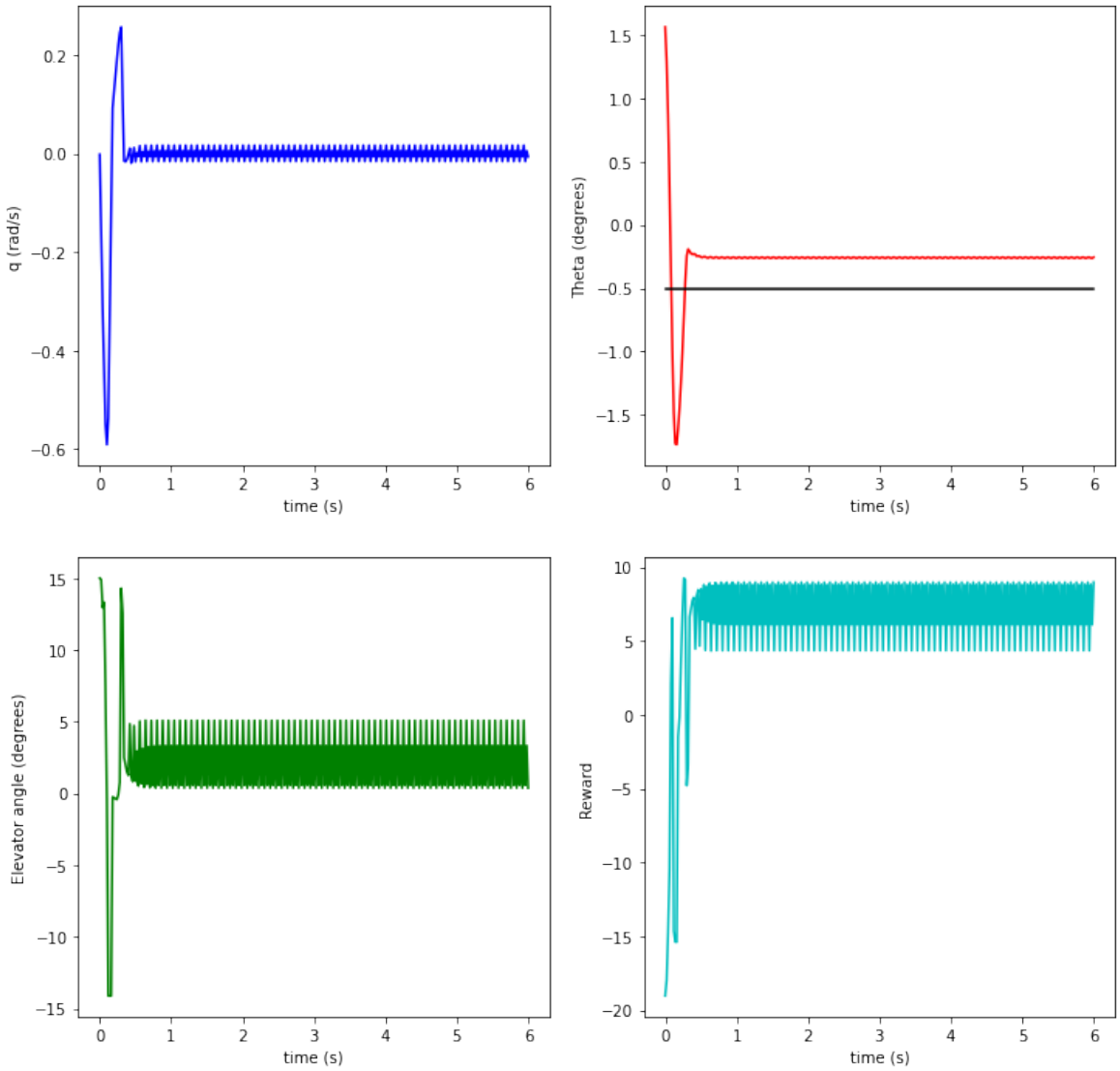

Figure 5.29: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -0.5°$

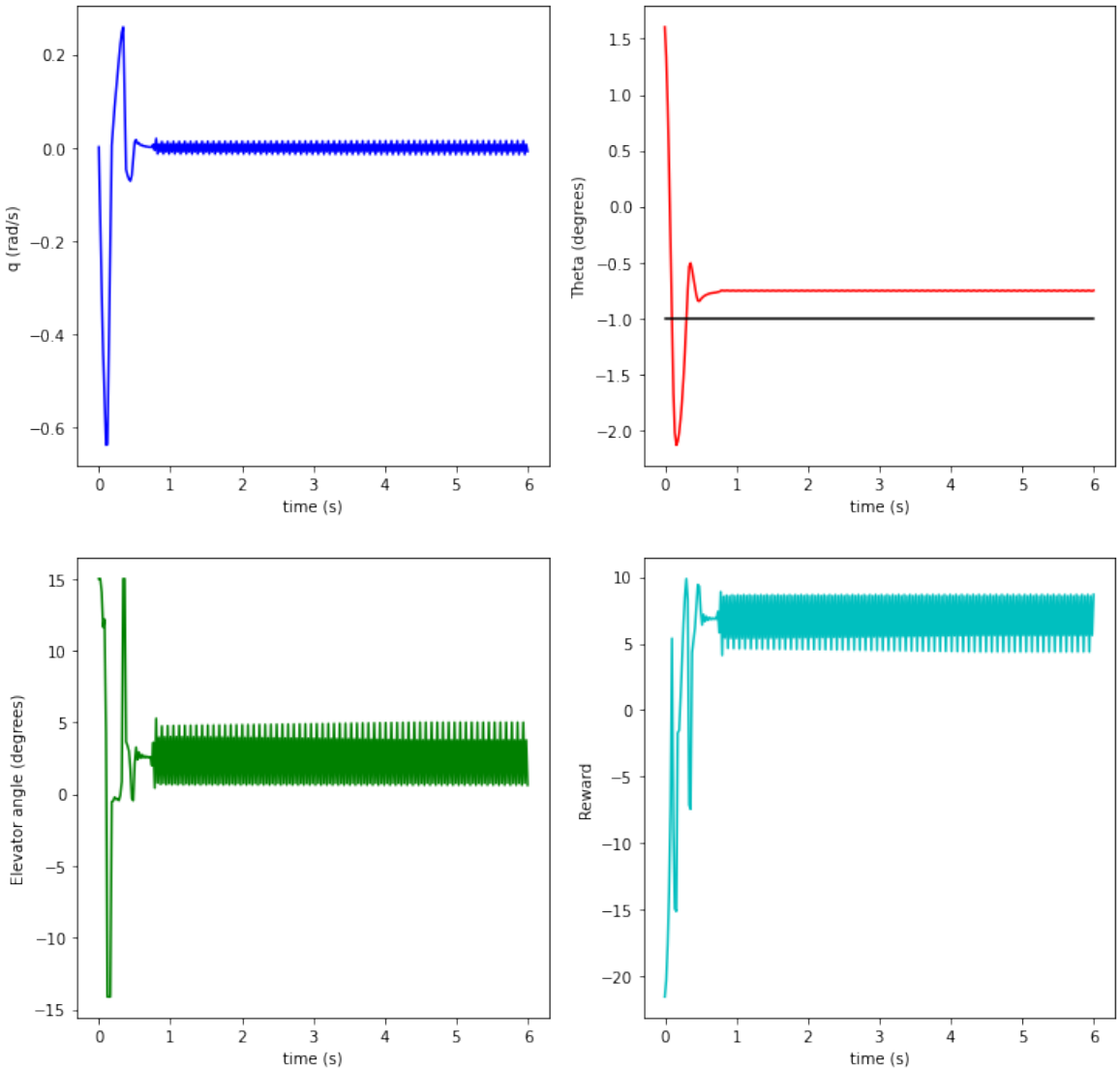

Figure 5.30: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1°$

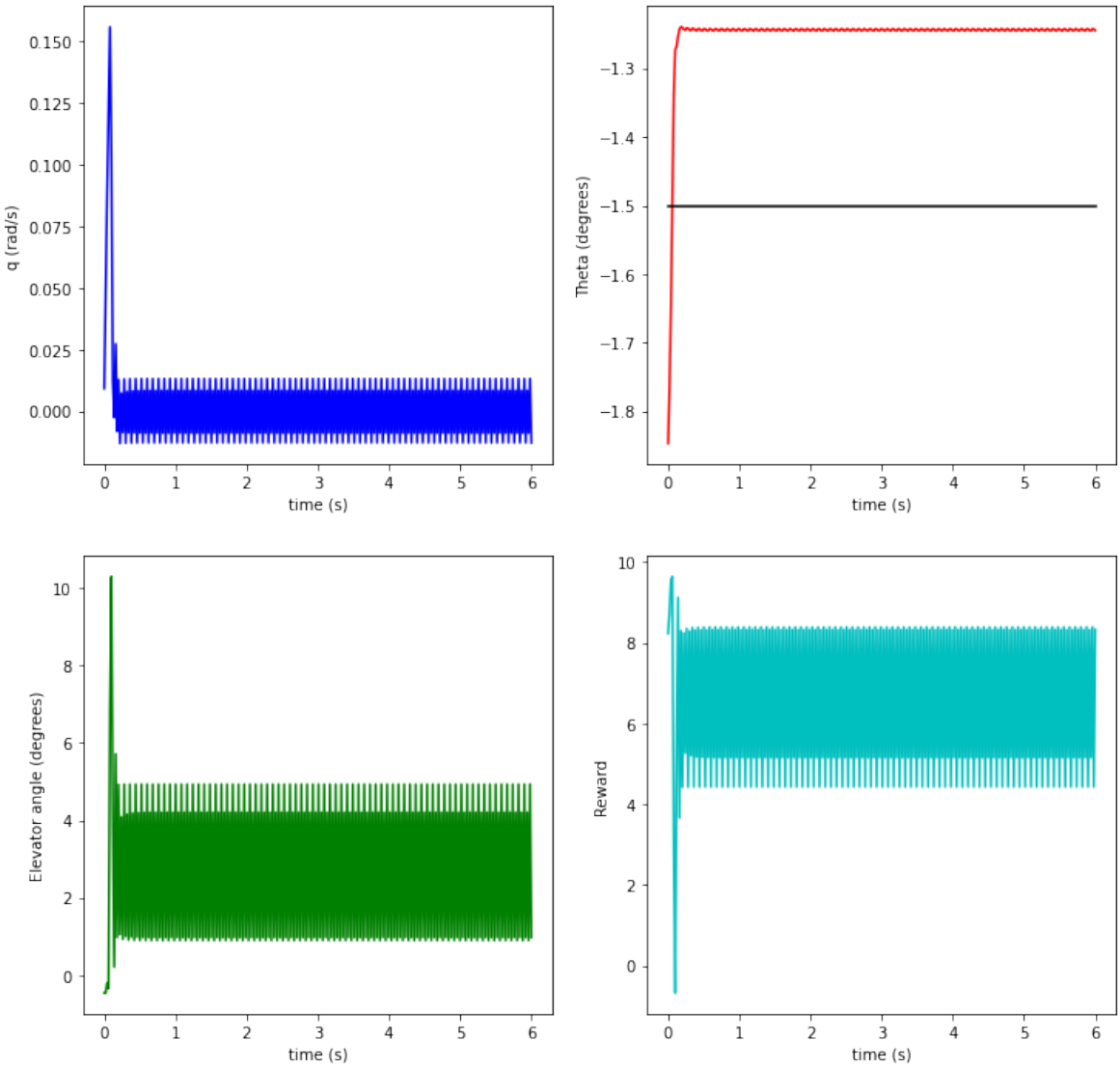

Figure 5.31: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -1.5°$

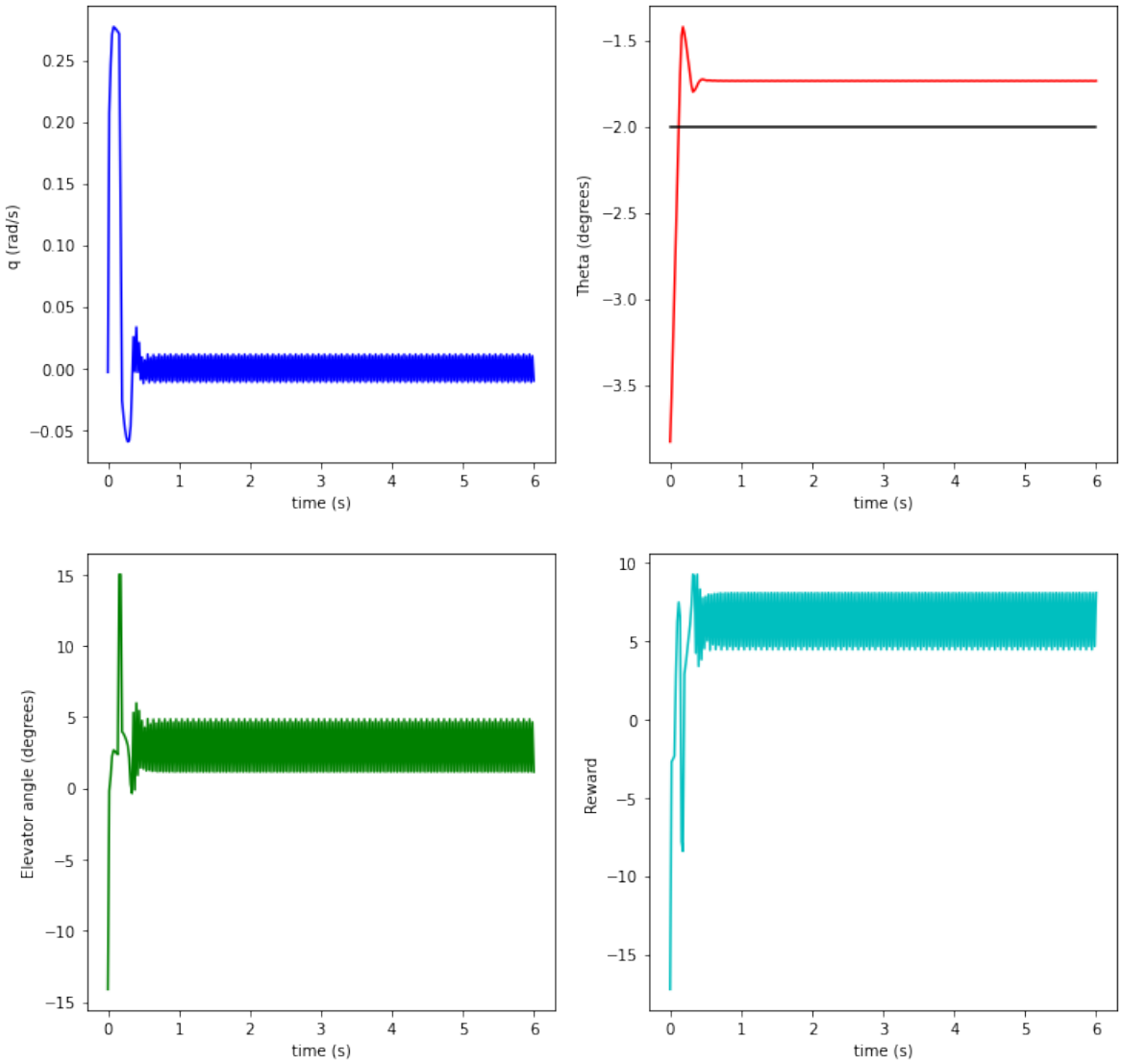

Figure 5.32: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -2°$

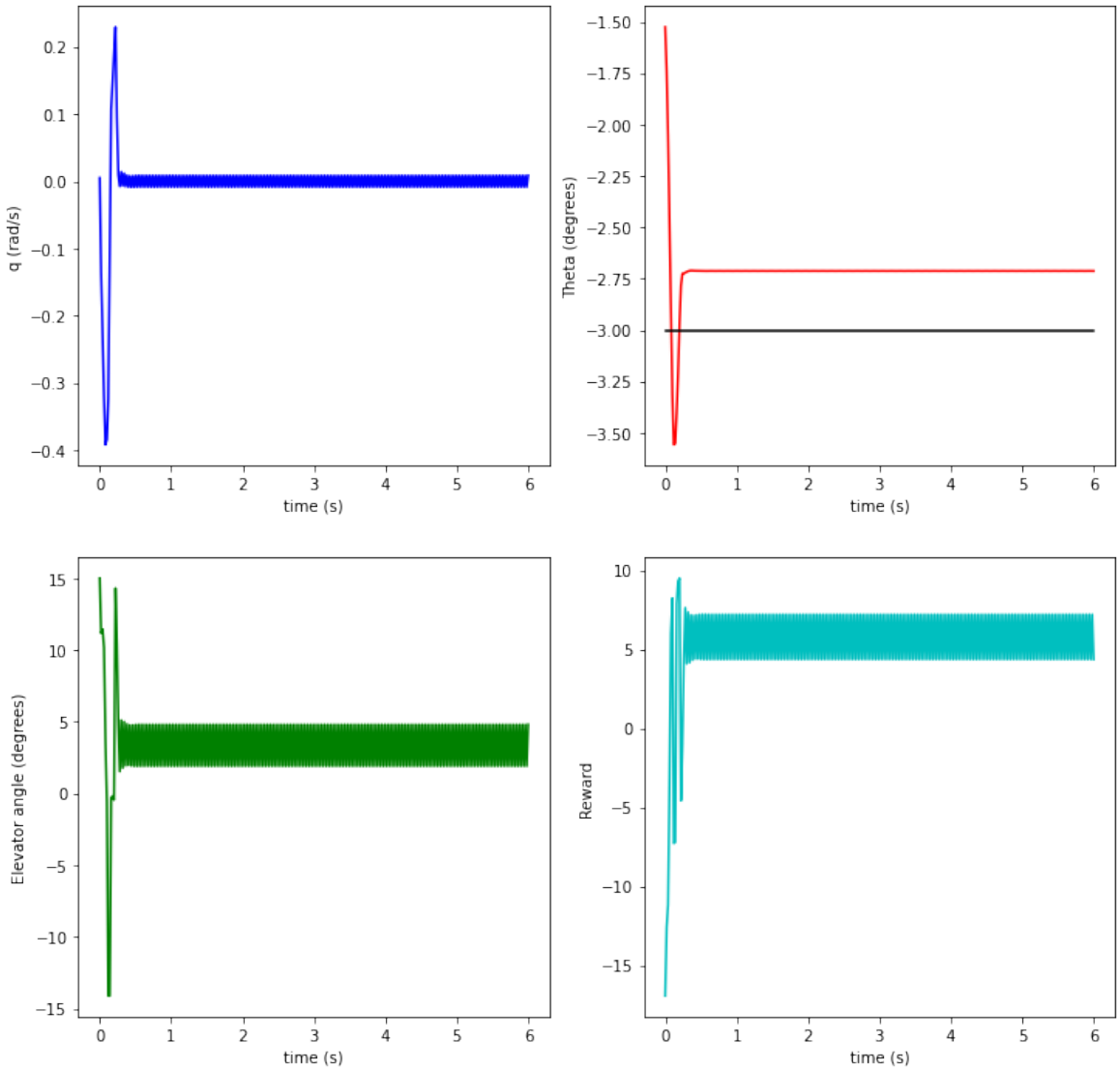

Figure 5.33: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -3°$

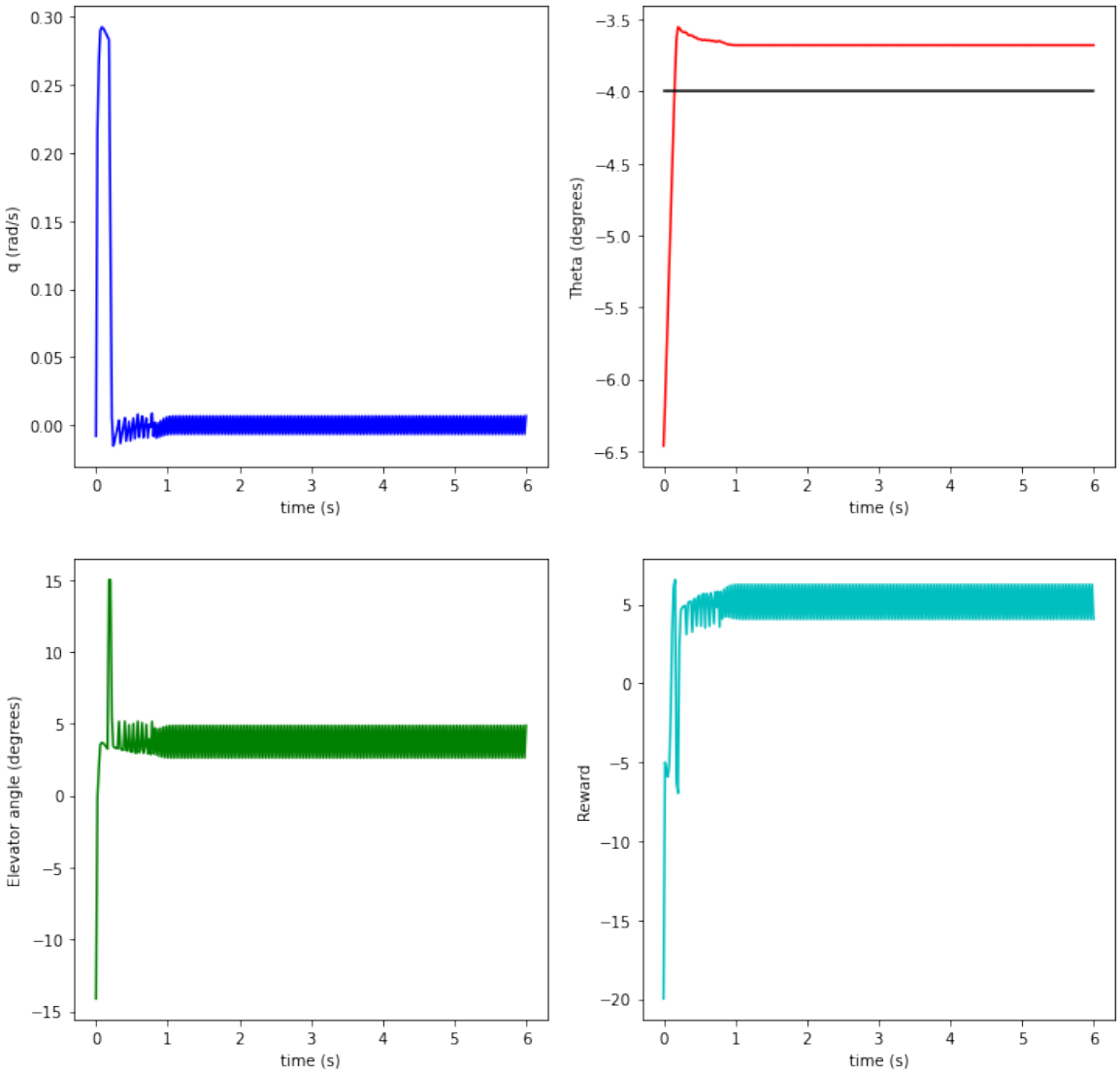

Figure 5.34: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -4°$

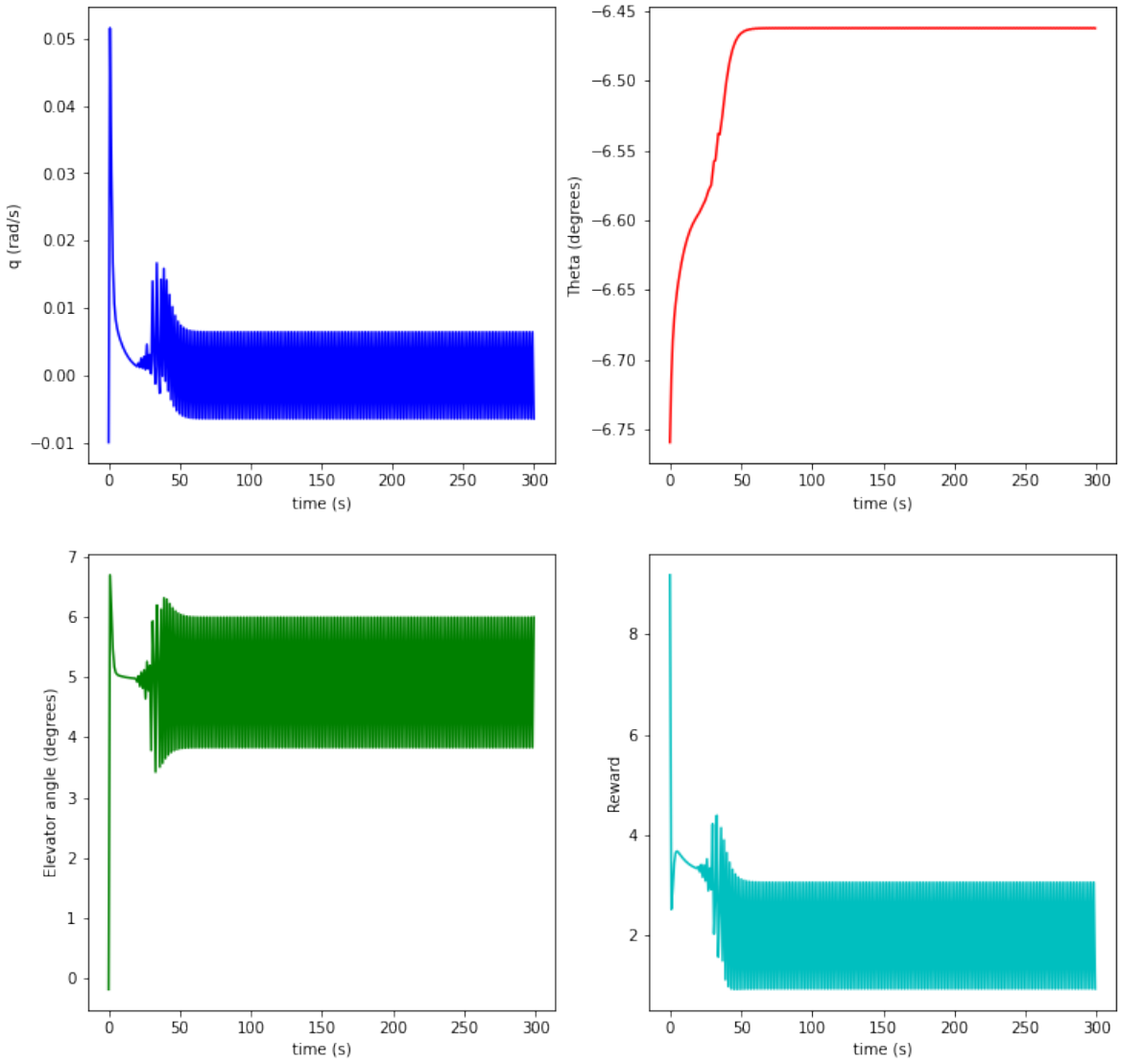

Figure 5.35: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -7°$

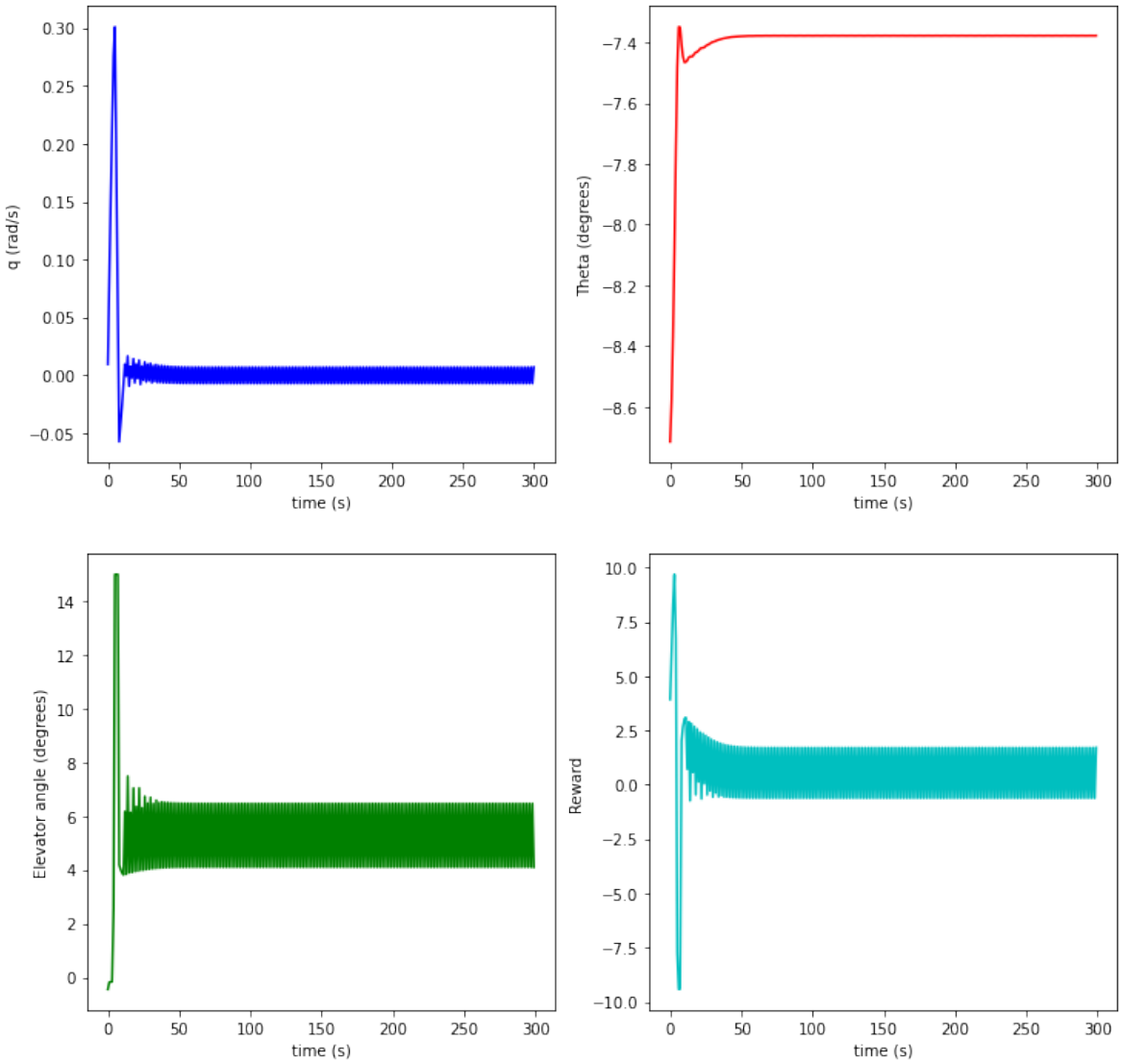

Figure 5.36: Results obtained for DDPG agent trained using the state $(q, \Delta\theta, \theta)$ using the second training methodology for a $\theta_{goal} = -8°$

The following observations can be made from the above plots.

1. The convergence is smooth in most cases.
2. In between -5 and 5 degrees the steady state error is low with a maximum value of 0.3 degrees
3. The high frequency oscillations are not eliminated but has decreased range as compared to the previous agent.
4. The high frequency oscillations are restricted between 0 and 5 degrees of the elevator.
5. The best possible convergence is for a θ_{goal} of 3 degrees where high frequency oscillations are completely absent from all variables.

Chapter 6

Conclusion and Future Works

6.1 Conclusion

In the time frame of the project, a reinforcement learning based non-linear controller for the pure pitching motion of a conventional aircraft could be developed to some extent. More research needs to be done further to develop state-of-the-art non-linear controllers which can perform better or are at par with PID controllers. There exists enough motivation to believe that this is possible as it has been shown in the case of quad-copters. The best of the different controllers developed during this project can settle to angles between -5 and 5 degrees with an SSE of less than 0.3 degrees. The settling time is also as low as 1 second. There exist high frequency oscillations in both q and δ_e and these need to be studied further to eliminate them. This was possible by the use of a DDPG agent with a discount factor of $\gamma = 0.99$ and the reward function

$$R = -\Delta\theta^2 - q^2 - |\Delta\delta_e|$$

If $\Delta\theta < 1^\circ$,

$$R = R + 10 \cdot (1^{\circ^2} - \Delta\theta^2)$$

The actor and the critic had the properties of the second network discussed in chapter 3 with 300 and 400 neurons each in the two hidden layers for the actor and critic network.

6.2 Future Works

The research can be continued further in the following directions so that once this pure pitching motion is established, longitudinal control can be implemented which has 2 control inputs, namely the elevator and the throttle.

6.2.1 Reward function improvement

The reward function currently used is a quadratic function based reward. There can be other reward functions that can provide better convergence results. They need to be explored further.
Another approach can be to model the problem as an optimization function. It is observed from the literature that this method is not generally used for reinforcement learning problems. But since the problem being solved is also new, optimization of the reward function can be tried. The reward function type can be decided a priori, the variables can be the weights given to each term. Now the reinforcement learning model must be trained and then based on some metrics of the results obtained like the settling time, and maximum overshoot the weights can be updated.

6.2.2 Neural network shape and structure

The current neural network uses 2 hidden layers of equal size. The number of hidden layers is probably enough for the pure pitching problem, but the sudden increase in the neuron size from 3 to 300 may cause some difficulty in the gradient flow. This increase can be smoothed out to see if it affects the results.

6.2.3 Elevator modeling

Another aspect that was not looked into appropriately in the project is the dynamics of the elevator. The initial idea was to develop a controller for the ideal conditions where the saturation, and rate saturation of the elevator are ignored. Once a good controller for the ideal conditions has been eshtablished it can be applied to along with the physical restrictions of the elevator. But making the elevator more realistic could help the agent in finding an optimal policy more easily as the action space is restricted when real conditions are imple-

mented.

6.2.4 SAC and A3C using the current network properties

Using the second network improved the functioning of the agent to a great extent. Hence SAC and A3C agents can be implemented with this network and we can expect some improvement.

6.2.5 Modification in the problem representation

As θ was taken in degrees in the reward function, even the actor can be trained to predict the elevator angle in degrees. For the same reasons why the reward function θ representation change improved the functioning, it can be hoped that there will be some improvement with this too. Changing the way the governing equations are written can also be changed, for instance, the whole formulation can be done with the variables being the errors in θ and q.

Bibliography

[1] Philip Becker-Ehmck, Maximilian Karl, Jan Peters, and Patrick van der Smagt, "Learning to Fly via Deep Model-Based Reinforcement Learning" https://doi.org/10.48550/arXiv.2003.08876

[2] D. Kroezen, "Online Reinforcement Learning for Flight Control", https://doi.org/10.2514/6.2020-1844

[3] Feng Liu 1, Shuling Dai and Yongjia Zhao, "Learning to Have a Civil Aircraft Take Off under Crosswind Conditions by Reinforcement Learning with Multimodal Data and Preprocessing Data", https://doi.org/10.3390/s21041386

[4] Emil Andreas Lund, "Path Following in Simulated Environments using the A3C Reinforcement Learning Method", http://hdl.handle.net/11250/2563040

[5] Willian Koch, Renato Mancuso, Richard West and Azer Bestavros, "Reinforcement Learning for UAV Attitude Control", https://doi.org/10.1145/3301273

[6] Timothy P. Lillicrap, Jonathan J. Hunt, Alexander Pritzel, Nicolas Heess, Tom Erez, Yuval Tassa, David Silver and Daan Wierstra, "Continuous Control with Deep Reinforcement Learning", https://doi.org/10.48550/arXiv.1509.02971

[7] Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Tim Harley, Timothy P. Lillicrap, David Silver, Koray Kavukcuoglu, "Asynchronous Methods for Deep Reinforcement Learning", https://doi.org/10.48550/arXiv.1602.01783

[8] Tuomas Haarnoja, Aurick Zhou, Pieter Abbeel, Sergey Levine, "Soft Actor-Critic: Off-Policy Maximum Entropy Deep Reinforcement Learning with a Stochastic Actor", https://doi.org/10.48550/arXiv.1801.012904

[9] https://www.researchgate.net/publication/335018802_Intelligent_trajectory_tracking_of_an_aircraft_in_the_presence_of_internal_and_external_disturbances/figures?lo=1

[10] https://towardsdatascience.com/understanding-actor-critic\-methods-931b97b6df3f

[11] https://theaisummer.com/Actor_critics/

[12] https://www.youtube.com/watch?v=OcIx_TBu90Q

[13] https://opensourcelibs.com/libs/a3c

[14] https://blog.tensorflow.org/2018/07/deep-reinforcement-learning-keras-eager-execution.html

[15] https://github.com/germain-hug/Deep-RL-Keras/tree/master/A3C

[16] https://github.com/stefanbo92/A3C-Continuous/blob/master/a3c.py

[17] http://inoryy.com/post/tensorflow2-deep-reinforcement-learning/

[18] https://towardsdatascience.com/soft-actor-critic-demystified\-b8427df61665

[19] http://bair.berkeley.edu/blog/2018/12/14/sac/

[20] https://gym.openai.com/

[21] https://keras-rl.readthedocs.io/en/latest/

[22] https://github.com/keras-rl/keras-rl

[23] https://towardsdatascience.com/deep-deterministic-\policy-gradient-ddpg-theory-and-implementation-747a3010e82f

[24] https://towardsdatascience.com/deep-q-learning-tutorial\-mindqn-2a4c855abffc#:~:text=Critically%2C%20Deep%20Q%2DLearning%20replaces,process%20uses%202%20neural%20networks.

[25] https://towardsdatascience.com/reinforcement-learning\
-w-keras-openai-dqns-1eed3a5338c

[26] https://github.com/keras-rl/keras-rl/blob/master/docs/
sources/agents/ddpg.md

[27] https://keras.io/examples/rl/ddpg_pendulum/

[28] https://subscription.packtpub.com/book/
data/9781838982546/3/ch03lvl1sec27/
\implementing-the-deep-deterministic-\
policy-gradient-algorithm-and-ddpg-agent

[29] https://github.com/CUN-bjy/gym-ddpg-keras/tree/master/
experiments

[30] https://www.youtube.com/c/MachineLearningwithPhil

[31] https://www.youtube.com/watch?v=FgzM3zpZ55o&list=
PLoROMvodv4rOSOPzutgyCTapiGlY2Nd8u

[32] https://www.youtube.com/channel/UCHXa4OpASJEwrHrLeIzw7Yg

[33] https://www.youtube.com/channel/UCNIkB2IeJ-6AmZv7bQ1oBYg

[34] https://towardsdatascience.com/the-idea-behind-actor-critics-and\
-how-a2c-and-a3c-improve-them-6dd7dfd0acb8

[35] https://datascience.eu/machine-learning/
synchronous-actor-critic-agents-a3c/

Appendix A

Packages used for the implementation

The software and libraries used in the implementation are given below. The implementation was done in python.

A.1 Gym-AI

Open AI Gym is a library that provides several classic environments to work with. It also has options to create our custom environments which were used for the study.

A.2 RL Keras

RL-Keras is a library based on Keras for the Implementation of Certain RL Agents. The available agents are DQN, DDPG, SARSA, CEM, and NAF.