5-2023

Near-Optimal Control of a Quadcopter Using Reinforcement Learning

Alberto Velazquez-Estrada
The University of Texas Rio Grande Valley

NEAR-OPTIMAL CONTROL OF A QUADCOPTER USING REINFORCEMENT LEARNING

A Thesis

by

ALBERTO VELAZQUEZ-ESTRADA

Submitted in Partial Fulfillment of the

Requirements for the Degree of

MASTER OF SCIENCE IN ENGINEERING

Major Subject: Mechanical Engineering

The University of Texas Rio Grande Valley

May 2023

NEAR-OPTIMAL CONTROL OF A QUADCOPTER USING REINFORCEMENT LEARNING

A Thesis
by
ALBERTO VELAZQUEZ-ESTRADA

COMMITTEE MEMBERS

Dr. Constantine Tarawneh
Co-Chair of Committee

Dr. Tohid Sardarmehni
Co-Chair of Committee

Dr. Horacio Vasquez
Committee Member

Dr. Qi Lu
Committee Member

Dr. Lei Xu
Committee Member

May 2023

ABSTRACT

Velazquez-Estrada, Alberto, Near-Optimal Control of a Quadcopter Using Reinforcement Learning. Master of Science in Engineering (MSE), May, 2023, 71 pp., 2 tables, 18 figures, references, 27 titles.

This paper presents a novel control method for quadcopters that achieves near-optimal tracking control for input-affine nonlinear quadcopter dynamics. The method uses a reinforcement learning algorithm called Single Network Adaptive Critics (SNAC), which approximates a solution to the discrete-time Hamilton-Jacobi-Bellman (DT-HJB) equation using a single neural network trained offline. The control method involves two SNAC controllers, with the outer loop controlling the linear position and velocities (position control) and the inner loop controlling the angular position and velocities (attitude control). The resulting quadcopter controller provides optimal feedback control and tracks a trajectory for an infinite-horizon, and it is compared with commercial optimal control software. Furthermore, the closed-loop controller can control the system with any initial conditions within the domain of training. Overall, this research demonstrates the benefits of using SNAC for nonlinear control, showing its ability to achieve near-optimal tracking control while reducing computational complexity. This paper provides insights into a new approach for controlling quadcopters, with potential applications in various fields such as aerial surveillance, delivery, and search and rescue.

DEDICATION

This thesis is dedicated to my loving family who has encouraged me throughout my education. To my father, Tiburcio Velazquez, and my mother, Celia Velazquez, I am forever grateful for your love, encouragement, and sacrifices. To my brothers, thank you for always being there for me.

ACKNOWLEDGMENTS

I want to express my gratitude to Dr. Tohid Sardarmehni for introducing me to this field of research and guiding me throughout my master's degree. Your support, guidance, and mentorship have shaped my academic and professional journey. Your expertise, patience, and encouragement during the research were critical to my success. Thank you for being a great mentor and inspiring me to pursue excellence in everything I do.

I am grateful to Dr. Constantine Tarawneh for the support and availability during my research. I could not have completed this journey without your help. Your dedication to excellence and passion for research has been a source of inspiration, and I am thankful for your positive impact on my academic and personal growth.

I want to thank my friends who have been a constant source of support and encouragement throughout my research. I am grateful for the time you took to provide feedback, discuss ideas, and help me better understand complex topics. Your friendship and encouragement have inspired me to reach for excellence, and I could not have done it without you.

TABLE OF CONTENTS

Page

LIST OF TABLES

Page

LIST OF FIGURES

Page

CHAPTER I

BACKGROUND AND INTRODUCTION

1.1 Reinforcement Learning Overview

Reinforcement learning (RL) is a paradigm of machine learning that focuses on the learning process of an agent interacting with its environment to achieve a specific objective. RL aims to train an agent using feedback from the environment to perform actions that maximize a cumulative reward. RL is inspired by the natural learning methods of humans and animals, where rewards or punishments reinforce good or bad actions. An agent learns from interaction by exploring the environment and adjusting its behavior based on the rewards or penalties it receives.

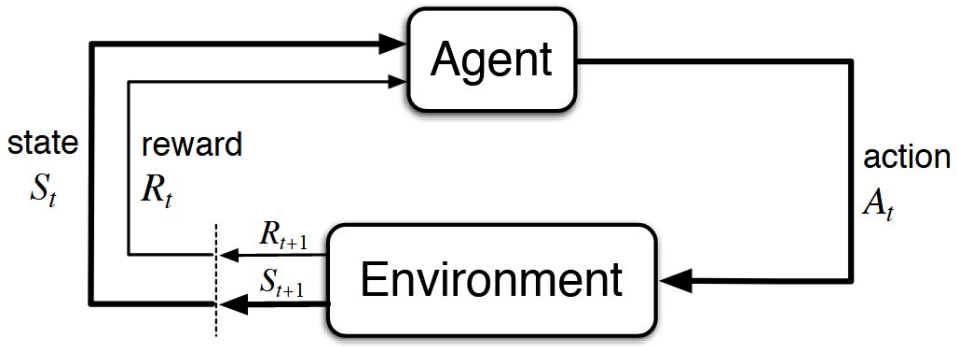

Figure 1: Agent interaction with the environment (Sutton, 2018).

The Markov Decision Process (MDP) is a mathematical framework used in reinforcement learning that models the agent's decision-making process in a stochastic environment. At each discrete time step t, the agent observes the current state S_t of the

environment and selects an action A_t based on the observed state. The time step advances and the agent receives a numeric reward, R_{t+1}, and is in the next set of states, S_{t+1}, due to the propagation of the action. Figure 1 shows the discrete-time MDP. The mapping between the state and action is called the policy. In RL, the objective is to improve the policy to maximize the cumulative reward, called value. This requires the policy to be forward-looking such that the agent may receive lower rewards in the short term but higher rewards in the long term.

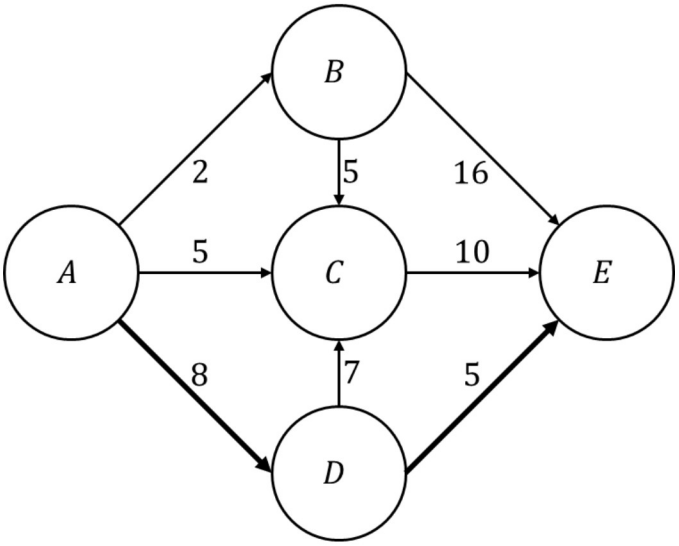

Figure 2: Principle of optimality.

Reinforcement learning is closely related to optimal control theory (Sutton, 2018). In both, an agent or controller selects actions or generates controls that maximize a performance measure. In optimal control, the objective is to determine an optimal state-action mapping policy that maximizes a performance measure. This optimal policy is defined using Bellman's optimality principle, which states that an optimal policy must be optimal for any initial state and all subsequent states (Bellman, 1965). Figure 2 illustrates a simple system to travel from the starting point, A, to the endpoint, E, while minimizing the cost. The numbers indicate the cost of traveling between the nodes. In this case, the optimal policy would be to select the path ADE,

which has a total cost of 13. However, if the starting point was B instead of A, the optimal policy would select path BCE, with a total cost of 15. The principles of optimal control are easily extended to reinforcement learning, with Bellman's optimality principle serving as the foundation for optimal control and RL algorithms. In control synthesis, the reinforcement learning terms agent, environment, and action are interchangeable with the terms controller, plant dynamics, and control signal. The controller is the agent responsible for making decisions, the plant dynamics represent the environment that the controller interacts with, and the control signal is the action that the controller takes in response to the current state of the plant dynamics.

In order to understand popular reinforcement learning algorithms, it is helpful to understand the foundational algorithms of optimal control theory. To find an optimal control, the minimum principle of Pontryagin or dynamic programming (DP) can be used (Kirk, 2004). Pontryagin's minimum principle solves a Hamiltonian function using variational calculus, while DP is a recursive, backward-in-time method that explicitly calculates and tabulates cost. Dynamic programming is vital in understanding popular reinforcement learning algorithms such as actor-critic (AC) and their variations.

1.2 Dynamic Programming

Dynamic programming is a backward-in-time, recursive method that satisfies the conditions of Bellman's principle of optimality. To apply DP to continuous-time systems, the system must first be discretized in time, and the states and controls must be quantized between allowable values. DP then finds the optimal policy by explicitly calculating the immediate cost of all quantized controls for all quantized states (Kirk, 2004). For each state, the control that results in the lowest cost, and the cost itself, are stored in a table or cache, as well as the next state that results from the control propagation along the plant dynamics. This process proceeds

recursively, backward in time, so the future cost is always known. The optimal policy is the control law that minimizes the cumulative cost at any state and time. At any step between the final and initial time, dynamic programming solves the immediate cost by trying all controls at all states and calculating the resulting state using knowledge of the system's dynamics. If the resulting state falls outside the allowable states, it is discarded. The optimal control policy is the control law that minimizes the cost-to-go, or the cumulative cost, from any state in the current time to the final time, so the immediate cost of the current step is added to the immediate cost of all remaining steps. This information is pulled from memory, or a table, based on the resulting state and interpolating if necessary. Since the optimal policy is found for every allowable state, the resulting control law is a robust, closed-loop control method. If uncertainties or noise in the system push the states off the optimal path, a new optimal path can be determined from the new states based on saved data.

1.2.1 Curse of Dimensionality

As the state-space dimensions, which define the allowable range of states, and control-space dimensions, which define the allowable range of controls, of a dynamic problem grow, the required computation and memory requirements to solve for an optimal policy grow exponentially. This becomes a significant limitation of dynamic programming known as the curse of dimensionality (Powell, 2007). To solve for the optimal control of high-dimensional dynamic systems, function approximations using parameterized functions, such as neural networks, can simplify calculations for the policy and value functions.

1.3 Approximate Dynamic Programming

The use of function approximation to estimate the optimal policy and value function without needing to try every possible state and control input is called Adaptive Dynamic

Programming (ADP). In ADP, a dual-network architecture called Adaptive Critic (AC), consisting of a critic and an actor network, is used to determine the optimal policy using the Hamiltonian-Jacobi-Bellman (HJB) equation (Konda, 1999). The HJB (Hamilton-Jacobi-Bellman) equation is a partial differential equation fundamental in optimal control and reinforcement learning. It describes the optimal value function, which represents the expected cumulative rewards an agent or controller can receive overtime when following an optimal policy. The actor network approximates the optimal policy, while the critic network estimates the optimal value function. The AC algorithm has its namesake as the actor networks acts by choosing a control/action given a set of states, and the critic network then judges the actor's performance by estimating the expected future value of the states and control variables. In the field of control, the actor-critic algorithm is referred to interchangeably as the adaptive-critic algorithm. The two networks are trained and updated iteratively; the actor network updates its policy function approximation based on the approximated value function in the critic network, which results in lower costs or increased rewards. The critic network then updates its value function approximation based on the feedback received from the actor network. This continues until the networks converge and provide the necessary and sufficient conditions for a near-optimal solution. Notably, the approximation of the value function in the critic network allows ADP to solve optimal control problems forward in time.

1.3.1 Online and Offline Training

ADP and RL algorithms are trainable online or offline. In online training, an agent or controller interacts with the environment and learns and updates its policy in real time. This training method is well-suited to applications with changes or uncertainties in the environment or system dynamics.

Offline training does not require real-time interaction with the environment, as an agent is trained using a set of sampled state, control, or reward data. If the data is sampled within a limited domain, the trained agent could operate optimally within that domain. However, if the sampled data does not accurately represent the real-world environment, the agent may behave suboptimally in unexpected scenarios. Furthermore, a changing or uncertain real-world environment can result in suboptimal behavior optimality or failure. Careful consideration must be given to the domain and environmental representation to ensure that the data accurately reflects the real-world conditions the agent will encounter.

1.3.2 Heuristic Dynamic Programming and Dual Heuristic Programming

The AC algorithm described previously is classified as a heuristic dynamic programming (HDP) algorithm. In the HDP formulation, the actor network maps between the state and control, while the critic network maps between the state and the value. A similar formulation is the dual-heuristic programming (DHP) class of algorithms. In this method, the actor network also maps between the state and control while the critic network maps between the state and gradient of the value function called the costate (Padhi, 2006). By estimating the costate, DHP achieves faster convergence as the rate of change of the value function is maximized. The costate estimation requires knowledge of the full plant dynamics, which provides more information about the system and aids in the convergence and approximation of the policy function (Lewis, 2009).

1.3.3 Single Network Adaptive Critic

Single Network Adaptive Critics (SNAC) is an improvement to the AC dual network class of DHP. SNAC uses a single neural network to approximate the next costate as opposed to the current costate of DHP. This eliminates the need for an actor network, reducing computational costs and eliminating the approximation error of the action network (Padhi, 2006).

Moreover, the critic network not only approximates the costate but also predicts it for the next time step, providing more information about the system and aiding in the convergence and approximation of the policy function. However, to calculate the next costate, knowledge of the full plant dynamics is, again, necessary (Lewis, 2009). Furthermore, the SNAC algorithm also requires the optimal control to be explicitly expressible in terms of the current state variable and the following costate variable, which would occur in input-affine systems with quadratic cost functions.

1.4 Quadcopter Control

Small unmanned aerial vehicles (UAVs) have become increasingly popular in recent years due to their versatility and cost-effectiveness. These aerial drones are used for various applications such as surveillance, reconnaissance, search and rescue, forestry, flood and fire tracking, package delivery, and agriculture [1–4]. Their compact size and expendability make them ideal for missions in dangerous or hard-to-reach areas without risking human life or requiring significant resources. As such, they have become valuable tools in various industries and have the potential to transform many aspects of modern life.

Multi-rotor unmanned aerial vehicles (UAVs) come in several designs. Quadcopters are the most commercially available and popular type of multirotor drone. They consist of four rotors arranged in a square configuration at an equal distance from the center of mass, with two rotating clockwise and two rotating counterclockwise. Generally, quadcopters are limited in their payload capacity and would become unstable in the event of a rotor failure; however, quadcopters are small, inexpensive, and readily available. On the other hand, hexacopters use six rotors, with three rotating clockwise and three rotating counterclockwise. They are typically used for more complex aerial photography or videography due to their ability to carry heavier loads

and provide better stability. Additionally, a hexacopter could continue operating in the event of a rotor failure. Finally, Octocopters use eight rotors, with four rotating clockwise and four rotating counterclockwise, making them the largest, most powerful, and most robust type of multirotor drone. They are commonly used for industrial applications that require heavy lifting.

Despite quadcopters having relatively simple hardware, they are nonlinear, underactuated dynamic systems with six degrees of freedom and four inputs. Several control strategies have been implemented to control quadcopter drones. PID controllers are the most commonly used controllers in commercially available quadcopter drones. PID and LQR controllers are usually limited in controlling simplified and linearized dynamics (Argentim, 2013; Bouabdallah, 2004). Feedback linearization is one method of deriving linear systems from nonlinear systems. In the case of a quadcopter with uninvertible control dynamics, feedback linearization requires small angle assumption to deal with repeated differentiation resulting in derivative terms that are sensitive to noise (Lee, 2009). Backstepping control is an example of a nonlinear control technique that can be applied to quadcopters. Backstepping controls divide the dynamic model into several subsystems that are stabilized using the Lyapunov theorem (Bouabdallah, 2005; Madani, 2006). These subsystems typically consist of the quadcopter's linear translation and the angular translation. However, variations in assumptions and derivation of the mathematical model can result in different subsystems between papers. Other nonlinear control techniques, such as sliding mode control, can be applied to a quadcopter (Bouabdallah, 2005). Sliding mode can control a system subject to disturbances, uncertainties, or modeling errors which can be useful in controlling uncertain quadcopter dynamics such as the ground effect (Lee, 2009). However, it can result in high-frequency oscillations in a system's state trajectory. Some studies have effectively used a combination of subsystems, like those used in backstepping controls, to

control the quadcopter using sliding mode control. (Xu, 2006). Additionally, quadcopter control has been achieved with a combination of backstepping and feedback linearization to control the nonlinear quadcopter dynamics (Das, 2009). More recent work in quadcopter control has implemented popular reinforcement learning algorithms such as Deep Deterministic Policy Gradient (DDPG), Trust Region Policy Optimization (TRPO), and Proximal Policy Optimization (PPO) to control the attitude subsystem responsible for stabilization and control (Koch, 2019).

1.5 Motivation and Contribution

In this paper, a novel control method for quadcopters that achieves near-optimal tracking control for input-affine nonlinear quadcopter dynamics is developed. The method uses the reinforcement learning algorithm called Single Network Adaptive Critics (SNAC), which approximates a solution to the discrete-time Hamilton-Jacobi-Bellman (DT-HJB) equation using a single neural network trained offline. The control method involves two SNAC controllers, with the outer loop controlling the linear position and velocities (position control responsible for navigation) and the inner loop controlling the angular position and velocities (attitude control responsible for stabilization). As such, the quadcopter dynamics are divided into position and attitude subsystems. The SNAC controllers require knowledge of the system dynamics, for those dynamics to be input-affine, and require a cost function that results in the optimal control being expressible in terms of the state and costate. The resulting quadcopter controller provides optimal feedback control and tracks a trajectory for an infinite-horizon, and it is compared with commercial optimal control software. Furthermore, the closed-loop controller can control the system with any initial conditions within the domain of training. Overall, this research demonstrates the benefits of using SNAC for nonlinear control, showing its ability to achieve near-optimal tracking control while reducing computational complexity. This paper provides

insights into a new approach for controlling quadcopters, with potential applications in various fields such as aerial surveillance, delivery, and search and rescue.

1.6 Related Work

The Single Network Adaptive Critic algorithm has demonstrated versatility in solving various problems across different fields, including control, healthcare, and environmental engineering. SNAC has been utilized to design optimal tracking control for the velocity subsystem of a hypersonic flight vehicle with uncertain dynamics (Bu, 2020), where fuzzy approximators were used to estimate the system's behavior and enable the SNAC formulation to control such systems. SNAC has also been employed to regulate blood glucose levels in diabetic patients; this required a mathematical model that describes the glucose and insulin interaction in the blood system (Ali, 2011). Additionally, SNAC has been applied to a quarter-vehicle active suspension system with parametric uncertainty and time-varying system parameters such as mass (Fu, 2017). The algorithm has also been utilized in the optimal tracking control of a morphing aircraft during a pull-up maneuver by modeling significant system uncertainties, such as the lift-curve slope and static longitudinal stability derivative, as time-varying quantities (Nobleheart, 2013). This highlights SNAC's adaptability compared to traditional controllers like LQR, which require time-invariant system parameters. Furthermore, the SNAC algorithm has been applied to manage the beaver population in a given area by utilizing a reduced-order distributed parameter model, resulting in effective management and maintenance of the population at a desired level through optimal control policies (Padhi, 2006).

The first use of reinforcement learning for quadcopter control was applied to an altitude subsystem in 2005 by Waslander. A model-based RL algorithm was used to search for an optimal control policy capable of handling disturbances, including blade flex, ground effect, and

battery discharge dynamics (Waslander, 2005). Further work into quadcopter control with reinforcement learning was done by Hwangbo (2017), where a model-free algorithm that forgoes the need to divide the dynamics into different subsystems was presented. The algorithm was a deterministic on-policy learning algorithm that outperformed Deep Deterministic Policy Gradient (DDPG) and Trust Region Policy Optimization (TRPO), popular RL algorithms, in computation time (Hwangbo, 2017). In 2019, Koch implemented several RL algorithms, including DDPG, TRPO, and Proximal Policy Optimization (PPO), to control the attitude subsystem responsible for stabilization and control (Koch, 2019).

In this paper, we propose a novel approach to quadcopter control by utilizing the SNAC RL algorithm to control both the position and attitude subsystems. This contrasts with other implementations of RL on quadcopter control, which often focus on controlling only one subsystem or use alternative algorithms such as sliding mode control for each subsystem. Furthermore, the control of the quadcopter is presented in a control framework, with the environment limited to the plant dynamics.

The paper is organized as follows: Chapter I presents an overview of quadcopter control and reinforcement learning and its various forms, including dynamic programming and approximate dynamic programming. Chapter II delves into the details of the SNAC algorithm and its application to quadcopter dynamics, explaining the discretization and cost functions used to solve the discrete-time Hamilton-Jacobi-Bellmen equation. It also provides an overview of the quadcopter dynamics, including reference frames and state-space dynamics, which is essential for developing effective control strategies. Chapter III discusses the control scheme used for position and attitude control of the quadcopter and the training of position and attitude regulators using the SNAC algorithm. The chapter provides a comprehensive understanding of the SNAC

algorithm's implementation in quadcopter control. Chapter IV presents the results of applying the SNAC algorithm to the quadcopter dynamics, including stabilization, comparison with DIDO, a commercial optimal control software, and the SNAC controller's performance with significant noise applied to the controls. The results show that the SNAC algorithm is an effective method for controlling the quadcopter, even in the presence of significant noise. Finally, Chapter V concludes the paper by summarizing the study's main contributions and suggests future work.

CHAPTER II

SNAC ALGORITHM AND QUADCOPTER DYNAMICS

2.1 SNAC

SNAC is an infinite-horizon algorithm introduced by Padhi in 2006. It approximates a discrete-time Hamilton-Jacobi-Bellman equation to determine the optimal policy and value functions. Using a single neural network, the SNAC algorithm finds the optimal control by mapping between the current state and the next costate. Once the SNAC algorithm is trained, it can provide online optimal feedback control to problems with varying initial conditions and for an infinite-time horizon (Padhi, 2006).

2.1.1 Discretization and Euler Integration

SNAC requires full knowledge of a system's dynamics and for those dynamics to be described as a continuous-time input-affine system, shown below

$$\dot{\mathrm{x}}(t) = f_c\big(x(t)\big) + g_c\big(x(t)\big)u(t) \tag{1}$$

where $x \in \mathbb{R}^n$ is the state vector, $u \in \mathbb{R}^m$ is the control vector, $f_c : \mathbb{R}^n \to \mathbb{R}^n$ is the continuous-time drift dynamics of the system, and $g_c : \mathbb{R}^n \to \mathbb{R}^{n\times m}$ is the continuous-time control dynamics of the systems. The integers n and m are the number of states and the number of controls, respectively. The system can be discretized using a small sample time, Δt, to yield the following equation.

$$x_{k+1} = F(x_k) + G(x_k)u_k, k \in \{0,1,\dots,N-1\} \tag{2}$$

where k represents the time step, $N = \frac{t_f}{\Delta t}$ is the total number of time steps, $x_k = x(k\Delta t)$, $u_k = u(k\Delta t)$, and $F(x_k) = x_k + \Delta t f_c(x_k)$and $G(x_k) = \Delta t g_c(x_k)$ are derived using Euler integration.

2.1.2 Cost Function and Discrete-time Hamilton-Jacobi-Bellmen Equation

For state regulation in SNAC, the following discrete-time cost function is defined.

$$\mathrm{J} = \frac{1}{2}(x_N)^T S(x_N) + \frac{1}{2}\sum_{k=0}^{N-1}((x_k)^T Q(x_k) + {u_k}^T R u_k) \tag{3}$$

where $S \in \mathbb{R}^{n\times n}$ is the terminal state error penalizing matrix and $Q \in \mathbb{R}^{n\times n}$ is the state error penalizing matrix. Both $S \in \mathbb{R}^{n\times n}$ and $Q \in \mathbb{R}^{n\times n}$ are positive semi-definite matrices. $R \in \mathbb{R}^{m\times m}$ is a positive definite matrix that penalizes the magnitude of the control input. The cost function represents the magnitude of the states x_k and the overall control effort expended with u_k. The magnitude of the matrices S, Q, and R determine the emphasis of the performance measure. Higher values in the matrices result in higher costs and vice versa. From the discrete-time cost function, the cost-to-go function, $\mathrm{J}(\mathrm{x_k}, \mathrm{k})$, at step k can be derived.

$$\mathrm{J}(\mathrm{x_k}, \mathrm{k}) = \frac{1}{2}(x_N)^T S(x_N) + \frac{1}{2}\sum_{\kappa=k}^{N-1}((x_\kappa)^T Q(x_\kappa) + {u_\kappa}^T R u_\kappa) \tag{4}$$

The equation shows the cost at the final time and the cost at each time step before the final time. This allows us to derive the following recursive equation.

$$\mathrm{J}(\mathrm{x_k}, \mathrm{k}) = \frac{1}{2}((x_\kappa)^T Q(x_\kappa) + {u_\kappa}^T R u_\kappa) + J(x_{k+1}, k+1), k \in \{0,1,\dots,N-1\} \tag{5}$$

This equation shows that the cost-to-go at any time step is the cost of the current time step plus the cost of all future time steps. Once the cost function is in a recursive notation, the following discrete-time HJB (DT-HJB) can be derived.

$$J^*(x_k, k) = \min\left(\frac{1}{2}((x_\kappa)^T Q(x_\kappa) + u_\kappa{}^T R u_\kappa) + J(x_{k+1}, k+1)\right), k \in \{0,1,\dots,N-1\} \quad (6)$$

$J^*(x_k, k)$ is the optimal cost-to-go, which minimizes the recursive cost-to-go equation.

2.1.3 Optimality Condition and Costate

In order to find the optimal control, the optimality condition must be met.

$$\frac{\partial J(x_k, k)}{\partial u_k} = 0 \quad (7)$$

Taking the partial derivative of the cost-to-go function with respect to the control results in the following equation.

$$\begin{aligned} Ru_\kappa + \frac{\partial J(x_{k+1}, k+1)}{\partial u_k} &= Ru_\kappa + \frac{\partial J(x_{k+1}, k+1)}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial u_k} \\ &= Ru_\kappa + \frac{\partial J(x_{k+1}, k+1)}{\partial x_{k+1}} g(x_k)^T = 0 \end{aligned} \quad (8)$$

The partial derivative, or gradient, of the cost-to-go function with respect to the state vector is called the costate.

$$\lambda_{k+1} = \frac{\partial J(x_{k+1}, k+1)}{\partial x_{k+1}} \quad (9)$$

The costate expands into the following.

$$\begin{aligned} \lambda_k = \frac{\partial J(x_k, k)}{\partial x_k} &= Q(x_k) + \frac{\partial J(x_{k+1}, k+1)}{\partial x_k} = Q(x_k) + \frac{\partial J(x_{k+1}, k+1)}{\partial x_{k+1}} \frac{\partial x_{k+1}}{\partial x_k} \\ &= Q(x_k) + \lambda_{k+1} \frac{\partial x_{k+1}}{\partial x_k} \end{aligned} \quad (10)$$

Where the gradient of the following states with respect to the current state is $A_k = \frac{\partial x_{k+1}}{\partial x_k}$

By combining the above equations, the optimality condition in Equation 8 can be written as

$$Ru_k + \lambda_{k+1} g(x_k) = 0 \tag{11}$$

from which the optimal control can be derived as

$$u_k^* = -R^{-1} \lambda_{k+1} g(x_k)^T \tag{12}$$

The costate, from Equation 9, at step $k+1$ can similarly be written as

$$\lambda_{k+1} = Q(x_{k+1}) + A_{k+1}^T \lambda_{k+2} \tag{13}$$

To train a SNAC controller effectively, it is crucial to use Equations 11 and 12 as they form the core equations for the controller. SNAC requires knowledge of the full system dynamics as the control dynamics are expressed in the control dynamics, $g(x_k)$, as seen in Equation 12.

2.1.4 Neural Network Approximator

SNAC uses a neural network (NN) that outputs the costate vector λ_{k+1} given the current state vector x_k. The NN form is shown below

$$\lambda_{k+1} = W^T \phi(x_k), k \in \{0,1, \dots, N-1\} \tag{14}$$

where $W_k \in \mathbb{R}^{m \times n}$ are the time-dependent weight matrix and $\phi : \mathbb{R}^n \to \mathbb{R}^m$ is a vector of smooth linearly-independent scalar basis functions.

To train the neural network's weights, least squares is used, as shown below

$$W_k^{i+1} = (\phi \phi^T)^{-1} \phi \left(\lambda^t \left(W_k^i \right) \right)^T \tag{15}$$

Notably, the training is iterative, with i showing the current iteration. The target costate, found with the costate equations, is a function of the previous iteration of the weights, with the first iteration being randomly initialized weights. During each iteration, the algorithm updates the weights to minimize the difference between the estimated and target costs. The training process continues iteratively until the error between the target costate and the estimated costate falls

below a preselected tolerance. This ensures that the neural network converges to an optimal solution for the control problem.

$$e_k(x_k) = \lambda_{k+1} - \lambda_{k+1}^t = W_k^t \phi(x_k) - \lambda_{k+1}^t \tag{16}$$

To train the neural network, the procedure is as follows (Padhi, 2006):

1. Randomly generate W
2. Randomly generate $x_k \in \Omega$ where $\Omega \subset \mathbb{R}^n$ is the domain of interest.
    a. Input x_k into the neural network, Equation 14, to generate λ_{k+1}.
    b. Calculate u_k using Equation 14, the optimal control equation.
    c. Calculate x_{k+1} from Equation 2, the state equation.
    d. Input x_{k+1} into the neural network, Equation 14, to generate λ_{k+2}.
    e. Calculate u_{k+1} using Equation 14, the optimal control equation.
    f. Use x_{k+1}, r_{k+1}, and λ_{k+2} in Equation 13, the costate equation, to find the target costate λ_{k+1}^t.
3. Train the neural network weights W using Equation 15, the least square equation, and input-target pair $\{x_k, \lambda_{k+1}^t\}$.
4. Calculate the training error $e_k(x_k)$ using Equation 16.
5. Iteratively repeat steps 2 to 4 until the weights W converge.

This process can be seen in the following diagram.

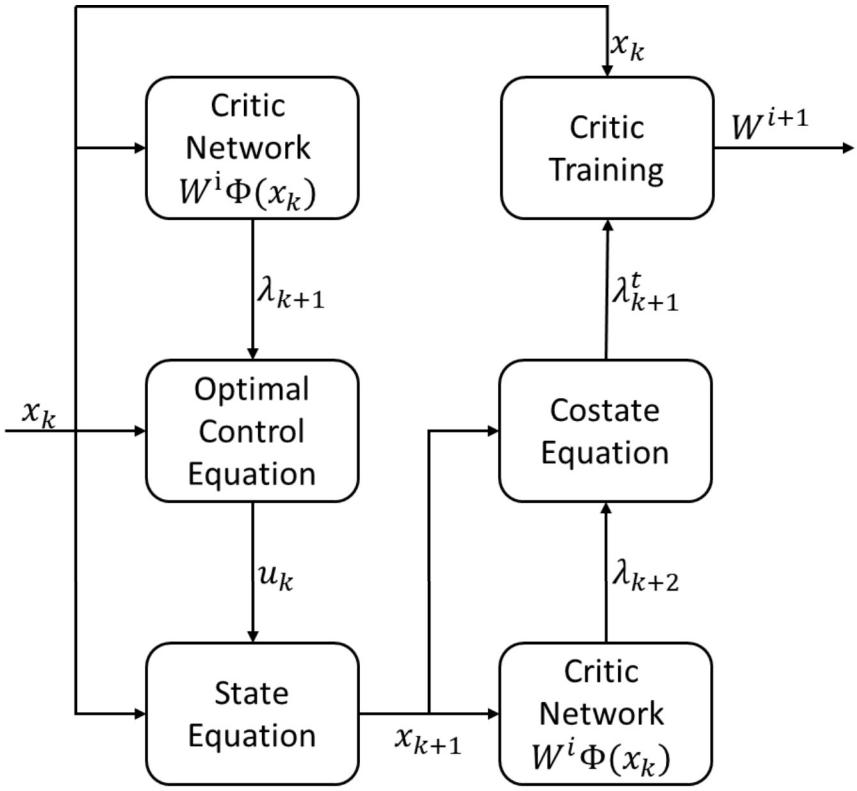

Figure 3: SNAC training diagram

2.2 Quadcopter Dynamics

2.2.1 Reference Frames

Two reference systems need to be related to describe the mathematical model of the quadcopter: the fixed earth frame and the mobile aircraft body frame. The fixed Earth frame uses the North-East-Down (O_{NED}) coordinate system, while the mobile aircraft body frame describes the Aircraft-Body-Center (O_{ABC}) coordinate system. Linear and angular positions are defined in the earth frame as the following vector: $[x\ y\ z\ \phi\ \theta\ \psi]^T$. Euler angles are used to describe the orientating of the quadcopter: ϕ describes the roll, θ describes the pitch, and ψ describes the yaw. In the aircraft body frame, linear and angular velocities are defined as: $[u\ v\ w\ p\ q\ r]^T$.

To relate the mobile aircraft body reference frame to the fixed earth reference frame, a combination of the following rotational matrices is used

$$R_x(\phi) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \sin(\phi) & \cos(\phi) \end{bmatrix} \tag{17}$$

$$R_x(\theta) = \begin{bmatrix} \cos(\theta) & 0 & \sin(\theta) \\ 0 & 1 & 0 \\ -\sin(\theta) & 0 & \cos(\theta) \end{bmatrix} \tag{18}$$

$$R_x(\psi) = \begin{bmatrix} \cos(\psi) & -\sin(\psi) & 0 \\ \sin(\psi) & \cos(\psi) & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{19}$$

Different combinations of rotational matrices can result in greatly simplified dynamics that omit the yaw ψ, such as $R_{xyz}(\phi, \theta, \psi)$. The $R_{zyx}(\phi, \theta, \psi)$ combination is used in this paper

$$R_{zyx}(\phi, \theta, \psi) = \begin{bmatrix} \mathrm{c}(\theta)\mathrm{c}(\psi) & s(\phi)s(\theta)\,\mathrm{c}(\psi) - c(\phi)s(\psi) & c(\phi)s(\theta)\,\mathrm{c}(\psi) + s(\phi)s(\psi) \\ \mathrm{c}(\theta)\mathrm{s}(\psi) & s(\phi)s(\theta)\,\mathrm{s}(\psi) + c(\phi)c(\psi) & c(\phi)s(\theta)\,\mathrm{s}(\psi) - s(\phi)c(\psi) \\ -s(\theta) & s(\phi)c(\theta) & c(\phi)c(\theta) \end{bmatrix} \tag{20}$$

Here $c()$ = cos() and $s()$ = sin().

The rotational matrix will be used to relate the derivative of the linear position and the linear velocities between the two reference frames. The following angular transformation matrix can be used to relate the derivative of the angular positions to the angular velocities in a similar manner

$$T(\phi, \theta) = \begin{bmatrix} 1 & \sin(\phi)\tan(\theta) & \cos(\phi)\tan(\theta) \\ 0 & \cos(\phi) & -\sin(\phi) \\ 0 & \dfrac{\sin(\phi)}{\cos(\theta)} & \dfrac{\cos(\phi)}{\cos(\theta)} \end{bmatrix} \tag{21}$$

The rotational matrix $R_{zyx}(\phi, \theta, \psi)$ and the translational matrix $T(\phi, \theta)$ are used to relate the fixed Earth and the mobile aircraft body frame. This can be done using the following

relations $v = Rv_B$ and $w = Tw_B$. Where $v = [\dot{x} \quad \dot{y} \quad \dot{z}]^T$, $v_B = [u \quad v \quad w]^T$, $w = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T$, and $w_B = [p \quad q \quad r]^T$ (Sabatino, 2015).

2.2.2 Newton and Euler Equations

The following process was outlined by Sabatino (2015). Newton's law can be used to find the total force applied to the quadcopter

$$f_B = m(w_B \times v_B + \dot{v}_b) \tag{22}$$

where m is the mass and $f_B = [f_x \quad f_y \quad f_z]$ is the total force. The total external force acting on the body frame can be given by

$$f_B = mgR^T\hat{e}_Z - f_t\hat{e}_3 \tag{23}$$

where g is gravity, $\hat{e}_Z$ is a unit vector along the global z-axis, $\hat{e}_3$ is a unit vector in the body-frame relative z-axis, and f_t is the total thrust.

Euler equations can be used to find the total torque acting on the quadcopter

$$m_B = I\dot{w}_B + w_B \times (Iw_B) \tag{24}$$

where I is a diagonal inertia matrix and $m = [m_x \quad m_y \quad m_z]$ is the total torque. The total external moments acting on the body frame are given by

$$m_B = \tau_B - g_a \tag{25}$$

where $\tau_B = [\tau_x \quad \tau_y \quad \tau_z]$ is the control torques generated by the quadcopter rotors, and g_a is the gyroscopic moments due to the rotors on the quadcopter.

The total thrust, f_t, and the control torques, $[\tau_x \quad \tau_y \quad \tau_z]$, can be defined as proportional to the squared speeds of the rotors (Bresciani, 2008, as cited in Sabatino, 2015)

$$\begin{aligned} f_t &= b(\Omega_1^2 + \Omega_2^2 + \Omega_3^2 + \Omega_4^2) \\ \tau_x &= bl(\Omega_3^2 - \Omega_1^2) \\ \tau_y &= bl(\Omega_4^2 + \Omega_2^2) \\ \tau_z &= d(\Omega_2^2 + \Omega_4^2 - \Omega_1^2 - \Omega_3^2) \end{aligned} \tag{26}$$

where b is the thrust factor, l is the distance between the center of the drone and a rotor, d is the drag factor, and Ω is the angular speed of the quadcopter's rotors. These factors are specific to the quadcopter and rotors.

By rewriting Newton's law as

$$m\dot{v} = Rf_B = mg\hat{e}_Z - f_t R\hat{e}_3 \tag{27}$$

and setting $[\dot{\phi} \quad \dot{\theta} \quad \dot{\psi}]^T = [p \quad q \quad r]^T$, an assumption that is true for small angles of movements (Das, 2009, as cited in Sabatino, 2015), the dynamics model of the quadcopter can be written as

$$\begin{aligned}
\ddot{x} &= -\frac{f_t}{m}[s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta)] \\
\ddot{y} &= -\frac{f_t}{m}[c(\phi)s(\psi)s(\theta) - c(\psi)s(\phi)] \\
\ddot{z} &= g - \frac{f_t}{m}[c(\phi)c(\theta)] \\
\ddot{\phi} &= \frac{I_y - I_z}{I_x}qr + \frac{\tau_x}{I_x} \\
\ddot{\theta} &= \frac{I_z - I_x}{I_y}pr + \frac{\tau_y}{I_y} \\
\ddot{\psi} &= \frac{I_x - I_y}{I_z}pq + \frac{\tau_z}{I_z}
\end{aligned} \tag{28}$$

2.2.3 State-Space Dynamics

The dynamic model of the quadcopter can be written in the state-space form to yield the following equation

$$\dot{x} = f(x) + g(x)u \tag{29}$$

$$\dot{x} = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{u} \quad \dot{v} \quad \dot{w} \quad \dot{\phi} \quad \dot{\theta} \quad \dot{\psi} \quad \dot{p} \quad \dot{q} \quad \dot{r}]^T \tag{30}$$

$$f(x) = \begin{bmatrix} u \\ v \\ w \\ 0 \\ 0 \\ g \\ p + q\big(s(\phi)t(\theta)\big) + r\big(c(\phi)t(\theta)\big) \\ q\big(c(\phi)\big) - r\big(s(\phi)\big) \\ q\dfrac{s(\phi)}{c(\theta)} + r\dfrac{\mathrm{c}(\phi)}{c(\theta)} \\ \dfrac{I_y - I_z}{I_x}qr \\ \dfrac{I_z - I_x}{I_y}pr \\ \dfrac{I_x - I_y}{I_z}pq \end{bmatrix} \tag{31}$$

$$g(x) = \begin{bmatrix} 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ -1/m(s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) & 0 & 0 & 0 \\ -1/m(c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta) & 0 & 0 & 0 \\ -1/m\big(c(\phi)s(\theta)\big) & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 1/I_x & 0 & 0 \\ 0 & 0 & 1/I_y & 0 \\ 0 & 0 & 0 & 1/I_z \end{bmatrix} \tag{32}$$

$$u = [f_t \quad \tau_x \quad \tau_y \quad \tau_z]^T \tag{33}$$

These state-space dynamics represent the drift and control dynamics that can be controlled by reinforcement learning algorithms such as SNAC.

CHAPTER III

EXPERIMENTAL SETUP

3.1 Control Scheme

To effectively control the underactuated and nonlinear dynamics of a quadcopter, two SNAC controllers are required. The quadcopter's dynamics are highly coupled and nonlinear, making it difficult to control. To control the linear position $[x \quad y \quad z]^T$, it is necessary to solve for the corresponding linear velocities $[u \quad v \quad w]^T$, simultaneously. This requires the determination of the total thrust produced by the quadcopter, f_t, and the angular position $[\phi \quad \theta \quad \psi]^T$, which is in turn determined by the angular velocities $[p \quad q \quad r]^T$. The control torques $[\tau_x \quad \tau_y \quad \tau_z]^T$ produced by the quadcopter's rotors are necessary to determine the angular velocities. The highly coupled angle dynamics make quadcopter control using a single SNAC control loop challenging. However, splitting the dynamics into a position loop and an attitude (angle) loop allows for the successful control of the quadcopter system.

3.1.1 Position Control

The position control splits the dynamics into the linear position, $[x \quad y \quad z]^T$, and linear velocities, $[u \quad v \quad w]^T$, resulting in simple, easily controllable dynamics, as shown below.

$$\dot{x}_p = f_p(x_p) + g_p(x_p)u_p \tag{34}$$

$$\dot{x}_p = [\dot{x} \quad \dot{y} \quad \dot{z} \quad \dot{u} \quad \dot{v} \quad \dot{w}\,]^T \tag{35}$$

$$f_p(x_p) = \begin{matrix} u \\ v \\ w \\ 0 \\ 0 \\ g \end{matrix} \tag{36}$$

$$g_p(x_p) = \begin{bmatrix} 1 & 0 & 0 \\ 0 & 1 & 0 \\ 0 & 0 & 1 \end{bmatrix} \tag{37}$$

$$u_p = [u_x \quad u_y \quad u_z]^T \tag{38}$$

where

$$\begin{aligned} u_x &= -\frac{f_t}{m}(s(\phi)s(\psi) + c(\phi)c(\psi)s(\theta) \\ u_y &= -\frac{f_t}{m}(c(\psi)s(\phi) - c(\phi)s(\psi)s(\theta) \\ u_z &= -\frac{f_t}{m(c(\phi)s(\theta))} \end{aligned} \tag{39}$$

The SNAC algorithm will control the position dynamics by regulating the states to zero, as defined by the cost function. By defining a reference trajectory, $[r_x \quad r_y \quad r_z]^T$, the difference between the current states and the reference states also be regulated, resulting in trajectory tracking. The controller will generate the inputs $[u_x \quad u_y \quad u_z]^T$, which are a function of the total thrust and the Euler angles $[f_t \quad \phi \quad \theta \quad \psi]$. The control inputs represent a 3-equation, 4-unknown system.

3.1.2 Euler Angle and Thrust Approximation

In order to solve this system of equations, an additional variable is required. The yaw angle, ψ, is selected as a reference input as it allows the quadcopter to face any direction. The

system can be solved using any number of system solvers. Numerically solving the system is the most accurate approach, with a function such as MATLAB's 'fsolve' iterating upon an initial guess until convergence. This approach is time-consuming in systems discretized by a small sample time, but it can account for nonlinearities in the equations. Analytically solving for the Euler angles is possible but can be prohibitively complex and computationally expensive. The equations are nonlinear and sinusoidal, requiring significant time and computational resources to find analytical solutions to the system. Furthermore, MATLAB functions such as 'solve' return possibly spurious solutions that lack the accuracy of the numerical solutions due to approximations and assumptions in their derivations. Finally, neural networks can also be used to map between the inputs $[u_x \quad u_y \quad u_z \quad \psi]$ and the outputs $[f_t \quad \phi \quad \theta]$. While the neural network is simple and computationally inexpensive to use once it has been trained, it may not be as accurate as numerical or analytical system solver methods.

Solving the system of equations yields the total thrust, f_t, and the Euler angles ϕ, and θ. The Euler angles are used as a reference trajectory $[r_\phi \quad r_\theta \quad r_\psi]^T$ in the attitude control.

3.1.3 Attitude Control

Once a reference signal for the attitude control has been generated, they are passed into a SNAC regulator trained on the following attitude dynamics.

$$\dot{x_a} = f_a(x_a) + g_a(x_a)u_a \tag{40}$$

$$\dot{x}_a = [\dot{\phi} \quad \dot{\theta} \quad \dot{\psi} \quad \dot{p} \quad \dot{q} \quad \dot{r}]^T \tag{41}$$

$$f_a(x_a) = \begin{bmatrix} p + q\big(s(\phi)t(\theta)\big) + r\big(c(\phi)t(\theta)\big) \\ q\big(c(\phi)\big) - r\big(s(\phi)\big) \\ q\dfrac{s(\phi)}{c(\theta)} + r\dfrac{c(\phi)}{c(\theta)} \\ \dfrac{I_y - I_z}{I_x}qr \\ \dfrac{I_z - I_x}{I_y}pr \\ \dfrac{I_x - I_y}{I_z}pq \end{bmatrix} \tag{42}$$

$$g_a(x_a) = \begin{bmatrix} 1/I_x & 0 & 0 \\ 0 & 1/I_y & 0 \\ 0 & 0 & 1/I_z \end{bmatrix} \tag{43}$$

$$u_a = [\tau_x \quad \tau_y \quad \tau_z]^T \tag{44}$$

The SNAC algorithm will regulate the Euler angle and angular velocities. Angle tracking can be achieved using the angle reference signal generated by the system solver.

3.1.4 Quadcopter Control Summary

Once the position and attitude of the SNAC controller are trained, they can be used to control the quadcopter. First reference inputs for the position and yaw are required. The difference between the current and desired positions is passed into the SNAC position regulator. This controller generates the necessary acceleration. These accelerations are a function of thrust, roll, pitch, and yaw. To determine these variables, a system solver in the form of a neural network or a numerical solver takes the acceleration and reference yaw as input and generates the resulting thrust, roll, and pitch. The roll, pitch, and yaw represent the required angles that allow the quadcopter to move from its current position to the desired one. These reference angles are then compared with the current angles, and the error is passed into the trained SNAC attitude

regulator. The regulator outputs the torques required to reach the angular rates that correspond to the desired angles. The following diagram shows a summary of the control scheme used to control a quadcopter.

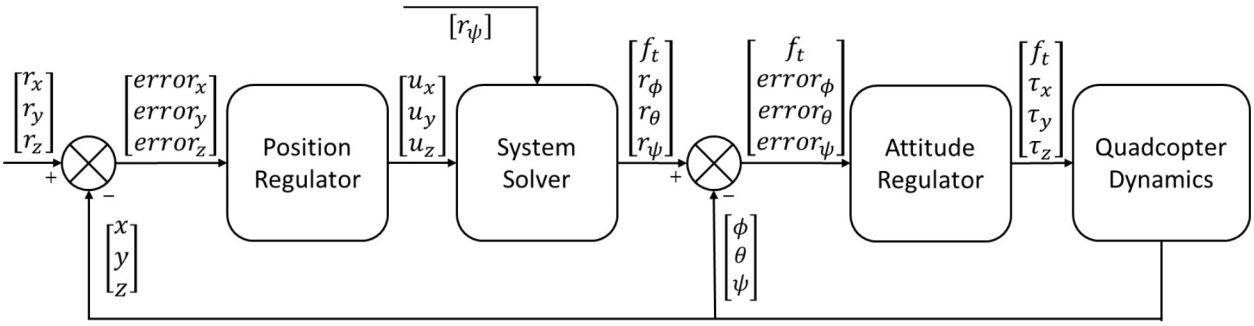

Figure 4: Quadcopter control diagram.

In the reinforcement learning framework, the Position and Attitude regulators are two agents that interact with the quadcopter dynamics environment and minimize a cost function. These agents are trained by sampling a state-space within a domain where the quadcopter is expected to operate. Within this domain, the agents are able to effectively generate actions that would regulate the states to zero, as determined by the reward function. As the error is input into these agents, the error is regulated to zero, resulting in trajectory tracking.

3.2 Position Regulator

The position SNAC controller was trained with the parameters outlined in Table 1. The number of basis functions can be selected to reduce approximation errors in the neural network. Given the simplicity of the position dynamics, the number of basis functions can functionally be much lower. This can be determined by the value at which the weights converge. If the value is near zero, the weight has little impact on the overall network accuracy. Despite this, there was little incentive to lower the number of basis functions for the position dynamics as the SNAC algorithm requires insignificant amounts of computation, with the network converging in less

than a minute. Furthermore, the resulting storage size of the neural network weights was negligible at 4 KB. As the neural network was trained using least squares, the number of patterns must be equal to or greater than the number of basis functions and be linearly independent such that the inverse matrix of Equation 15 exists. The max training loop was arbitrarily selected as 40,000 iterations. This would allow the controller ample time to converge or otherwise train for a sufficiently long time so that the resulting neural network would be effective. In the event that the network does not converge, the maximum training loop can be increased further. Similarly, the convergence threshold was selected so that the error between the target and actual neural network output, the costate, would be sufficiently low at convergence.

For the position control, a time step of 1 millisecond was selected to achieve high-frequency sampling and generate controls at a fast rate, resulting in smooth trajectory tracking. Although this works well in a simulation environment, sensor readings may have a higher frequency than one millisecond in practical applications, such as 2.5 milliseconds. However, due to the simplicity of the position dynamics, the position SNAC algorithm can have a higher time step, even up to 10 milliseconds, without significant loss of accuracy. The state penalizing matrix emphasizes tracking the trajectory in the xy-direction over the z-direction and velocities. This was selected as the SNAC controller prefers tracking in the z-direction over the xy-directions. The control penalizing matrix was selected to keep the controls sufficiently smooth while allowing for tracking. Finally, the domain of training for the position controller was chosen to be within a range of -100 meters to 100 meters, while for the velocity controller, it was set within a range of -100 m/s to 100 m/s. These ranges were chosen to ensure the trained controller could handle various scenarios.

Table 1: Parameters used to train the position SNAC controller.

Position Control Parameters	
Number of Patterns	100
Number of Basis Functions/Neurons	84
Max Training Loop	40000
Convergence Threshold	1e-5
Time Step dt	0.001 seconds
State Penalizing Matrix Q	$\begin{bmatrix} 1e6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e5 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e5 \end{bmatrix}$
Control Penalizing Matrix R	$\begin{bmatrix} 0.5e4 & 0 & 0 & 0 \\ 0 & 0.5e4 & 0 & 0 \\ 0 & 0 & 0.5e4 & 0 \\ 0 & 0 & 0 & 0.5e4 \end{bmatrix}$
Domain of Training	Position (x,y,z) range: [-100 100] (m) Velocity (u,v,w) range: [-100 100] (m/s)

3.3 Attitude Regulator

The attitude SNAC controller was trained with the parameters outlined in Table 2. As the attitude dynamics are more intensive, the number of basis functions was selected to reduce inaccuracies in the trained network; however, inspecting the weights shows that no weights converged near zero. This indicates the need for additional basis functions to accurately map between the input states and the next constates. The number of patterns was selected for the same reasons outlined in the position control parameters. The max training loop was also

selected as 40,000 iterations, and the convergence threshold was selected to reduce network error. Given the complexity of the attitude dynamics, the SNAC algorithm required more time to train than the position controller, with the algorithm reaching the max training loop before convergence. Several factors can improve training performance, including increasing the number of patterns to provide a richer number of samples, increasing the max training loop so that the network may further iterate and improve, and limiting the domain of training based on expected ranges. In effect, all these factors provide additional and denser information to the neural network about the domain in which the system will operate. The current domain of training was selected with those reasons in mind.

The attitude SNAC controller was trained with a time step of 1 millisecond. The same sensor sampling frequency issues would arise with the attitude controller as those outlined in the position controller. As opposed to the position controller, the attitude controller contains dynamics responsible for the quadcopter's stability. The frequency of sampling can have a significant effect on the stability and controllability of a quadcopter. Low-frequency sampling would result in slower control generation and instability; therefore, training and simulating the attitude controller to the base frequency of sensors would provide more realistic values. With this in mind, the SNAC controller was trained successfully with a 2.5-millisecond timestep. With a training time step of 10 milliseconds position controller and a time step of 2.5 milliseconds in the attitude controller, it would be possible to create a new control scheme with a slow navigation outer loop and a fast stabilization inner loop.

The state penalizing matrix was selected to track the roll, pitch, and yaw equally. Figure 8 shows the SNAC tracking of the angles. The algorithm tracks ϕ and ψ well, but the tracking of θ can improve. Modifying the state penalizing matrix to emphasize the tracking on θ would

improve the results; however, a different reference signal might produce different results in tracking. The control penalizing matrix limits the magnitude of the controller and ensures smooth signals. If these values were sufficiently high, the controller would prioritize minimizing control signals rather than tracking.

Table 2: Parameters used to train the attitude SNAC controller.

Attitude Control Parameters	
Number of Patterns	100
Number of Basis Functions/Neurons	84
Max Training Loop	40000
Convergence Threshold	1e-5
Time Step dt	0.001 seconds
State Penalizing Matrix Q	$\begin{bmatrix} 1e6 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1e6 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1e6 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1e5 & 0 & 0 \\ 0 & 0 & 0 & 0 & 1e5 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1e5 \end{bmatrix}$
Control Penalizing Matrix R	$\begin{bmatrix} 0.5e4 & 0 & 0 & 0 \\ 0 & 0.5e4 & 0 & 0 \\ 0 & 0 & 0.5e4 & 0 \\ 0 & 0 & 0 & 0.5e4 \end{bmatrix}$
Domain of Training	Euler Angle (ϕ, θ, ψ) range: [-1 1] (rad) Angular Velocity (p,q,r) range: [-1 1] (rad/s)

CHAPTER IV

RESULTS AND DISCUSSION

The SNAC controller is a closed-loop control method that enables feedback control and allows for controlling a system with any initial condition within the defined domain of training. Additionally, SNAC is an infinite-horizon algorithm capable of controlling systems for any length of time and defined reference signal. This would allow two individual controllers to effectively track a trajectory, with one controller focused on waypoint navigation and the other on quadcopter stabilization.

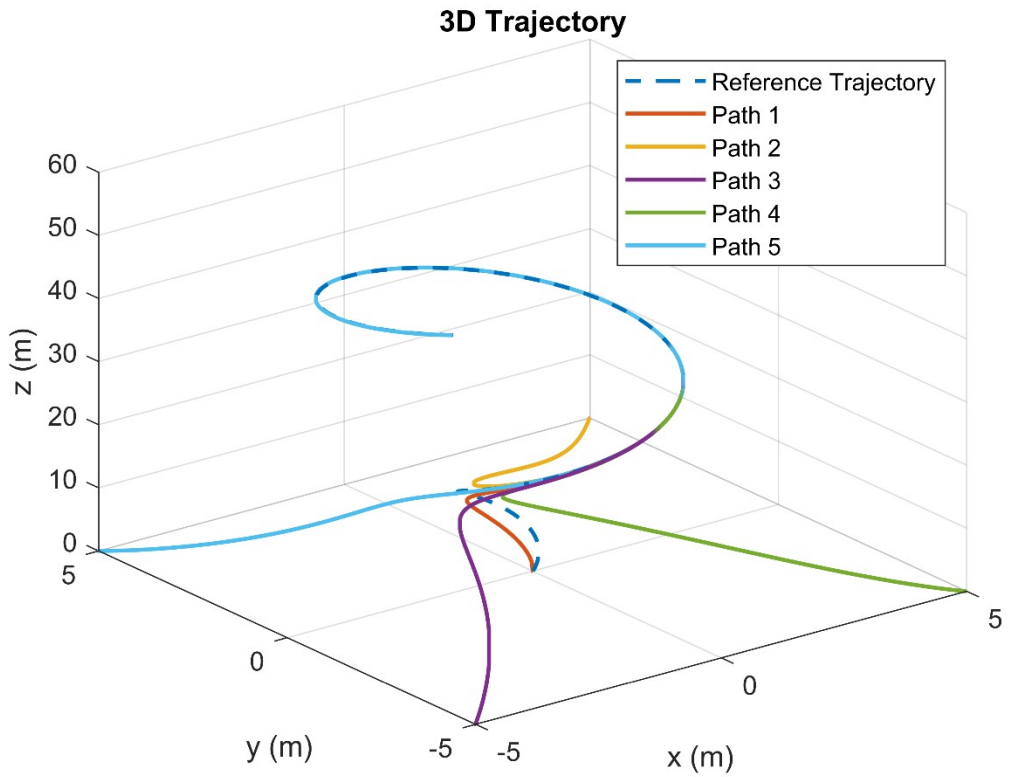

Figure 5: 3D helix trajectory tracking of the full quadcopter controller.

An example of the SNAC quadcopter controllers tracking a trajectory over 50 seconds with several initial conditions can be seen in Fig. 5. The trajectory is a helix with an initial starting point at zero. The initial conditions of all the quadcopter paths assume an initial velocity, initial angular position, and initial angular velocity of zero for all directions. The initial position in the xy-plane is arbitrarily chosen; however, the limiting factor is the domain of training of the linear position and velocities.

4.1 Position Control

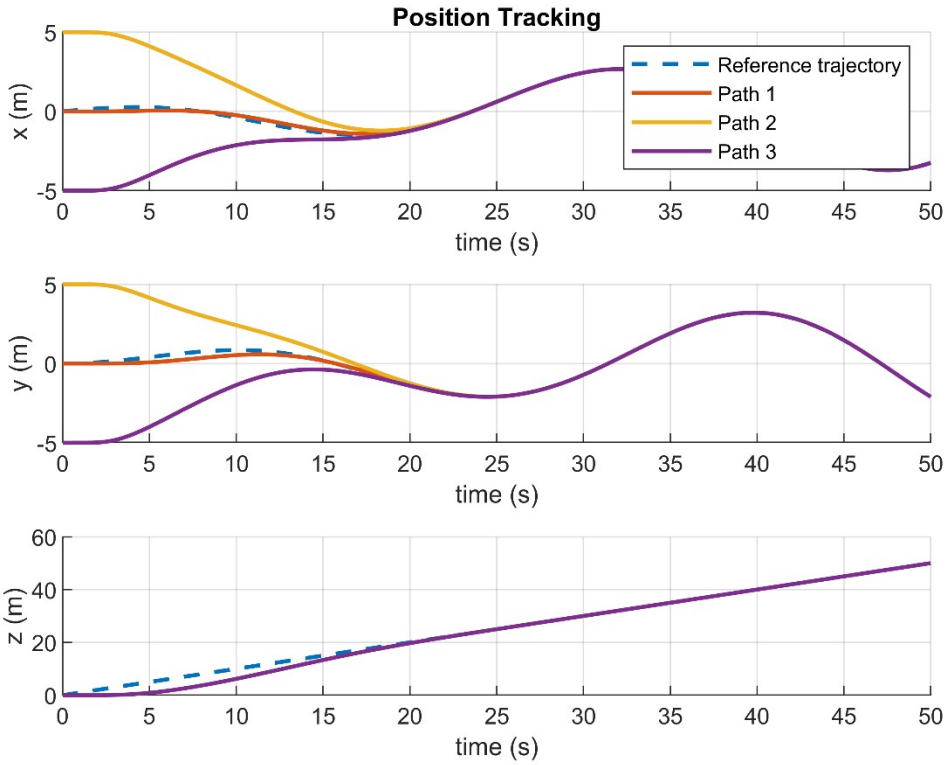

Figure 6: SNAC tracking of the XYZ position in the Position Control.

The simulation results for the position and velocity tracking are presented in Fig. 6 and Fig. 7, respectively. Figure 6 displays the position tracking of the helical trajectory with various initial conditions in the xy-coordinates while the remaining states had initial values of zero. The paths converge to the trajectory after 18 seconds, with the delay attributed to the quadcopter's initial takeoff, during which it stabilizes. Afterward, the quadcopter path closely follows the helical trajectory. The reference trajectory was generated as a parametric function with time as the input to provide a reference signal for the quadcopter to track at each time step. Another approach would be to generate waypoints, have the controller guide the quadcopter to each point, and maintain it until a new waypoint is provided.

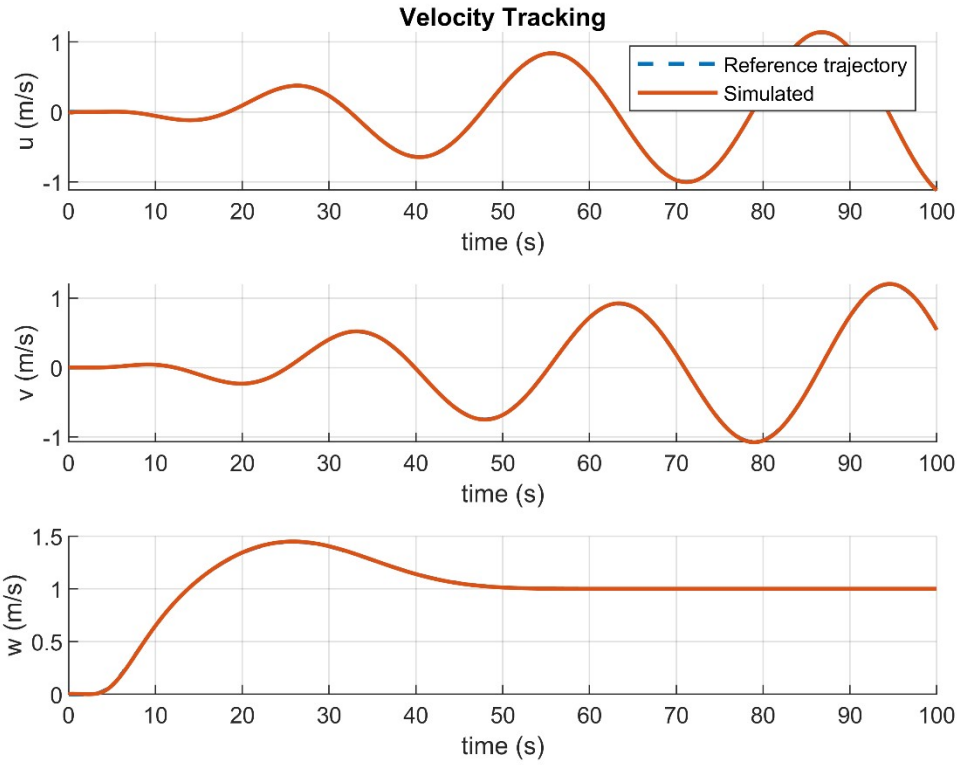

Figure 7: SNAC tracking of the velocity in the Position Control.

Figure 7 displays the velocity tracking of the quadcopter path, where the reference signal for velocity was generated by taking the numeric derivative of the position. The quadcopter started with zero initial conditions, and the velocity trajectory shows the quadcopter increasing its climbing rate initially to catch up with the reference trajectory. The climbing rate, w, then stabilizes as the quadcopter tracks the reference trajectory.

4.2 Attitude Control

The simulation results for the position and velocity tracking are presented in Fig. 8 and Fig. 9, respectively. Figure 8 displays the Euler angle tracking of the angular trajectory generated by the system solver. The initial conditions for the angles and angular velocities were selected as zero. The angles are tracked effectively with a more significant error in the pitch, θ, due to the state penalizing matrix. In order for the quadcopter to track the helical trajectory outlined so far, angles below 0.01 rad are sufficient. This is due to the large and smooth diameter of the helical trajectory. Different trajectories can be simulated to show more significant and more demanding angular positions. Notably, the yaw, ψ, is selected as zero. This keeps the quadcopter facing a single direction throughout the duration of the flight. This value was selected for simplicity, with the yaw being a reference input.

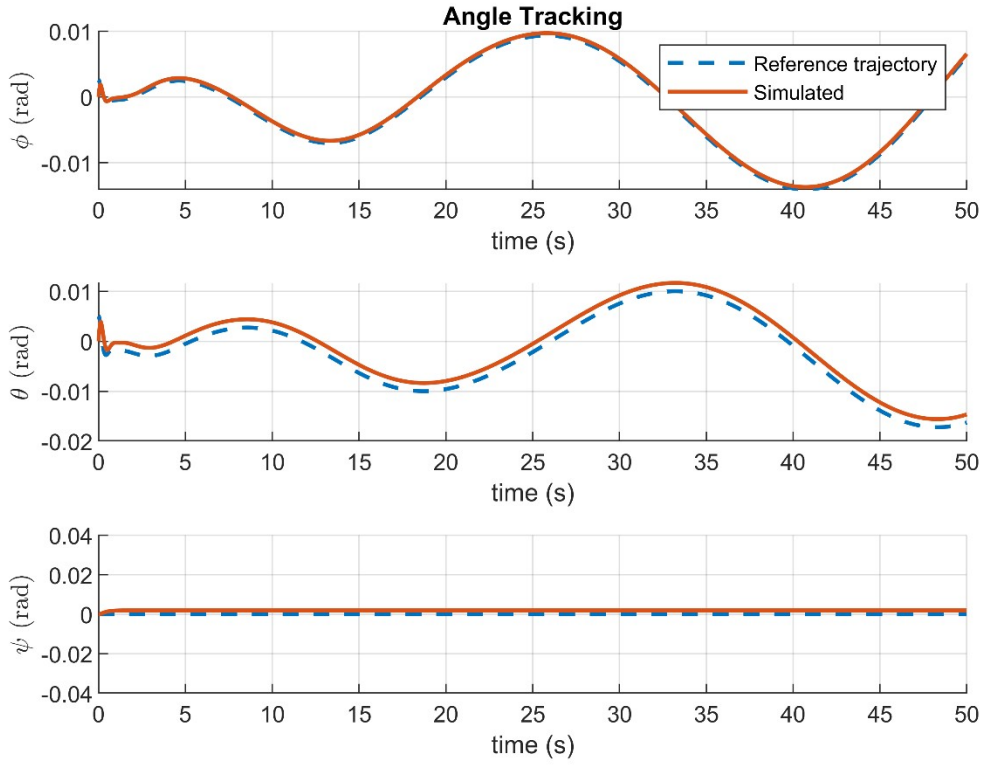

Figure 8: SNAC tracking angles generated by the approximator NN.

Figure 8 displays the angular velocity tracking, where the reference signal for velocity was generated by taking the numeric derivative of the angular reference signal generated by the system solver. The small magnitude of the angular velocities is also due to the low angular change demanded by the selected trajectory. Notably, the initial second of the angular velocity tracking shows the section responsible for the stabilization of the quadcopter during initial takeoff. A close look at these sections of the angular velocities can be seen in Fig. 11.

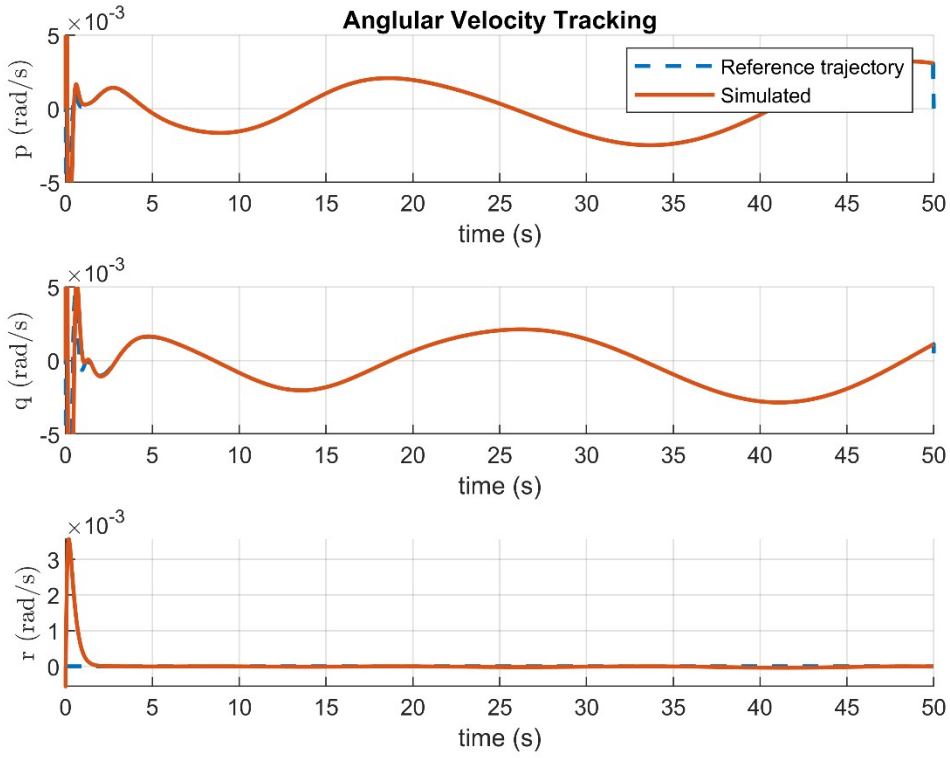

Figure 9: SNAC tracking angles generated by the approximator NN.

4.3 Quadcopter Control Inputs

Figure 10 shows the quadcopter inputs of the trust and torques in the principle axis. The thrust stabilizes slightly above 9.81 N, which is the force required to keep the 1 kg simulated quadcopter climbing at the desired rate. The input torques have a small magnitude corresponding to the small angular rates and angles demanded by the trajectory. The initial second of the torque controls shows the stabilization of the angular velocities. A more detailed look can be seen in Fig. 11.

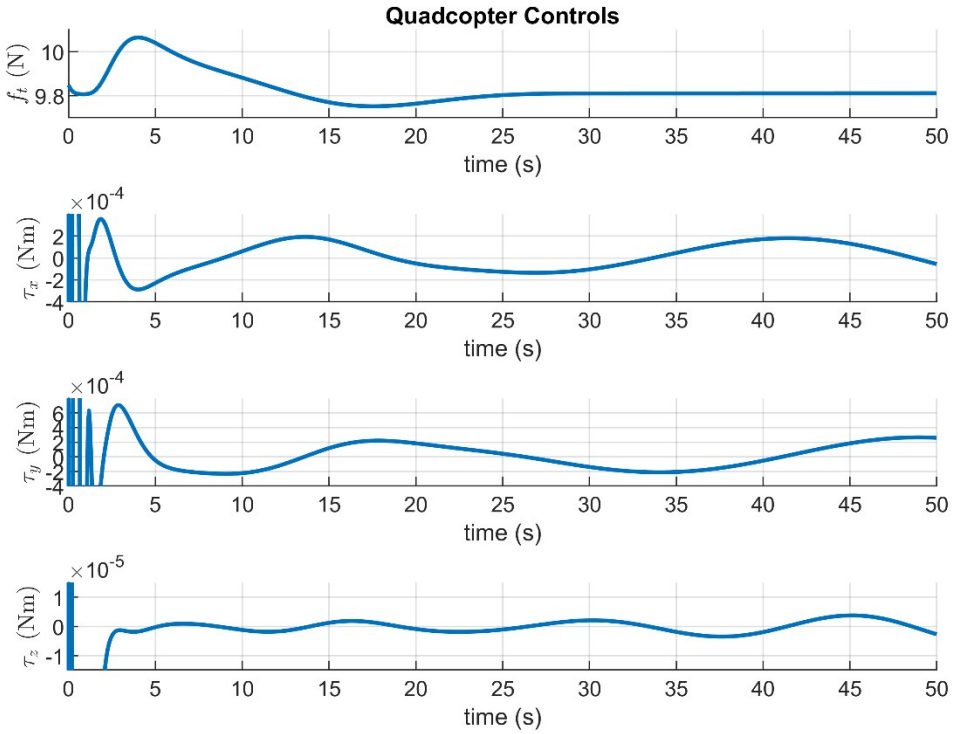

Figure 10: Quadcopter controls of thrust and torques in the principal axis.

4.4 Stabilization

Quadcopter performance is directly linked to its ability to maintain stability, which is crucial for the drone's maneuverability and robustness. A stable quadcopter is more flyable and resilient to external disturbances like wind or gusts, allowing it to maintain its position and orientation accurately. As the SNAC algorithm is a closed-loop feedback controller, it can handle noise and variations in the quadcopter dynamics. Figure 11 shows the angular velocities and corresponding stabilizing torque controls.

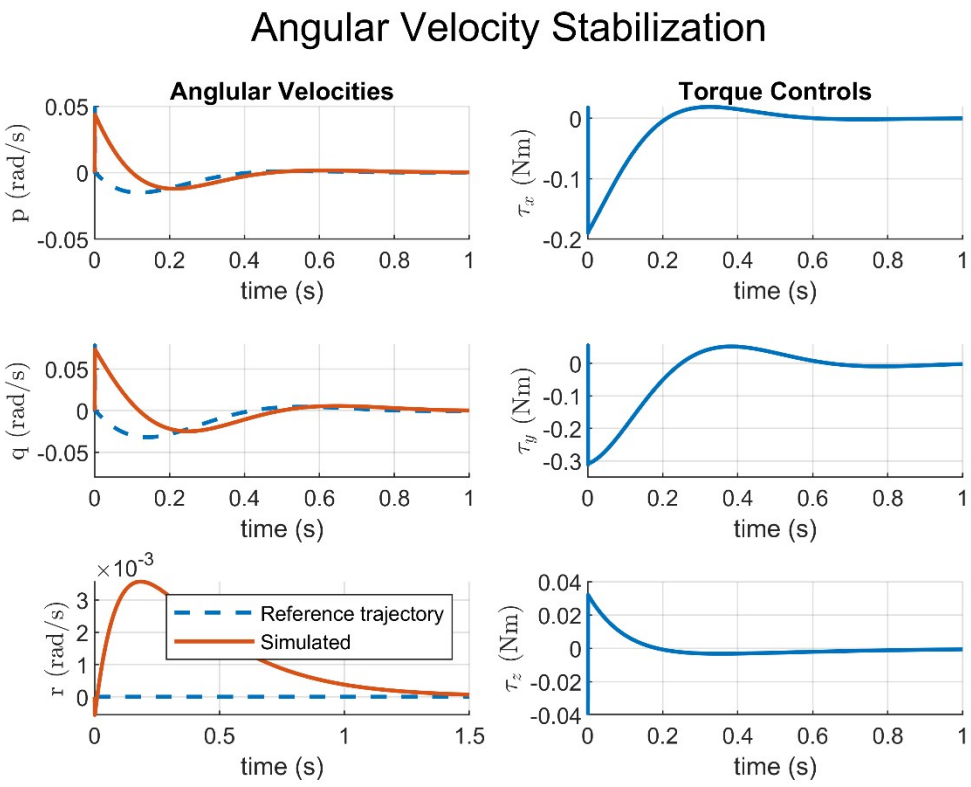

Figure 11: Quadcopter controls of thrust and torques in the principal axis.

4.5 SNAC and DIDO Comparison

DIDO is a commercial MATLAB optimal control software that finds the optimal control using Pontryagin minimum principle (Ross, 2012, 2015). DIDO can achieve optimal tracking control given the full quadcopter dynamics with no subsystem divisions. However, the result is an open-loop control that tracks a fixed trajectory, with fixed initial conditions, and within a selection time-horizon, requiring retraining for each scenario. DIDO's solutions are known as extremal solutions, representing high-quality solutions that may not necessarily be optimal. However, an optimal solution must also be an extremal solution. The results DIDO produces can be used to validate the effectiveness of the SNAC quadcopter optimal controllers.

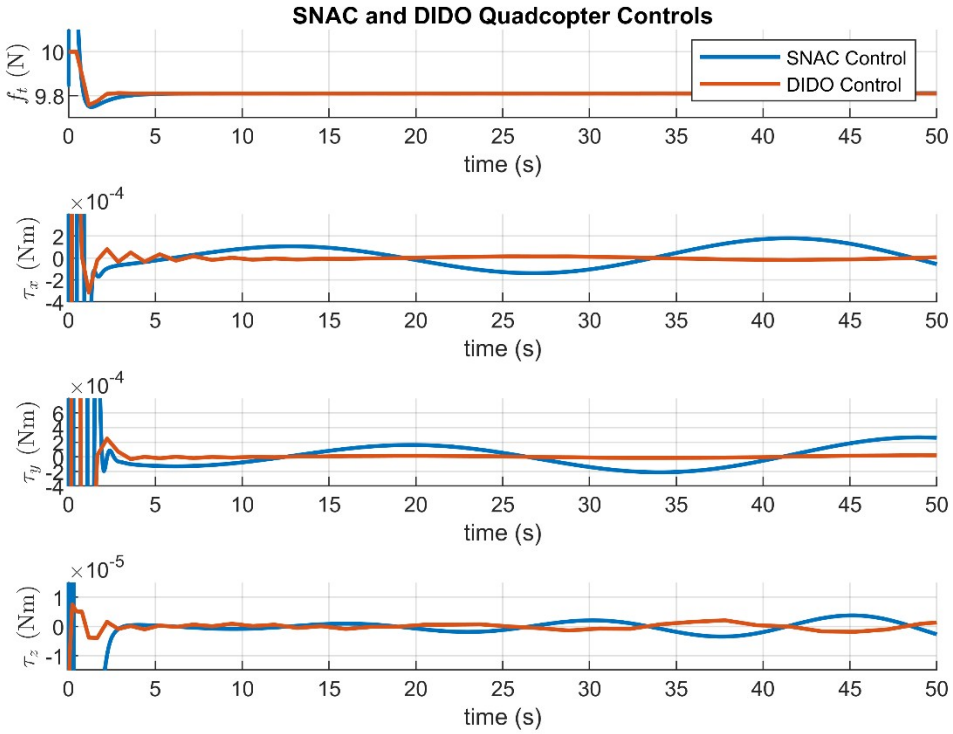

Figure 12: DIDO and SNAC quadcopter controls of thrust and torques in the principal axis.

Figure 12 shows the DIDO controls overlaid with the SNAC controls. As DIDO finds an extremal solution that can be optimal, the results of the SNAC controllers can be compared to determine its efficiency. Notably, the DIDO controller was trained with parameters similar to the parameters seen in Table 1 and Table 2 used to train the SNAC controls. The same dynamics were used to train both systems, with the dynamic model provided to DIDO being the full, undivided dynamics of the quadcopter. The result in Figure 12 serves to validate the magnitude and trend of the SNAC controls with those of the DIDO control. Variations in the controls exist due to differences in how DIDO is trained and discretized with respect to the SNAC controllers. For example, DIDO discretizes dynamics with various time steps resulting in dense, rapid

sampling during the first few iterations of the simulation. The time steps then increase during the middle of the training time.

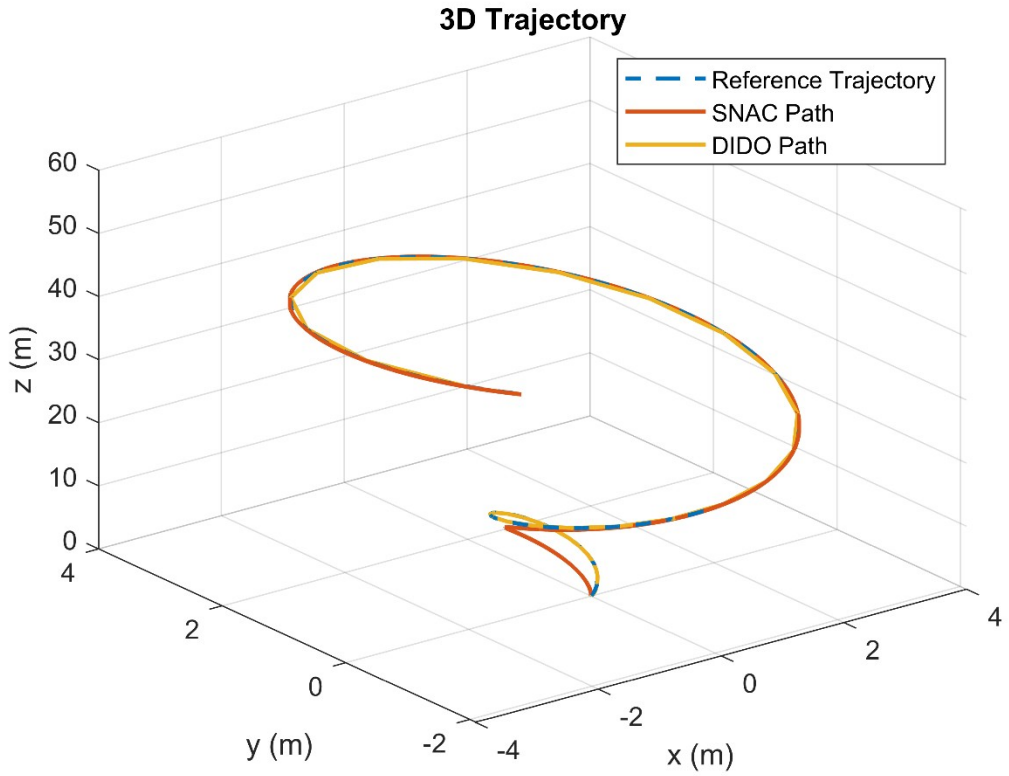

Figure 13: DIDO and SNAC 3D quadcopter trajectory.

Figure 13 shows a comparison of the 3D trajectory of the SNAC and DIDO controllers. Both converge to the desired reference signal with the DIDO path converging almost immediately. Notably, the SNAC path converges soon after. Once both converge, the tracking is nearly identical, with variations in the DIDO path present due to the variable time steps.

4.6 Robustness

The quadcopter control was additionally tested with varying levels of noise applied to the quadcopter thrust and torque inputs. Figure 14 shows up to 600% noise was added to the inputs with the quadcopter controller being able to control the system and track a trajectory. The quadcopter was able to continue tracking the helical trajectory with various initial conditions as seen in Figure 15 and at higher levels of noise. This type of simulated dynamic disturbance is limited in its realism, and does not accurately represent disturbances such as wind, gusts, or uncertainties in the dynamic model. In the case of wind, additional forces would be applied in the three position directions with the force of the wind corresponding to the cross-sectional area of the quadcopter, the quadcopter's coefficient of drag, and the wind's speed and heading.

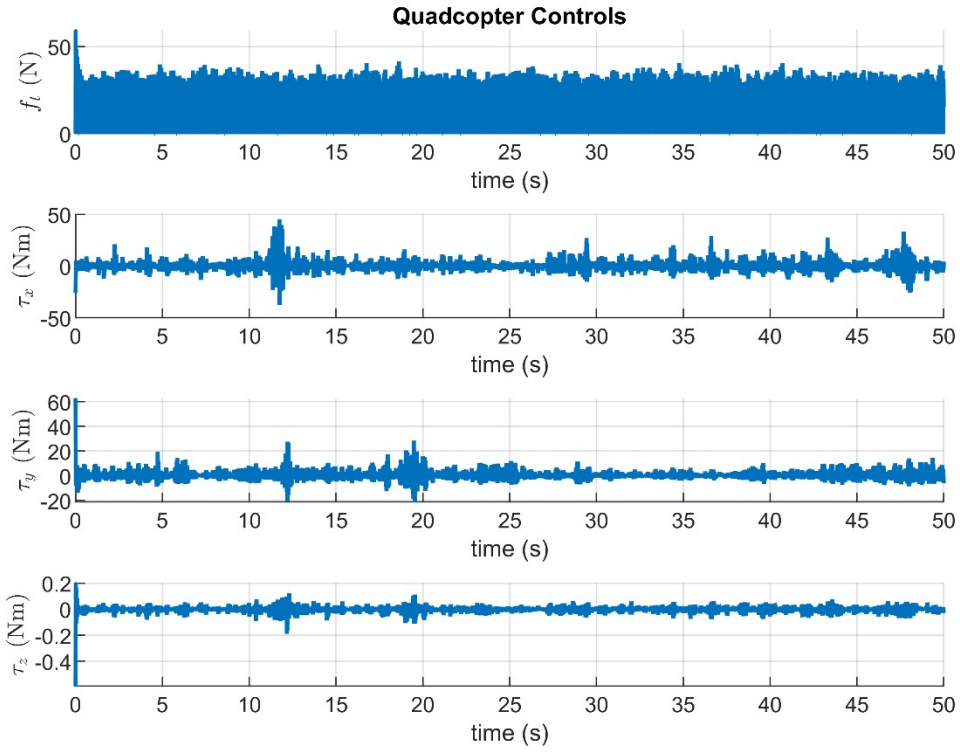

Figure 14: Noisy quadcopter controls of thrust and torques in the principal axis.

Figure 14 depicts the effects of noise on the control signal when the system has zero initial conditions. The noise was introduced by multiplying the current control by up to 600%. Since the quadcopter's rotors can only produce positive thrust, the maximum possible thrust is limited to zero. While the magnitude of torques is generally small, the accumulation of noise over time requires increasingly significant spikes of torques to stabilize the system.

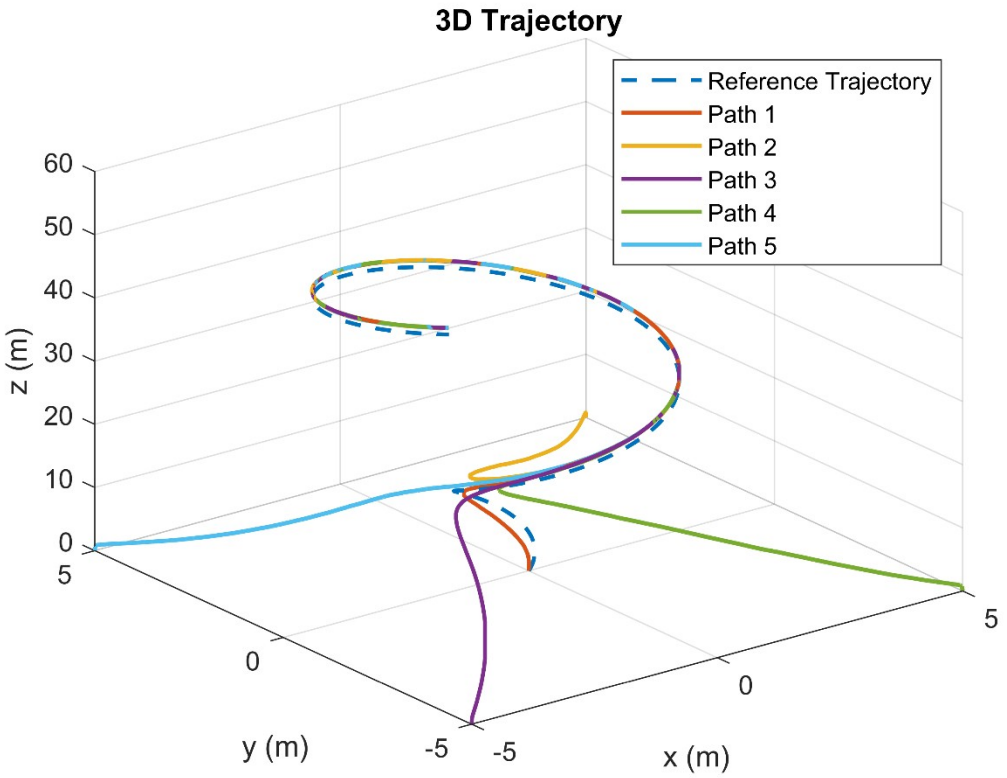

Figure 15: 3D quadcopter trajectory with noisy inputs.

Figure 15 demonstrates the robustness of the developed SNAC controllers by showing paths with different initial conditions tracking the helical trajectory defined earlier. To test the controllers for significant noise, each path has up to 600% noise added to its controls. Despite the

noise, the controllers maintain their ability to control systems with various initial conditions and effectively track the reference trajectory. The altitude tracking shows a small, constant offset, that can be alleviated by retraining the controller with greater emphasis on the altitude parameter z. Overall, the controllers perform well and prove to be robust against noise.

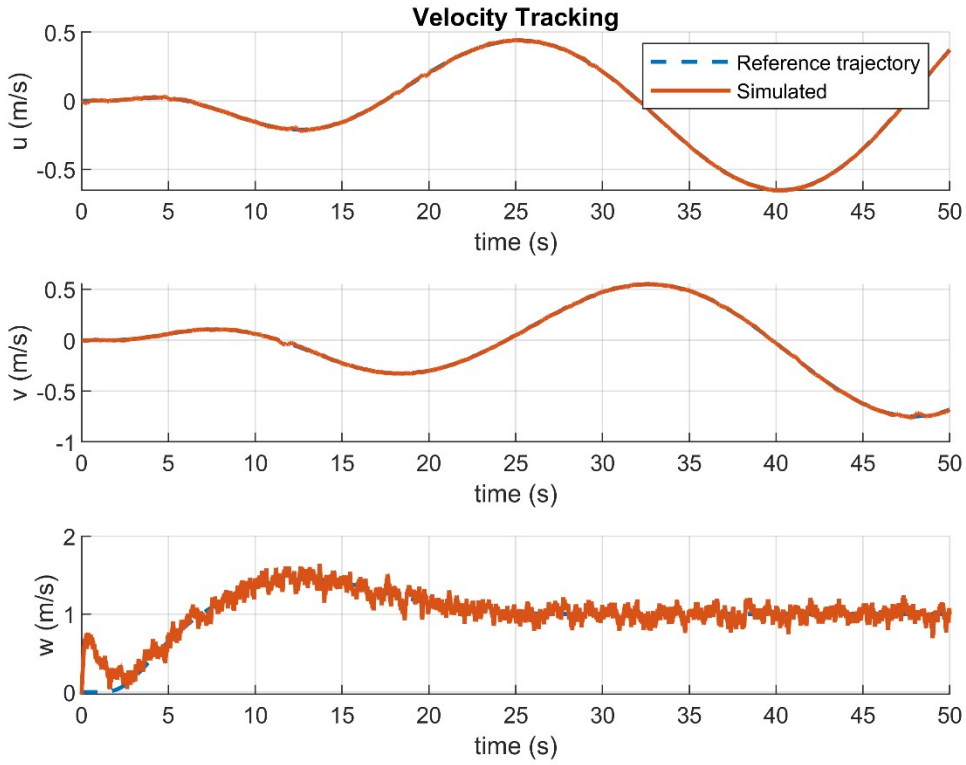

Figure 16: Velocity tracking for quadcopter with zero initial conditions and noise.

Figure 16 displays the velocity tracking of the quadcopter for the zero initial condition path. The SNAC controllers were designed to prioritize tracking in the x and y directions, as highlighted by the state penalizing matrix in Table 1, resulting in accurate and smooth tracking of the corresponding velocities. However, the training emphasis on x and y tracking leads to increased noise in the velocity of the quadcopter in the z direction, w. Despite the noise, the

SNAC controllers are able to effectively track the velocity trajectories that correspond to the desired position.

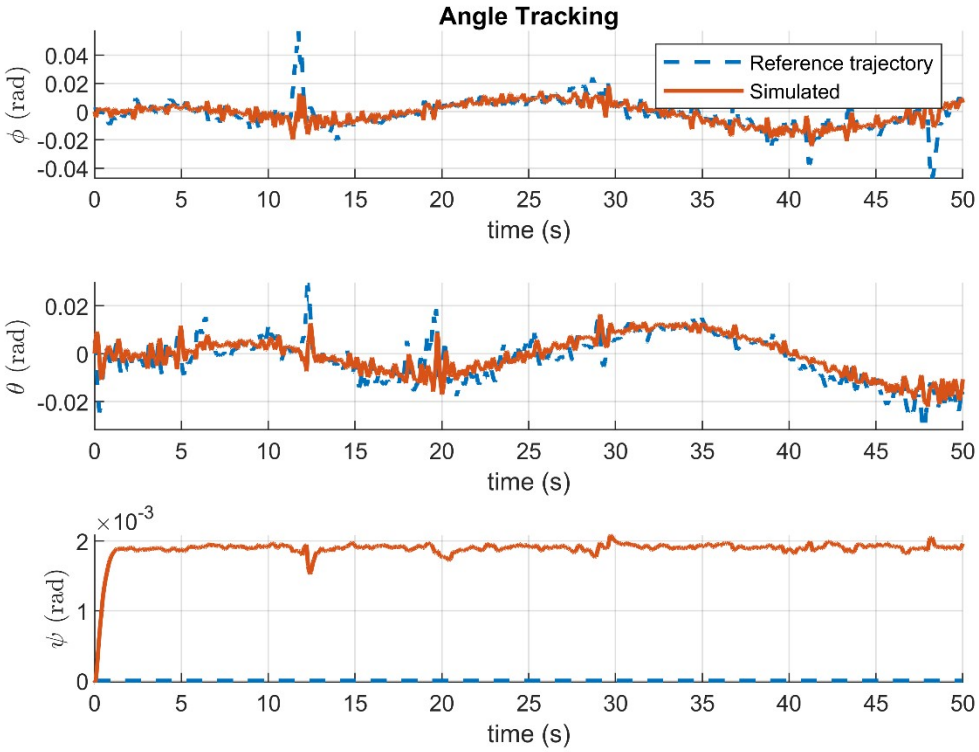

Figure 17: Angle tracking for quadcopter with zero initial conditions and noise.

Figure 17 shows the angle tracking of the quadcopter for the zero initial condition path. For the ϕ and the θ, notable spikes appear near 12 seconds. This occurred as the quadcopter needed to adjust angles due to the unpredictable noise inputs rapidly. For the yaw, ψ, a constant offset of 0.002 rad is present. This offset is of a sufficiently low magnitude that the attitude regulator prioritizes tracking for the pitch and roll. The yaw, ψ, reference is an input variable selected as zero, so there is no noise in the reference signal. However, the pitch and roll references are both outputs based on the control received from the position regulator.

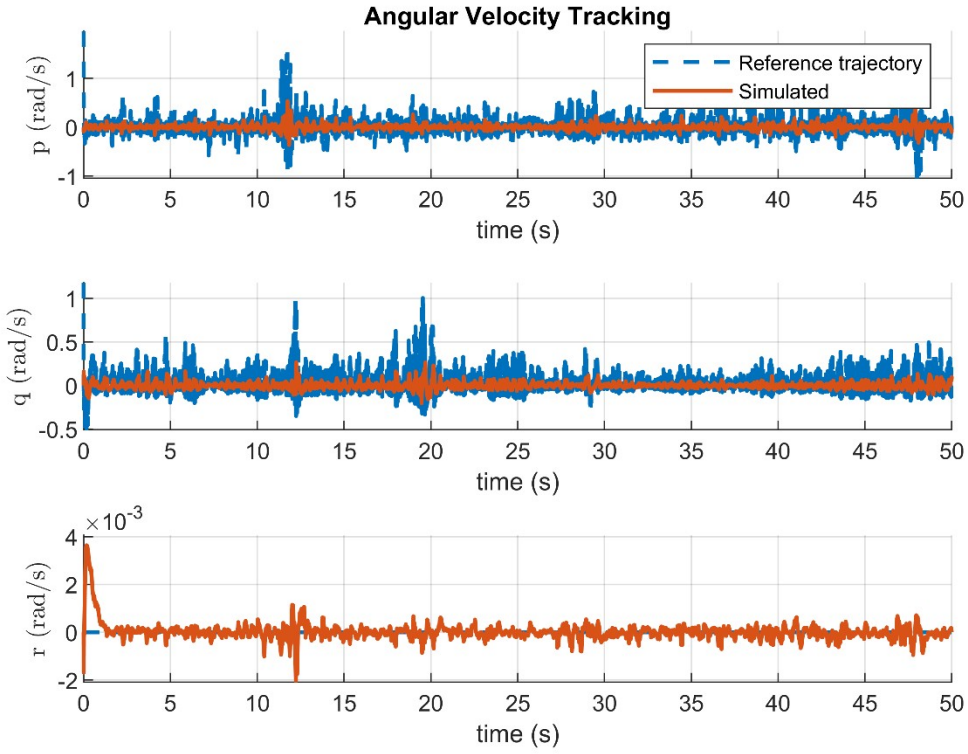

Figure 18: Angular velocity tracking for quadcopter with zero initial conditions and noise.

Figure 18 shows the noisy angular velocity tracking for the path with zero initial conditions. Small angular velocities are necessary due to the small angle requirements of the large-diameter helical trajectory.

The SNAC controller's ability to maintain the reference position with large percentages of noise in the controls demonstrates its robustness and versatility in dealing with varying control and dynamic inputs. To model the effect of environmental conditions more accurately, additional forces should be added to the three position directions corresponding to the wind force. Overall, the results demonstrate the effectiveness of the SNAC controllers in tracking complex reference trajectories with varying initial conditions and significant noise.

CHAPTER V

CONCLUSION

In this paper, the use of Single Network Adaptive Critics (SNAC) control for regulating the position and attitude of a quadcopter is presented. The SNAC algorithm was trained offline and applied to both subsystems, achieving state regulation for an infinite-horizon. The resulting controllers were compared with DIDO to validate the solution, demonstrating the effectiveness of SNAC in controlling quadcopter dynamics. The robustness of the closed-loop SNAC controller was also tested by adding substantial noise, showing its ability to maintain control in the presence of significant disturbances. The successful application of the SNAC controller highlights the potential of reinforcement learning techniques for controlling complex systems in real-world scenarios. However, future work in quadcopter control using SNAC should consider more sophisticated environments that account for sensor sampling frequency, control output to rotor speeds, and more realistic external disturbances. Overall, this research demonstrates the benefits of using SNAC for nonlinear control, showing its ability to achieve near-optimal tracking control while reducing computational complexity. With the increasing demand for autonomous vehicles and drones, this work contributes to the development of more efficient and effective control strategies for these complex systems.

REFERENCES

Ali, S. F., & Padhi, R. (2011). Optimal blood glucose regulation of diabetic patients using single network adaptive critics. Optimal Control Applications and Methods, 32(2), 196-214.

Argentim, L. M., Rezende, W. C., Santos, P. E., & Aguiar, R. A. (2013, May). PID, LQR and LQR-PID on a quadcopter platform. In 2013 International Conference on Informatics, Electronics and Vision (ICIEV) (pp. 1-6). IEEE.

Bellman, R., & Kalaba, R. E. (1965). Dynamic programming and modern control theory (Vol. 81). New York: Academic Press.

Bouabdallah, S., Noth, A., & Siegwart, R. (2004, September). PID vs LQ control techniques applied to an indoor micro quadrotor. In 2004 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)(IEEE Cat. No. 04CH37566) (Vol. 3, pp. 2451-2456). IEEE.

Bouabdallah, S., & Siegwart, R. (2005, April). Backstepping and sliding-mode techniques applied to an indoor micro quadrotor. In Proceedings of the 2005 IEEE international conference on robotics and automation (pp. 2247-2252). IEEE.

Bresciani, T. (2008). Modeling, identification, and control of a quadrotor helicopter. MSc theses.

Bu, X., & Qi, Q. (2020). Fuzzy optimal tracking control of hypersonic flight vehicles via single-network adaptive critic design. IEEE Transactions on Fuzzy Systems, 30(1), 270-278.

Das, A., Subbarao, K., & Lewis, F. (2009). Dynamic inversion with zero-dynamics stabilisation for quadrotor control. IET control theory & applications, 3(3), 303-314.

Fu, Z. J., Li, B., Ning, X. B., & Xie, W. D. (2017). Online adaptive optimal control of vehicle active suspension systems using single-network approximate dynamic programming. Mathematical Problems in Engineering, 2017.

Heydari, A., & Balakrishnan, S. N. (2014). Fixed-final-time optimal tracking control of input-affine nonlinear systems. Neurocomputing, 129, 528-539.

Hwangbo, J., Sa, I., Siegwart, R., & Hutter, M. (2017). Control of a quadrotor with reinforcement learning. IEEE Robotics and Automation Letters, 2(4), 2096-2103.

Kirk, D. E. (2004). Optimal control theory: an introduction. Courier Corporation.

Koch, W., Mancuso, R., West, R., & Bestavros, A. (2019). Reinforcement learning for UAV attitude control. ACM Transactions on Cyber-Physical Systems, 3(2), 1-21.

Konda, V., & Tsitsiklis, J. (1999). Actor-critic algorithms. Advances in neural information processing systems, 12.

Lee, D., Jin Kim, H., & Sastry, S. (2009). Feedback linearization vs. adaptive sliding mode control for a quadrotor helicopter. International Journal of control, Automation and systems, 7, 419-428.

Lewis, F. L., & Vrabie, D. (2009). Reinforcement learning and adaptive dynamic programming for feedback control. IEEE circuits and systems magazine, 9(3), 32-50.

Madani, T., & Benallegue, A. (2006, October). Backstepping control for a quadrotor helicopter. In 2006 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 3255-3260). IEEE.

Nobleheart, W., Shivanapura Lakshmikanth, G., Chakravarthy, A., & Steck, J. E. (2013). Single network adaptive critic (SNAC) architecture for optimal tracking control of a morphing aircraft during a pull-up maneuver. In AIAA Guidance, Navigation, and Control (GNC) Conference (p. 5003).

Padhi, R., & Balakrishnan, S. N. (2006). Optimal management of beaver population using a reduced-order distributed parameter model and single network adaptive critics. IEEE transactions on control systems technology, 14(4), 628-640.

Padhi, R., Unnikrishnan, N., Wang, X., & Balakrishnan, S. N. (2006). A single network adaptive critic (SNAC) architecture for optimal control synthesis for a class of nonlinear systems. Neural Networks, 19(10), 1648-1660.

Powell, W. B. (2007). Approximate Dynamic Programming: Solving the curses of dimensionality (Vol. 703). John Wiley & Sons.

Ross, I. M., A Primer on Pontryagin's Principle in Optimal Control, Second Edition, Collegiate Publishers, San Francisco, 2015.

Ross, I. M. and Karpenko, M., "A Review of Pseudospectral Optimal Control: From Theory to Flight," Annual Reviews in Control, Vol.36, 2012, pp.182-197.

Sabatino, F. (2015). Quadrotor control: modeling, nonlinear control design, and simulation.

Sutton, R. S., & Barto, A. G. (2018). Reinforcement learning: An introduction. MIT press.

Waslander, S. L., Hoffmann, G. M., Jang, J. S., & Tomlin, C. J. (2005, August). Multi-agent quadrotor testbed control design: Integral sliding mode vs. reinforcement learning. In 2005 IEEE/RSJ International Conference on Intelligent Robots and Systems (pp. 3712-3717). IEEE.

Xu, R., & Ozguner, U. (2006, December). Sliding mode control of a quadrotor helicopter. In Proceedings of the 45th IEEE Conference on Decision and Control (pp. 4957-4962). IEEE.

APPENDIX

APPENDIX

Code for the SNAC training loop.

```
% Nonvectorzied SNAC training loop
for i = 1:max_training_loop
    basis_func = zeros(N_neurons, N_patterns);
    lambda_k_plus_1_target = zeros(N_states, N_patterns);
    % Generating target costate for all number of patterns
    for j = 1 : N_patterns
        X1 = X_min + (X_max - X_min) * rand;%(1, N_patterns);
        X2 = Y_min + (Y_max - Y_min) * rand;%(1, N_patterns);
        X3 = Z_min + (Z_max - Z_min) * rand;%(1, N_patterns);
        X4 = u_min + (u_max - u_min) * rand;%(1, N_patterns);
        X5 = v_min + (v_max - v_min) * rand;%(1, N_patterns);
        X6 = w_min + (w_max - w_min) * rand;%(1, N_patterns);
        % Random states within defined domain of trainig
        x_k = [X1; X2; X3; X4; X5; X6];
        % Running states through nerual network
        basis_func(:,j) = Basis_Func_84(x_k);
        lambda_k_plus_1 = W' * Basis_Func_84(x_k);
        % Optimal control equation
        u_k = - R^-1 * G(x_k).' * lambda_k_plus_1;
        % Discretized dynamics
        x_k_plus_1 = F(x_k) + G(x_k) * u_k;
        % States through nerual network
        lambda_k_plus_2 = W' * Basis_Func_84(x_k_plus_1);
        % Target costate equation
        A_k_plus_1 = A(x_k_plus_1);
        lambda_k_plus_1_target(:,j) = Q * (x_k_plus_1) ...
            + A_k_plus_1.' * lambda_k_plus_2;
    end
    % Least squares to update network weights
    W = (basis_func * basis_func')\(basis_func * lambda_k_plus_1_target');
    if isnan(Position_W)
        fprintf('Divergence in trainig \n')
        break
    end
    % Check for convergence
    error(:, :) = W' * basis_func - lambda_k_plus_1_target;
    if mae(error(:,:))< threshold
        break
    end
end
```

Published with MATLAB® R2022a

BIOGRAPHICAL SKETCH

Alberto Velazquez-Estrada was born in McAllen, Texas, on September 6, 1999. He attended Vanguard Academy Rembrandt and graduated in 2018. He pursued his educational career as an undergraduate student at the University of Texas Rio Grande Valley, where he graduated Summa Cum Laude with a bachelor's degree in Mechanical Engineering in May 2021. Alberto continued his education at the University of Texas Rio Grande Valley and was awarded the Presidential Research Fellowship in the Spring of 2022. Alberto served as the Co-Captain and Design Lead for the RGV Baja Racing at UTRGV team from 2020 to 2023, designing, manufacturing, and testing an off-road 4x4 mini-Baja vehicle. Alberto completed his Master of Science in Engineering degree in Mechanical Engineering in May 2023. Alberto Velazquez-Estrada can be reached at albertovelazquez387@gmail.com.