Multicopter PID Attitude Controller Gain Auto-tuning through Reinforcement Learning Neural Networks

Daewon Park
Faculty of Mechanical and Aerospace Engineering Sejong University South Korea, Seoul, 05006, Korea
dwon0204@gmail.com

Hyona Yu
Faculty of Intelligence Mechatronics Engineering Sejong University South Korea, Seoul, 05006, Korea
gysk0507@gmail.com

Nguyen Xuan-Mung
Faculty of Mechanical and Aerospace Engineering Sejong University South Korea, Seoul, 05006, Korea
xuanmung1009@gmail.com

Junyong Lee
Faculty of Mechanical and Aerospace Engineering Sejong University South Korea, Seoul, 05006, Korea
wlsgudrh04@gmail.com

Sung Kyung Hong
Faculty of Mechanical and Aerospace Engineering Sejong University
skhong@sejong.ac.kr

ABSTRACT

Multicopters continue to gain their applications in the real world and their control problem has attracted a great number of studies. Among several control techniques, the proportional-integral-derivative control method appears to play important roles in seeking a simple and efficient controller for a complex system like a multicopter. However, the gain tuning process of a proportional-integral-derivative controller is still a time-consuming task and, therefore, finding an automatic gain tuning method which saves time and ensures satisfactory control performance has become one of the most important efforts that researchers all around the world are nowadays undertaking. In this paper, we present a proportional-integral-derivative controller gain auto-tuning method using the reinforcement learning neural networks. Software and hardware-in-the-loop simulations were carried out to demonstrate the effectiveness of the proposed method.

CCS Concepts

Computing methodologies→Reinforcement learning

Keywords

PID control; attitude control; PX4; ROS; reinforcement learning; neural networks; auto-tuning

1. INTRODUCTION

In the Fourth Industrial Revolution, the importance of robotics and unmanned transportation has increased dramatically. Although many modern control methods have been studied and developed in the area of control engineering, PID control has become the most widely used technique in a variety of applications because it is simple and easy to design and typically delivers a satisfactory performance [1]. The PID approach was used in many studies [1]–[6] to achieve not only quadcopter attitude stabilization control, but also altitude and horizontal position control. The PID control can be divided into two types, i.e., conventional PID and a multi-loop PID control. In many cases, e.g., attitude tracking control, trajectory tracking control, the conventional PID is not able to exhibit good control performance. Therefore, in recent studies [7]–[8], the authors proposed the use of a multi-loop control architecture (i.e., inner-loop and outer-loop) to control quadcopters in specific applications. The outer-loop controllers were designed in different ways while the inner-loop controllers were all implemented using the PID control law.

However, it is difficult to expect accurate values because the actual model is mostly a nonlinear model and the disturbance such as wind is added in the environment. Simulations using a linear model also have different transfer functions for each vehicle, so that the gain values are always different, which makes the calculation complex.

In quadcopters, the gains are different for each vehicle because a number of computations are needed. Several external disturbances influence the system so that this leads to an imperative requirement of an auto-tuning method. In this paper, we present a PID gain auto-tuning method in PX4, which is an open-source autopilot system using reinforcement learning neural networks (RLNN) with Robot Operation System (ROS). We call this new method as Multicopter Gains Auto-Tuning (MGAT).

To verify the proposed algorithm, several numerical and hardware-in-the-loop simulations are carried out. A DJI-F450 multicopter is used as the experimental platform to conduct on-testbed attitude flight for the evaluation of the effectiveness of our method.

ICCRT 2019, December 12–14, 2019, Jeju, Republic of Korea ICCR T 2019, December 12–14, 2019, Jeju, Republic of Korea © 2019 A ssociation for Computing Machinery.
ACM ISBN 978-1-4503-7252-7/19/12…$15.00
https://doi.org/10.1145/3387304.3387327

2. PRELIMINARIES AND PROBLEM F ORMULATION

2. 1 Quadcopter Dynamics Model

Here, we briefly summarize the quadcopter attitude dynamics model as it has already been presented previously [1], [7]-[9].

Let ϕ, θ, and ψ denote the three Euler angles roll, pitch, and yaw, respectively, where, $|\phi| < \pi / 2$, $|\theta| < \pi / 2$, and $|\psi| \le \pi$ (see Fig. 1). J_x, J_y, and J_z the moments of inertia along the x, y, and z axes, respectively; l the arm length of the vehicle. The quadcopter dynamics model can be described as follows:

$$\begin{cases} \ddot{\phi} = \left(\dfrac{J_y - J_z}{J_x}\right)\dot{\theta}\dot{\psi} + \dfrac{l}{J_x}U_2 \\ \ddot{\theta} = \left(\dfrac{J_z - J_x}{J_y}\right)\dot{\phi}\dot{\psi} + \dfrac{l}{J_y}U_3 \\ \ddot{\psi} = \left(\dfrac{J_x - J_y}{J_z}\right)\dot{\phi}\dot{\theta} + \dfrac{1}{J_z}U_4 \end{cases} \tag{1}$$

where $U_i (i = 1,2,3,4)$ denotes the control inputs which are:

$$\begin{cases} U_1 = F_1 + F_3 + F_2 + F_4 \\ U_2 = F_4 - F_2 \\ U_3 = F_3 - F_1 \\ U_4 = C_d\left(F_1 + F_3 - F_2 - F_4\right) \end{cases} \tag{2}$$

where $F_i = C_t\Omega_i^2$ denotes the thrust force generated by the motor i; Ω_i is the speed of the motor i; C_t and C_d are the thrust and drag coefficients, respectively.

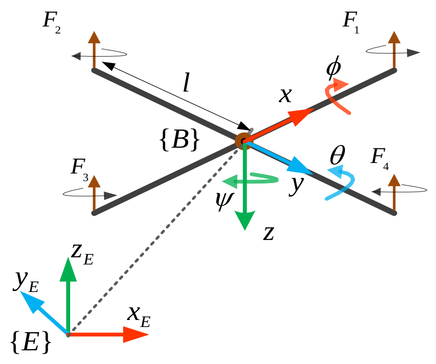

Figure 1. Quadcopter configuration.

2. 2 Quadcopter Multi-Loop Attitude Controller

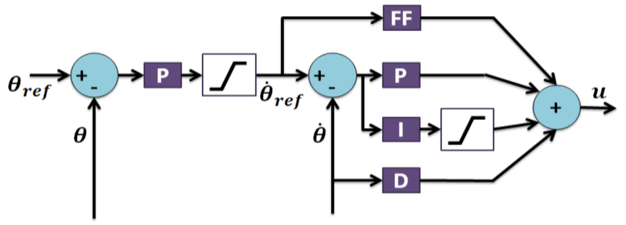

Figure 2. Quadcopter multi-loop P-PID attitude controller

The multi-loop PID attitude controller of the quadcopter consists of two loops (Fig. 2), i.e., outer loop and inner loop. The outer loop is a proportional (P) controller with a controller gain k while the inner loop is a PID with controller gains k_p, k_i, k_d. Since the controllers designed for the quadcopter's roll, pitch, and yaw attitude are of the same form we will only present the pitch controller to avoid repeatability.
The outer loop generates control action as follows.

$$u_{outer} = ke \tag{3}$$

with e being the attitude tracking error, i.e., $e = \theta_{ref} - \theta$ (Table I). Thus, the inner loop outputs control action as:

$$u = k_p e_1 + k_i \int e_1 + k_d \dot{e}_1 \tag{4}$$

where $e_1 = u_{outer} - \dot{\theta}$.

Table 1 Explanation of the Symbols

Symbol	Unit	Description
θ	rad	Roll angle
$\dot{\theta}$	rad/s	Angular velocity of roll angle
θ_{ref}	rad	Reference angle for roll angle
$\dot{\theta}_{ref}$	rad/s	Reference angular velocity of roll angle
e		Input minus output of angle or angular velocity
k		Controller gain

2. 3 Problem Statement

The duty of the gain auto-tuning is finding, automatically, an appropriate set of controller gains k, k_p, k_i, and k_d that can be applied to the controller to achieve superior control performance.

2. 4 Robot Operating System

The well-known open-source robotics middleware Robot Operating System (ROS) is used as a meta-operating system for the vehicle [10]. ROS provides tools and libraries for getting, building, writing, and running code that works on multiple computer systems. The main goal of ROS is to make it easier to reuse code in robot research and development. Regardless of which system is used, ROS can be used together.

2. 5 Reinforcement Learning and Neural Networks

The main goal is to maximize all rewards that will get during the training at the given time. Reinforcement Learning is part of Machine Learning composed of agents and environment [11]-[13]. In the environment, we use the Markov decision process (MDP) theory MDP is consisted of $(S, A, \{P(s,\ a)\}, R,\ \gamma)$, where, S is a set of states, A a set of actions, $\{P(s,a)\}$ a state of a transition probability, R a reward function, and γ a discount factor of rewards. The following formula is about Action Value function (Q function) based on the Bellman Equation, where $Q(s,a)$ is Q function, R a reward, s a state, and a an action. When the learning starts at s and a, we can predict a reward it will get if it is defined by both s and a.

$$Q(s,a) = R(s,a) + \gamma \sum s' \in SP(s,a)(s') max \in Q(s', a') \tag{5}$$

It means each reward (state, action) multiplied by γ is saved. To make Reinforcement Learning and Neural Networks (RLNN) and so on, we use a programming language python and Keras [14] with Tensorflow backend. The optimizers used in MGAT are the

Rectified Linear Unit (ReLu) and Softmax activation functions.

3. PROBLEM FORMULATION

3.1 RLNN State Generation Process

This study applies step attitude setpoints to the vehicle and observe the system response. The attitude step input is set of ± 5 degrees (and can be adjusted by users) and the output is the attitude measurement provided by an inertial measurement unit (IMU).In the main code, real-time IMU data is stored as an array and is used as an input value of MGAT's RLNN. Besides, the IMU data array is used to obtain a reinforced learning reward. In the reward function, the reward is returned according to the predetermined criteria. The algorithm learns the behavior (gain tuning) to maximize the value of the reward depending on the current state (fast responsiveness, accuracy, etc.).

3.2 Neural Network Structure of MGAT A lgorithm

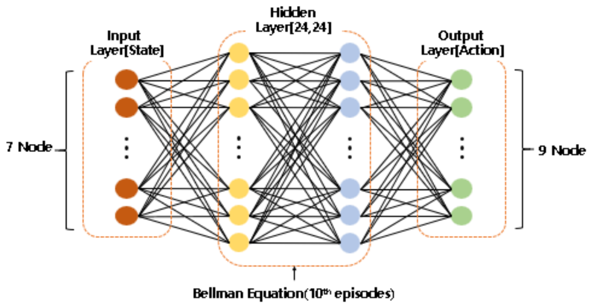

Figure 3. MGAT Neural Network Structure

MGAT's RLNN consists of four layers consist of two hidden layers, an input layer and an output layer with a learning rate of 0.01 (Fig. 3). The input layer and the output layer have 7 nodes and 9 nodes each, respectively, while the hidden layer consists of 24 nodes and uses Relu and Softmax as the activation functions. This RLNN uses the Stochastic Gradient Descent (SGD) algorithm to regress cost in reinforcement learning. The loss function is a negative log-likelihood function. In this algorithm, we set the discount factor of 0.99. The main learning theory we use is MDP. To find out which action is the best choice, the Action Value Function will be used.
MGAT uses the state array as an input data, switching attitude PPID controller gain values as an action, and return rewards. The software using reinforcement algorithm with rospy [16], which is a python's library for using ROS, will get parameters of a multicopter with mavros [15], and that will become an input data in this main code. With parameters in each episode, MGAT returns the state array. MGAT finds out the best action to do in each episode with this input data array. The first policy for the action and the state in MGAT is a random probability so that it can predict the better action with a trial and error. The action array is made of increasing or decreasing of the controller gains k, k_p, k_i, k_d of roll angle and roll angular velocity. The quadcopter will act the last chosen action and return rewards of the latest episode. The reward function is based on the requirement of controller criteria. The standard of the rewards is how many state-inputs can be satisfied with the controller criteria that our requirement. This part can tune to satisfy the user's requirement with the compensation for the state based on the desired state (such as fast settling time or accurate response). In this paper, we focused on reaching our target quickly and accurately.

3.3 Procedure of MGAT

Before the training with MGAT, we set the communication parameter SYS_COMPANION in PX4. And then we wire the UART port. After launching the roscore, central node for connection and message communication between nodes, at last, and connect TX2 [17], which is an embedded AI computer, with Pixhawk (PX4) using mavros.
When MGAT starts, it will repeat the step input ± 5 degrees of a quadcopter so that it moves at every episode automatically. This will continue during 1000 episodes. It also chooses the action, a bunch of PPID gains in roll attitude controller, according to the probability of selecting action. In each episode, a set of action, a state, a reward is saved for an RLNN. Using these sets, MGAT learns how to get a better reward than the latest learning and updates a neural network in every 10th episode.

4. SOFWARE-IN-THE-LOOP SIMULATIO N RESULTS

Before doing the simulation, it is needed to apply the vehicle we use in it. Through the motor thrust test and the multicopters' specifications, we implemented the following.
We performed it in GAZEBO World to adapt MGAT, and the following is the result. This experiment only confirmed the tendency without prior learning (Fig. 4-6).
Through this, we were able to find out the possibility of the auto-tuning algorithm and its applicability to the actual vehicle. The controller has shown that the reactivity about the angle has reached the ideal target. If the experiment goes further, we will get more ideal gains value. We did not put the thrust control into the internal MGAT algorithm, and it would be necessary to have a very large space for the actual flight in case it would fall. The result of the operation of this algorithm in TX2 will be shown in the following results.

Figure 4. F450 Multicopter Model in Gazebo

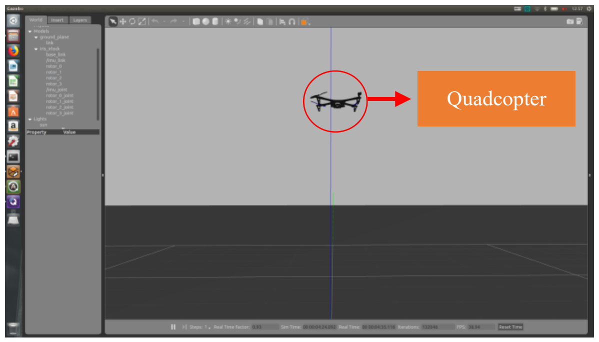

Figure 5. Simulation Using Gazebo

5. HARDWARE-IN-THE-LOOP SIMUAL TION RESULTS

This study proceeds the examination of MGAT in px4, using mavros rospy, and python. The experiment was focused on tuning to adjust the outdoor environment before the in-air flight. In this experiment, we put a TX2 and a quadcopter in a tripod, and train it with MGAT. After making it like Fig. 7, we proceeded the MGAT algorithm to learn about the quadcopter attitude controller.

This is the 20th trained result (Table II, Fig. 8). The stabilization of the attitude controller can be confirmed clearly during the experiment. According to this graph, the roll is getting better in following the setpoint than the early part.

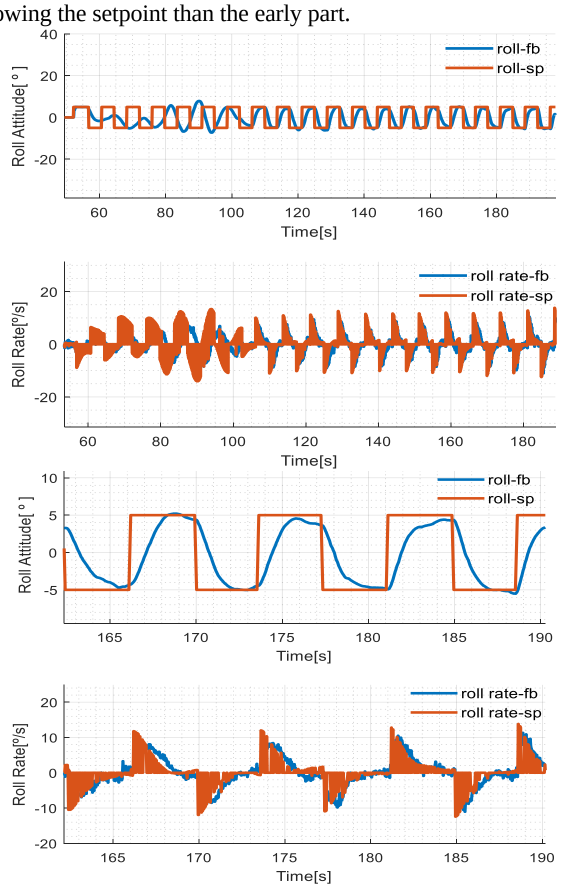

Figure 6. The Attitude and Angular Rate Performance During The Tuning Process of the proposed MGAT Method in Gazebo

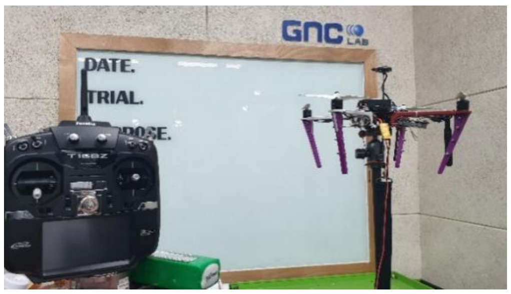

Figure 7. Hardware-in-The-Loop Test Bench with The F450 Multicopter

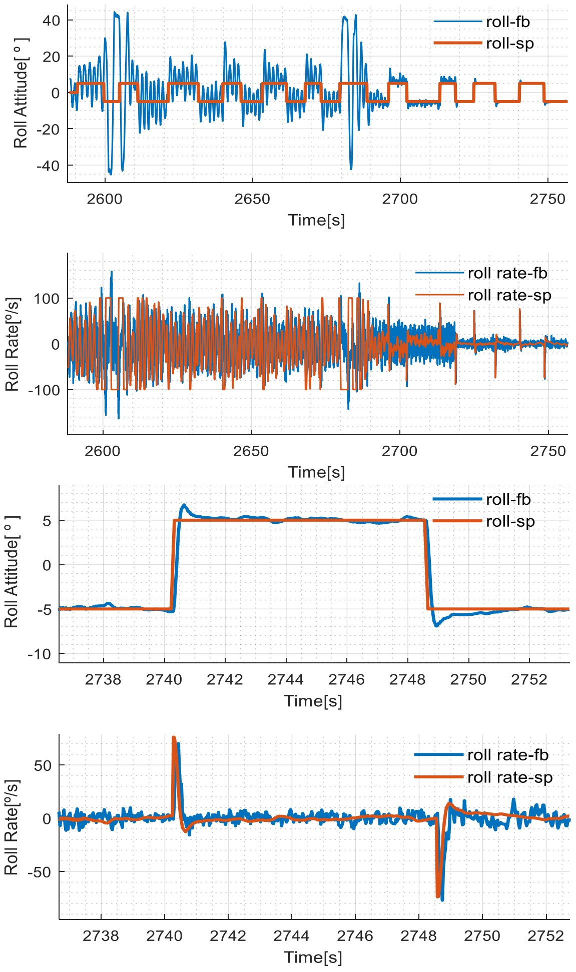

Figure 8. The Attitude and Angular Rate Performance During The Tuning Process of the proposed MGAT Method

The blue line is the actual movement and the orange is the setpoint. Both in roll and roll rate, approximately after 15 episodes, the actual movement follows the setpoint well than the previous episodes.

Table 2 Gain Conversion Process Result

Name	Angle P	Rate P	Rate I	Rate D
Before	7.2	0.1	0.35	0.024
After	7.3	0.20	0.20	0.012
Default	6.5	0.15	0.05	0.003

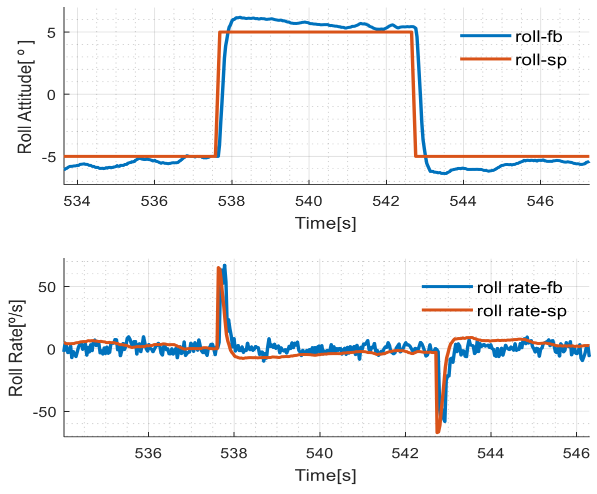

Figure 9. The Attitude and Angular Rate Performance of The Attitude Controller with Default Controller Gains

As expected, inner loop gains (It is D gain of outer loop) has increased more than before, and the other roll P gains also (Fig. 9). In this result, we can find out that the MGAT algorithm has better results for our requirement as we mentioned in the following setpoint correctly compared to the default gain tuned one.

6. CONCLUSIONS

Through the simulation and experimental results, we demonstrated that new method can tune the PID attitude controller gains automatically and the tuned controller exhibits satisfactory performance. Particular vehicles based on PX4 can use this method with at least 20 episodes. The MGAT is applicable to various multicopter platforms without prior knowledge of the system parameters. Future work will focus on the experiment of MGAT algorithm using real multicopters flying in disturbance-rich environments.

7. ACKNOWLEDGMENT

This research was supported by the MSIT (Ministry of Science and ICT), Korea, under the ITRC (Information Technology Research Center) support program (IITP-2019-2018-0-01423) supervised by the IITP (Institute for Information & communications Technology Promotion).

This work was also supported by the Technology Innovation Program (10077717, Development of the Low Weight and High Efficiency All-in-One Electric Drive Module for Multicopters) funded By the Ministry of Trade, Industry & Energy (MOTIE, Korea).

8. REFERENCES

[1] Xuan-Mung, N.; Hong, S. K. Improve Altitude Control Algorithm for Quadcopter Unmanned Aerial Vehicles, *Applied Sciences*, vol. 9, 2122. 2019.

[2] Pounds, P.E.I.; Bersak, D.R.; Dollar, A.M. Stability of small-scale UAV helicopters and quadrotors with added payload mass under PID control. *Auton Robot*., vol. 33, pp. 129–142, 2012.

[3] Salih, A.L.; Moghavvemi, M.; Mohamed, H.A.F.; Gaeid, K.F. “Modelling and PID controller design for a quadrotor unmanned air vehicle,” in *Proc. AQTR '10* , May 2010, pp. 1-5.

[4] Li, J.; Li, Y., “Dynamic analysis and PID control for a quadrotor,” in *Proc. IEEE International Conference on Mechatronics and Automation*, 7–10 Aug. 2011.

[5] Ahmed, A.H.; Ouda, A.N.; Kamel, A.M.; Elhalwagy, Y.Z., “Attitude stabilization and altitude control of quadrotor,” in *Proc. ICENCO*, 28–29 Dec. 2016, pp. 578.

[6] Khan, H.S.; Kadri, M.B., “Attitude and altitude control of quadrotor by discrete PID control and non-linear model Predictive control,” in *Proc. ICICT*, 12–13 December 2015.

[7] Nguyen, N.P.; Hong, S.K, “Position control of a hummingbird quadcopter augmented by gain scheduling,” *Int. J. Eng. Res. Technol*, Vol. 11, pp. 1485–1498, Nov. 2018.

[8] Xuan-Mung, N.; Hong, S.K, “A Multicopter ground testbed for the evaluation of attitude and position controller,” *Int. J. Eng. Technol*. Vol. 7, pp.65–73, Jul. 2018.

[9] Xuan-Mung, N.; Hong, S.K, “Robust Adaptive Formation Flight Control of Quadcopters based,” *Leader-Follower Approach. International Journal of Advanced Robotic System*, Vol. 16, pp. 1–11, Jul. 2019.

[10] (2019) The ROS (Robot operation system) website. [Online]. Available: https://www.ros.org/.

[11] L.P.Kaelbling, M.L.Littman, A.W.Moore, “Reinforcement Learning: A Survey.”, vol. 4, pp.237-285, May. 1996.

[12] Tommi Jaakkola, “Reinforcement Learning Algorithm for Partially Observable Markov Decision Problems,” in *NIPS'94*, 1994, p.345-352.

[13] Volodtmyr Mnih. “Playing Atari with Deep Reinforcement Learning,”
in *NIPS Deep Learning Workshop 2013*, 2013, vol. abs/1312.5602.

[14] (2019) The Keras website. [Online]. Available: https://keras.io/.

[15] (2019) The MAVROS website. [Online]. Available: http://wiki.ros.org/mavros.

[16] (2019) The Rospy website. [Online]. Available: http://wiki.ros.org/rospy.

[17] (2019) The TX2 website. [Online]. Available: https://www.nvidia.com/.