Application Of Reinforcement Learning In Heading Control Of A Fixed Wing UAV Using X-Plane Platform

Kimathi, S., Kang'ethe, S., Kihato, P

Abstract: Heading control of an Unmanned Aerial Vehicle, UAV is a vital operation of an autopilot system. It is executed by employing a design of control algorithms that control its direction and navigation. Most commonly available autopilots exploit Proportional-Integral-Derivative (PID) based heading controllers. In this paper we propose an online adaptive reinforcement learning heading controller. The autopilot heading controller will be designed in Matlab/Simulink for controlling a UAV in X-Plane test platform. Through this platform, the performance of the controller is shown using real time simulations. The performance of this controller is compared to that of a PID controller. The results show that the proposed method performs better than a well tuned PID controller.

Keywords: UAV, Reinforcement Learning, PID, X-Plane

Introduction

An unmanned Aerial vehicle, UAV is a space traversing vehicle that flies without a human crew on board. UAVs can be remotely controlled, semi-autonomous, autonomous or a combination of these. They are the future of aerial vehicles and they present an area of great interest to the control engineering fraternity. UAVs have a wide range of applications including surveillance, search and rescue, target tracking, digital mapping and weather observations. To accomplish these autonomous missions, it is essential to have a reliable heading control i.e. lateral direction control, thus an autopilot system is used. Autopilots were first developed for missiles but later extended to aircrafts and ships. Due to the nonlinearity of the system dynamics and parameter uncertainty in UAVs, several control techniques including PID control [1] [2], where two PID controllers were used in tandem, for the lateral and longitudinal motions. In [3] H ∞ control strategy was used. Adaptive control strategies have been applied; fuzzy systems in [4] [5], active disturbance rejection control, ADRC in [6] [7]. In [7] ADRC was applied to stabilize the UAV during aeriel refueling. In [8] an adaptive backstepping approach was used to obtain directional control of a fixed wing UAV, where the dynamics of the cross track error was derived using the lateral system equations of motion. This paper presents an adaptive control strategy based on reinforcement learning technique. This is due to the high nonlinearity of the system dynamics associated with small flying aerial vehicles and lack of complete knowledge vehicle dynamics for parameter estimation.

- *Kimathi S., Department of Electrical and Electronic Engineering, DeKUT (cell: +254 724 422 204; kimathisteve@gmail.com).*
- *Kang'ethe S., and Kihato P., Department of Electrical, Electronic and Information Engineering, JKUAT (samuelkangethe7@gmail.com, kamitakhat1@yahoo.com)*

Reinforcement learning explores actions from available courses of action and chooses the best course of action based on the reward it gets, hence suitable for this kind of application. A nonlinear model of a small sized fixed wing UAV is taken, which is linearized about a stable trim point and decoupled into longitudinal and lateral designs. The lateral design will be used in design of the controller. The proposed controller will act on the deflection angles of the two lateral control surfaces i.e. the aileron and rudder. This rest of this paper is organized as follows: Section II presents the basics of UAV control, section III introduces reinforcement learning principles, section Iv gives a brief preview of X-Plane test platform, section V gives a design of the controller, section VI provides the results and discussion and the conclusion is given in the last section.

UAV Control Basics

A brief discussion of a UAV control basics is considered. Figure 1 shows the UAV can move about the axes of motion (x,y,z) from its centre of gravity [9].

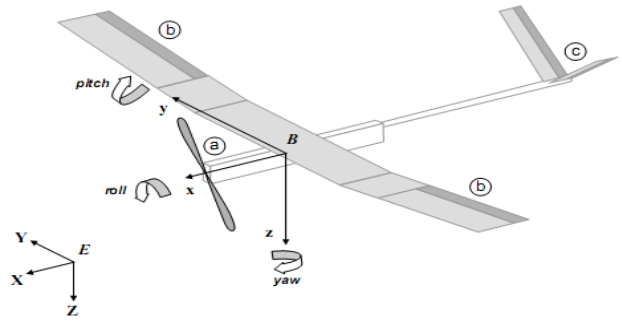

Figure 1: *UAV axes*

The position control of the UAV is converted to angular control in the three principle control axes: roll (φ), pitch (θ) and yaw (Ψ). The control surfaces for a fixed wing UAV are as illustrated in Figure 2:

- Ailerons to control the rolling
- Elevator to control the pitching
- Rudder to control the yawing movement

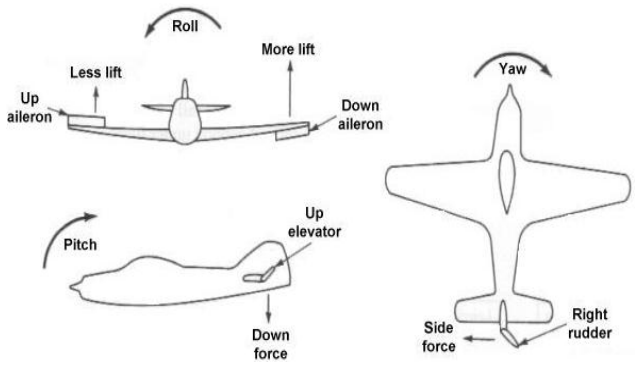

Figure 2: *UAV control surfaces*

In addition to these three control surfaces, the engines throttle controls the engines power. The derivation of equations of motion for fixed wing UAV is given in [10] [11]. The nonlinear equations of motion are linearized around a level flight trim condition and the linear models thus obtained are used to cater for the aircraft natural longitudinal and lateral modes. Thus a lateral state space model decoupled from within the linear model is then used with inputs of aileron and rudder to control the heading of an aircraft [12]. The decoupled lateral model state space equation is given as

$$\dot{x} = A_{lat}x_{lat} + B_{lat}u_{lat}$$

Where x_{lat} is the decoupled lateral state space model with $[\rho\ \beta\ r\ \varphi]^T$ as the state variables. ρ is the roll rate, β is the sideslip angle, r is the yawing rate and φ is the roll angle of a UAV. u_{lat} is the control input and comprises of δ_a the aileron deflection and δ_r the rudder deflection. A_{lat} is the state matrix and B_{lat} the input matrix. The linear state space model is given as in [11] as

$$\begin{bmatrix}\dot{p}\\ \dot{\beta}\\ \dot{r}\\ \dot{\psi}\end{bmatrix} = \begin{bmatrix}L_p & L_\beta & L_r & 0\\ Y_p & Y_\beta & Y_r - 1 & mg\cos\theta_e\\ N_p & N_\beta & N_r & 0\\ 0 & 0 & 1 & 0\end{bmatrix}\begin{bmatrix}p\\ \beta\\ r\\ \phi\end{bmatrix} + \begin{bmatrix}L_{\delta a} & L_{\delta r}\\ Y_{\delta a} & Y_{\delta r}\\ N_{\delta a} & N_{\delta r}\\ 0 & 0\end{bmatrix}\begin{bmatrix}\delta_a\\ \delta_r\end{bmatrix}$$

Reinforcement Learning

Reinforcement learning, RL is learning what to do – how to map situations to actions, so as to maximize some numerical reward. The learning agent is not told the correct actions; instead it explores the possible actions and remembers the reward it receives. It is inspired by natural learning mechanisms where animals adjust their actions based on the reward or punishment stimuli received from interacting with the environment [13]. The RL model consists of a set of environment states $s_t \in S$; a set of actions $a_t \in A$ that an agent can perform at each state, and as a consequence of its action, the agent receives a numerical reward r_t. At each time step, an agent implements a mapping from states to probabilities of selecting each possible action. This mapping is called the agent's policy and denoted as π_t which maximizes the cumulative reward of an agent over time as [13]

$$R = \sum_{t=0}^{\infty} \gamma^t r_t$$

where $0 < \gamma < 1$ is a discount factor, which reduces the value of future rewards. In reinforcement learning there is dynamic programming, DP Monte Carlo methods and temporal difference, TD method which comprise of Q-learning and sarsa algorithm where the latter is an online learning method [14]. Like Monte Carlo methods, TD methods can learn directly from raw experience without a model of the environment dynamics and like DP, TD methods updates estimates based in part on other learned estimates without having to wait for a final outcome [13]. The simplest TD method is given as

$$V(s_t) \leftarrow V(s_t) + \alpha[r_{t+1} + \gamma V(s_{t+1}) - V(s_t)]$$

The temporal difference update is $r_{t+1} + \gamma V(s_{t+1})$. SARSA being an online TD method, estimates $Q^\pi(s,a)$ for the current behavior policy π and for all states s and actions a. This is done using the same TD method described above but the transitions from state-action pair to state-action pair is considered rather than from state to state and hence the value of the state-action pairs.

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

This update is done after every transition from a non terminal state s_t. Thus in sarsa we continually estimate Q^π for the behavior policy π and at the same time change π towards greediness with respect to Q^π.

X-Plane

X-Plane is powerful flight simulator for personal computers. It is not a game but rather an engineering tool that can be used to predict the flying qualities of fixed and rotary wing aircraft with considerable accuracy [15]. The accuracy of X-Plane makes it a useful tool to predict and test the performance of an aircraft and its characteristics. It has the capacity to send and receive data to and from other devices. This is achieved using the User Datagram Protocol, UDP. UDP uses a simple transmission method without explicitly handshaking, ordering or data integrity. Hence, UDP provides a fast communication due to less overhead of network level processing thus suitable for this time-sensitive real time applications as in X-Plane [16]. X-Plane is able to send and receive data at 99.9 data packets per second via UDP. Each data packet is configured to carry specific aircraft parameters that are checked in the check boxes provided in the X-Plane Input and Output data interface as shown in Figure 3. For instance, in the case of lateral motion, parameters such as roll, yaw, heading rate, roll rate, aileron and rudder stick deflections should be checked. Once these are selected on X-Plane, through UDP and loopback addresses it is possible to receive them in Matlab/Simulink. This is made possible by using X-Plane Communication Library which enables communication

between Matlab/Simulink and X-Plane as in Figure 5; receiving and sending. Whereby the received packets are repackaged for use in Matlab/Simulink environment and the sent data is

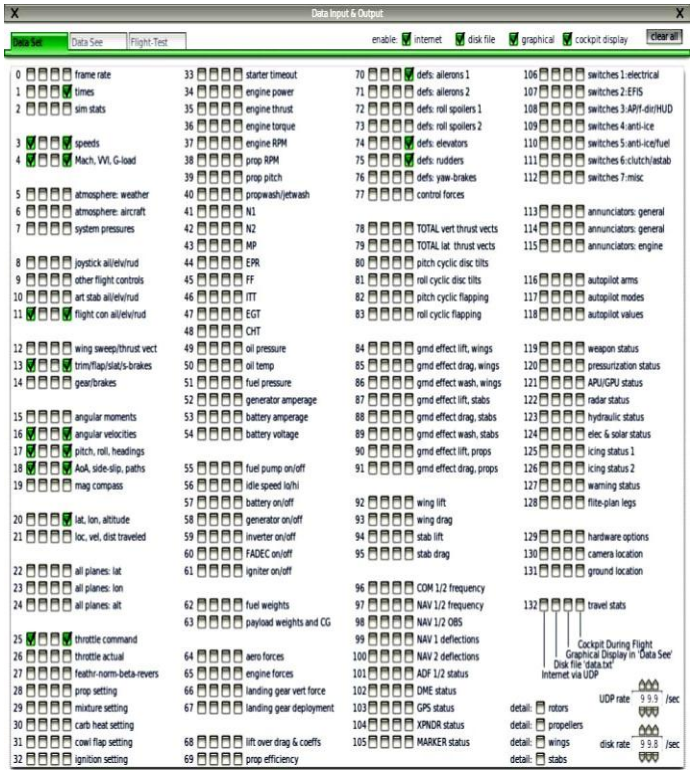

Figure 3: *X-Plane Test platform UDP communication*

Repacked in a format that can be received and processed by X-Plane. X-Plane also has the functionality of altering the weather conditions i.e. wind speed, shear speed and direction and turbulence of an altitude layer as in Figure 4. This allows for close to real life flying conditions hence a robust simulation environment.

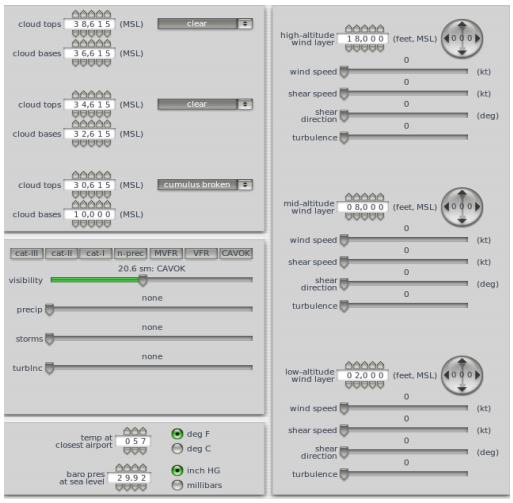

Figure 4: *X-Plane atmospheric layers*

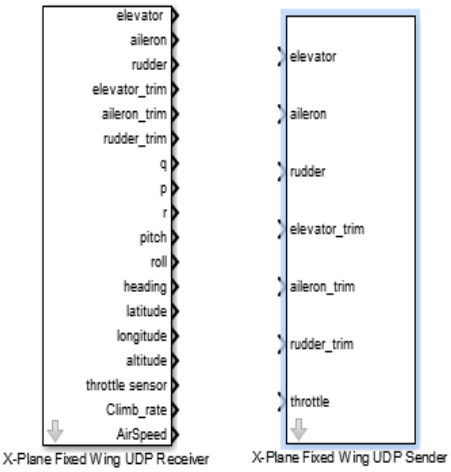

Figure 5: *Simulink to X-Plane Communication Library*

Controller Design

The main control objective is to obtain lateral-directional control in order to follow a desired reference heading. SARSA algorithm is used to design the controller. From the state space model presented above, the associated Ricatti coefficient, P is calculated which is formulated as the Algebraic Ricatti Equation, ARE. This Ricatti coefficient is used to calculate the Cost function for each state-action pair using a simple Lyapunov function $V = X^T \mathbf{P} X$, which is reformulated to include references as $Q(s,a) = (X - Z)^T \mathbf{P} (X - Z)$ where X are states and Z are references. A reward function is calculated as the deviation of the target state from the desired state as in [17] which in this work is taken as the heading error.

$$r(s) = -C_1(\emptyset - \emptyset_{ref})$$

The total value function is calculated, which is a sum of previous state-action value function, the current reward and the current state-action value function as

$$Q(s_1, a_1) \leftarrow Q(s_1, a_1) + \alpha[r_2 + \gamma Q(s_2, a_2) - Q(s_1, a_1)]$$
$$Q(s_2, a_2) \leftarrow Q(s_2, a_2) + \alpha[r_3 + \gamma Q(s_3, a_3) - Q(s_2, a_2)]$$
$$\vdots$$

This is updated as the total value function for the next cycle of learning. According to [18] it is allowed to have a one step gradient search of the value function - exploitation for ease of real time implementation and less computational burden. This was achieved as the temporal difference which is evaluated as

$$\delta_t = r_{t+1} + \gamma Q(s_{t+1}, a_{t+1}) - Q(s_t, a_t)$$

According to [19], an optimal control effort is given as $u^* = -KX$ but according to differential games algorithm, this is expressed as $u^* = -\frac{1}{2} R^{-1} g^T \nabla V^*$ where ∇V^* is taken as the change in the optimal cost function – which in this work the temporal difference between successive cost functions is used as the reinforcement to the optimal control. This optimal control implemented this way is considered to be compensated of the error(s)/ deviation from the reference signal hence the best control effort. The

control signals from a feed forward neural network are compared with this control signal. Then back propagation algorithm updates the feedforward neural network weights using back propagation. This implies that we are correcting the error in the control deflection in the next control deflection through update of weights thus slowly taking our deflections to the best available control effort in each consecutive cycle greedily. In this work the issue of exploration which is also central to RL alongside exploitation is not addressed. Here the RL controller only exploits the value function and new states are found by inference of favourable value functions.

SIL implementation

The aerodynamic coefficients that constitute the values in the mathematical model as provided for in [11] are taken from [3] and [12] as $\mathrm{m} = 1.9\mathrm{kg}, \mathrm{b} = 1.2\mathrm{m}, \mathrm{g} = 9.8\frac{\mathrm{m}}{\mathrm{s}^2}$, $S = 0.32\ \mathrm{m}^2, \bar{c} = 0.3\ \mathrm{m}, \rho = 1.225\mathrm{kg/m}^3, \frac{1}{\pi e \mathrm{AR}} = 0.0815$

Aerodynamic Coefficients

$C_{L_0} = 2.30 \times 10^{-1}$	$C_{L_\alpha} = 4.58$	$C_{L_{\delta_e}} = 1.30 \times 10^{-1}$
$C_{L_\alpha} = 1.97$	$C_{L_q} = 7.95$	$C_{L_{min}} = 2.30 \times 10^{-1}$
$C_{D_0} = 4.42 \times 10^{-2}$	$C_{D_{\delta_e}} = 1.35 \times 10^{-2}$	$C_{D_{\delta_r}} = 3.03 \times 10^{-2}$
$C_{Y_\beta} = -8.30 \times 10^{-1}$	$C_{Y_{\delta_r}} = 1.91 \times 10^{-1}$	$C_{Y_p} = 0$
$C_{Y_r} = 0$	$c_{l_\beta} = -1.30 \times 10^{-1}$	$c_{l_{\delta_a}} = 8.55 \times 10^{-2}$
$c_{l_p} = -5.05 \times 10^{-1}$	$c_{l_r} = 2.52 \times 10^{-1}$	$c_{m_\alpha} = -1.50$
$c_{m_{\delta_e}} = -9.92 \times 10^{-1}$	$c_{m_{\dot{\alpha}}} = -1.04 \times 10^{1}$	$c_{m_q} = -3.82 \times 10^{1}$
$c_{n_\beta} = 7.26 \times 10^{-2}$	$c_{n_{\delta_r}} = -6.93 \times 10^{-2}$	$c_{n_p} = -6.90 \times 10^{-2}$
$c_{n_r} = -9.46 \times 10^{-2}$		

Moment of Inertia

$I_{xx} = 8.94 \times 10^{-2}$	$I_{yy} = 1.44 \times 10^{-2}$	$I_{zz} = 1.62 \times 10^{-2}$
$I_{xz} = 1.40 \times 10^{-2}$	$I_p = 1.30 \times 10^{-2}$	$I_m = 1.30 \times 10^{-4}$

With the above parameters, the trim condition was obtained as:

$$\mathrm{A} = \begin{bmatrix} -1.4000 & 0 & 0 & 9.4953 \\ -30.9000 & -12.8000 & 14.4000 & 0 \\ 1.4781 & -0.4480 & -6.080 & 0 \\ 0 & 1.0000 & 0 & 0 \end{bmatrix}$$

$$\mathrm{B} = \begin{bmatrix} 0 & 0.7412 \\ 61.4000 & 12.4000 \\ -3.6700 & 15.0000 \\ 0 & 0 \end{bmatrix}$$

Two simulations were carried out in Matlab/Simulink, one using a well tuned PID controller and another using the reinforcement learning method described above. Then the two designed controllers were used for real time UAV control in X-Plane as in Figure 6 below.

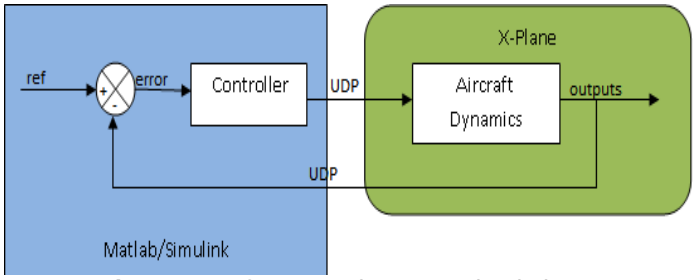

Figure 6: *Software-In-the-Loop Simulation*

Results and Discussion

The state space model given above was used to design controllers and hence get the response of the heading rate in Matlab/Simulink. Using a well tuned PID controller and the method highlighted in [1], the response of the heading rate is as shown in Figure 7. The dark solid line shows the reference signal and the red solid line the heading rate response of the model. Figure 8 presents the heading rate response of the model using reinforcement learning controller. It can be seen that the controller utilizing reinforcement learning has better tracking response as compared to a well tuned PID controller; has no overshoots and tracks the reference as closely as possible.

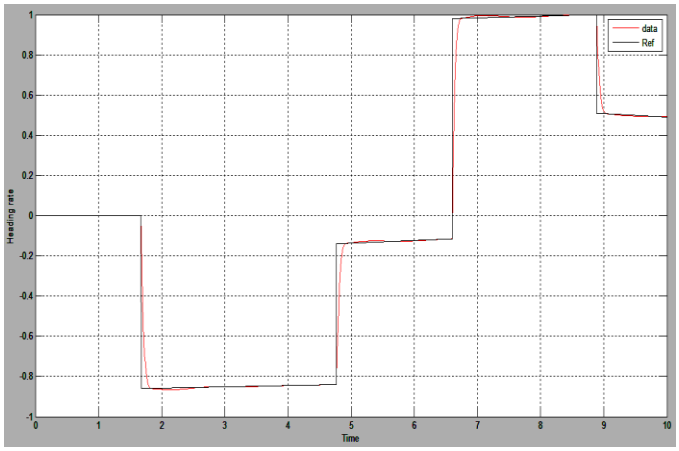

Figure 7: *Heading rate response of the PID controller*

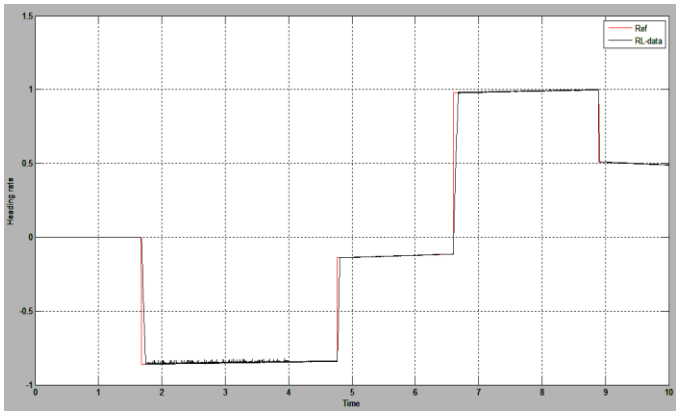

Figure 8: *Heading rate response of the RL controller*

The designed controllers that gave the above results were integrated to become actual real time controllers for a UAV in X-Plane platform. At first, a single step heading

reference was designed for an actual flight regime of 50^0 initial heading to 100^0 final heading and the results were as below. Figure 9 shows the PID response to the step reference. The rise time is 1.4007 seconds and the system does not settle within the bounds of 0.5% of final value that was specified. The percentage overshoot was calculated as 6.3119%., which is close to that reported in [1] where a mathematical model was used.

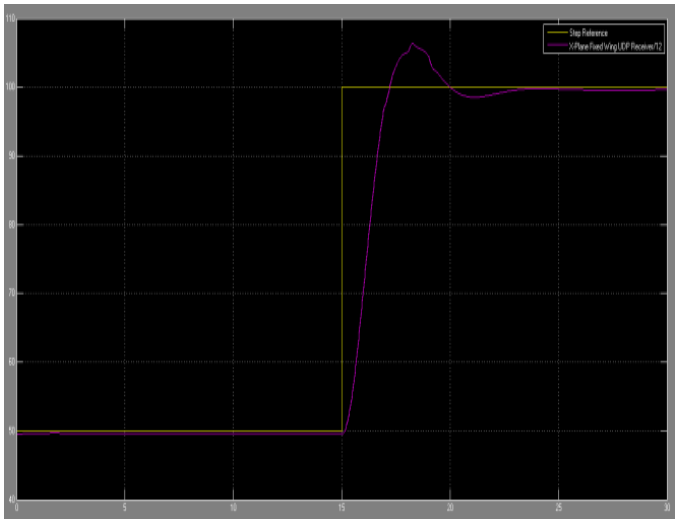

Figure 9: *PID controller response to a step heading reference*

Figure 10 shows the step reference for a RL controller. The rise time is 1.2663 seconds as compared to PID's 1.4007 seconds and the system settles after 21.371 seconds. As can be seen the overshoot is much lesser than the PID's. The percentage overshoot is 3.1083% for the RL controller.

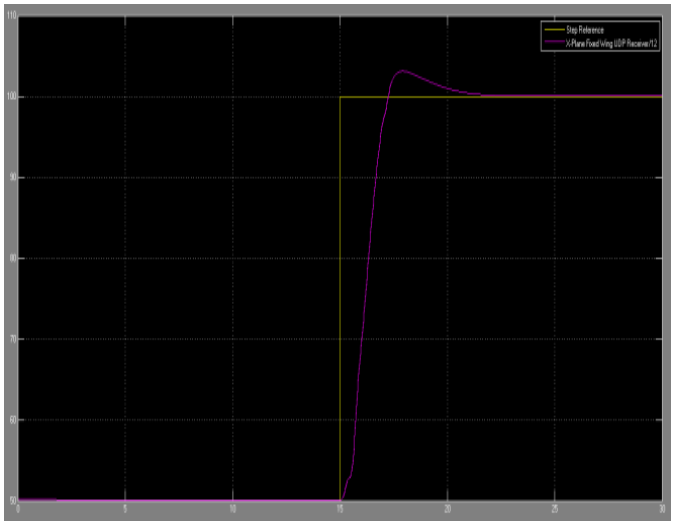

Figure 10: *RL controller response to a step heading reference*

The preceding results are for UAV simulation in which it is commanded from an initial heading of 40^0. The aircraft is then commanded to go to 80^0 and then 50^0 and so on. The Figures below display the results where the heading in degrees is plotted on the vertical axis against time in seconds on the horizontal axis. Figure 11 shows the performance of the PID controller for real time heading control of a UAV. Figure 12 shows the response of the Reinforcement learning controller for the same heading control of a UAV. For the RL controller, the first 10 seconds the response is poor as compared to the PID controller; this is due to the fact that the artificial neural network weights are being continually adjusted where initially big adjustments are expected then they settle around the optimum weights. Thus after 10 seconds the response stabilizes and follows the reference more robustly than the PID controller. It can also be seen that there is an overshoot on the first step heading change but due to adaptation that overshoot is eliminated in the consecutive step heading angle changes as is evident from the Figure 12. PID controller performs well to tracking the reference initially from any random position but its tracking response is poor, it overshoots in every step reference heading change and does not track the reference robustly as is illustrated in Figure 11.

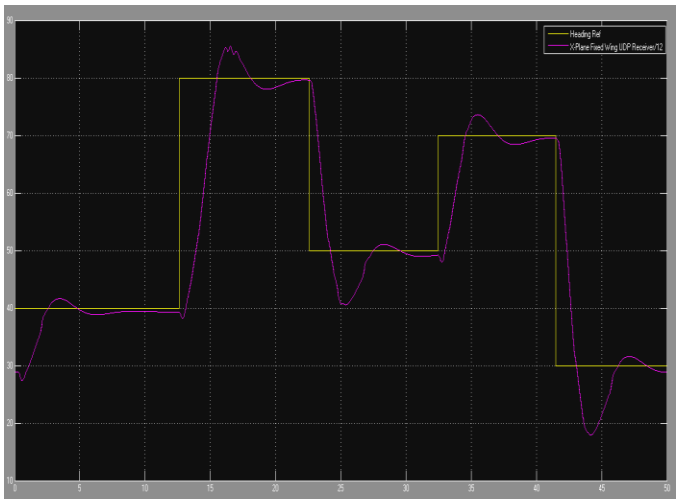

Figure 11: *PID controller response to real time heading control in X-Plane*

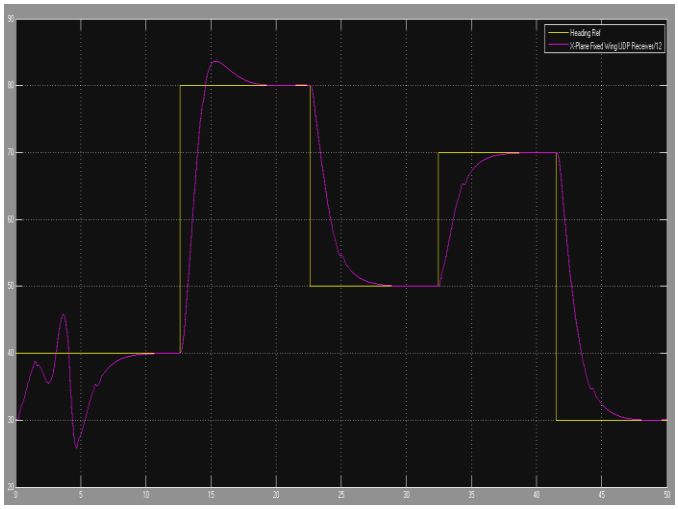

Figure 12: *RL controller response to real time heading control in X-Plane*

Conclusion

In this paper, an online adaptive reinforcement learning controller for heading control in a fixed wing UAV is presented. As it has been shown using simulations in X-Plane, the RL controller produced better tracking results than a well tuned PID controller. The results indicate that an online tuned adaptive controller allowed the UAV to achieve more satisfactory performance by eliminating overshoots while tracking a desired reference heading.

References

[1] A. Mansoor et. al, "Heading control of a FIxed wing UAV using Alternate control surfaces," IEEE, vol. Vol. 2, December 2012.

[2] Z. Qing and G. Zhiqiang, "On Practical application of Active Disturbance Rejection Control," in Proc. of the 29th Chinese Control Conference, Beijing, 2010, pp. 6095-6100.

[3] H. Ferreira et.al, "Disturbance Rejection in a Fixed WIng UAV using nonlinear H-Infinty stste feedback," in 9th International Conference on Control and Automation, Santiago, USA, 2011.

[4] M. Prabhudas and V.Nagababu, "A Fuzzy logic strategy on attitude controlling of Longitudinal autopilot for better disturbance rejection," International Journal of Engineering Research and Technology, vol. Vol. 2, no. Issue 12, pp. 186-190, December 2013.

[5] D. Stojcsics, "Fuzzy controller for small size Unmanned Aerial Vehicles," in 10th IEEE Intl. Symposium on Applied machine Intelligence and Informatics, Herl'any, Slovakia, 2012, pp. 91- 95.

[6] X. Hua et. al, "Disturbance rejection in UAV's velocity and Altitude Control: Problems and Solutions," in Proc. of the 30th Chinese Control Conf., Yantai, China, 2011, pp. 6293-6298.

[7] L. Jing-Mei and Z. Ke, "Design of Active Disturbance Rejection Controller for Autonomous Aerial Refuelling UAV," in IEEE, 2013.

[8] A. Brezoescu et. al, "Adaptive Trajectory following for a Fixed wing UAV in Presence of Crosswind," Journal of Intelligent Robotic Systems, vol. Vol. 69, pp. 257-271, 2013.

[9] A. Noth et. al, "Dynamic Modelling of Fixed Wing UAVs," Swiss Federal Institute of Technology, Zurich, Laboratoty Report 2006.

[10] R. Nelson, Flight Stability and Automatic Control, 2nd ed. Singapore: McGraw Hill, 1998.

[11] R. Beard and T. McLain, Small Unmanned Aircraft; Theory and Practice, 1st ed. Princeton, New Jersey: Princeton University Press, 2012.

[12] Y. Paw, "Synthesis and Validation of Flight Control for UAV," University of Minnesota, Miinnesota, PhD Thesis Dec. 2009.

[13] R. Sutton and A. Barto, Reinforcement Learning; An Introduction, 1st ed. Massachusetts: MIT press, 2005.

[14] S. Rusell and P.Norvig, Artificial Intelligence; A Modern Approach, 3rd ed. New Jersey: Pearson Education, 2010.

[15] A. Meyer, X-Plane 10: Operation Manual., 2012.

[16] R. Lucio and O. Neusa, "UAV Autopilot Controllers Test Platform Using Matlab/Simulink and X-Plane," in 40th ASEE/IEEE Frontiers in Education Conference, Washington, Dc, 2010, pp. pg 6262-6269.

[17] B. Haitham et. al, "Controller Design for Quadrotor UAVs using Reinforcement Learning," in IEEE International Conference on Control Applications, Yokohama, Japan, 2010, pp. 2130-2135.

[18] R. Sutton et. al, Experiments with Reinforcemnet Learning in Problems with Continous State and Action Spaces, Unpublished.

[19] B. Anderson and J. Moore, Optimal Control, Linear quadratic Methods. London, UK: Prentice-Hall International, 1989.