Auto-tuning of PID parameters using Q learning

Deepak Rajasekhar Karishetti, *10846936* Ramprasad Rajagopalan, *10846647*
Siddharth Viswanathan, *10831326* Vicknesh Balabaskaran, *10847953*

Abstract

Auto-tuning of PID paramters using Q-learning is a project that was an attempt in controlling a quadrotor by tuning the PID paramters using a reinforcement learning technique. The quadrotor modelling, system dynamics and control theory to be used and implemented were self designed for our requirements in the project. Q-learning is the reinforcement tool used in tuning the PID parameters of the quadrotor. The controlling involved position control and attitudinal control using PID tuning specific to each instance of controlling variable over the entire system state inputs. Prior to the implementation of the reinforcement learning for tuning the parameters, the PID paramters were manually tuned and the desired trajectory was achieved. The system model and dynamics were accordingly validated for capability and performance of the quadrotor under specified system input states. In addition to the implementation of the RL technique, the performance of the RL was validated by setting disturbances to the system in-course of the travel from the origin to the designated goal state. Effectively, the implementation witnessed the successful traversing of the quadrotor to the designated target in the desired trajectory even when the system was perturbed with disturbances enroute.

I. Introduction

The system chosen is an X configuration quadrotor with the co-ordinates frames and labels given in Fig.1. A quadrotor has 4 rotors controlled by a flight controller and mounted on 4 arms. 2 rotors on diagonally opposite side rotates in clockwise direction while 2 other rotors rotate in anticlockwise direction. Fig.1 shows the body frame x - axis x_B and y - axis y_B and inertial frame x - axis $x_{Inertial}$ and y - axis $y_{Inertial}$. The forward direction of the quadrotor is x_B and roll about x_B is given by ϕ. The pitch θ is rotation about y_B and yaw ψ is rotation about z_B. Each rotating propeller produces an upward force given by,

$$F_i = k_f {\omega_i}^2 \tag{1}$$

where F_i is the thrust generated by each propeller, k_f is the thrust factor and ω_i is the anuglar velocity of each propeller. Apart from upward thrust, a rotating propeller also produces an opposing moment given by,

$$M_i = k_m {\omega_i}^2 \tag{2}$$

where M_i is the moment generated by each propeller about z_B and k_m is the drag factor. The dynamic equations of the system is derived as follows. The total thrust from the 4 rotating propellers is given by,

$$F_{total} = k_f\big({\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 + {\omega_4}^2\big) \tag{3}$$

The linear acceleration equations are obtained as,

$$\begin{bmatrix} \ddot{x} \\ \ddot{y} \\ \ddot{z} \end{bmatrix} = \begin{bmatrix} 0 \\ 0 \\ -g \end{bmatrix} + \frac{R}{mass} \begin{bmatrix} 0 \\ 0 \\ F_{total} \end{bmatrix} \tag{4}$$

where $R = R_z(\psi)R_y(\theta)R_x(\phi)$ is the rotation matrix from world frame to body frame obtained from corresponding rotation along the body axes. The moments along the respective axes are given by,

$$\tau_x = \frac{Lk_f}{\sqrt{2}}\big({\omega_1}^2 + {\omega_2}^2 - {\omega_3}^2 - {\omega_4}^2\big) \tag{5}$$

$$\tau_y = \frac{Lk_f}{\sqrt{2}}\big(-{\omega_1}^2 + {\omega_2}^2 + {\omega_3}^2 - {\omega_4}^2\big) \tag{6}$$

$$\tau_z = k_m\left({\omega_1}^2 - {\omega_2}^2 + {\omega_3}^2 - {\omega_4}^2\right) \tag{7}$$

The body frame angular accelerations are given by the Euler's equation for rigid body dynamics,

$$\begin{bmatrix} \ddot{\phi} \\ \ddot{\theta} \\ \ddot{\psi} \end{bmatrix} = I^{-1}\left(\begin{bmatrix} \tau_x \\ \tau_y \\ \tau_z \end{bmatrix} - \omega \times I\omega \right) \tag{8}$$

where $I = \begin{bmatrix} I_{xx} & 0 & 0 \\ 0 & I_{yy} & 0 \\ 0 & 0 & I_{zz} \end{bmatrix}$ and $\omega = \begin{bmatrix} \dot{\phi} \\ \dot{\theta} \\ \dot{\psi} \end{bmatrix}$ in the body frame. Therefore, the equations of motion from the above relations are written as follow,

$$\ddot{x} = \frac{F_{total}(cos(\phi)sin(\theta)cos(\psi) + sin(\phi)sin(\psi))}{mass} \tag{9}$$

$$\ddot{y} = \frac{F_{total}(cos(\phi)sin(\theta)sin(\psi) - sin(\phi)cos(\psi))}{mass} \tag{10}$$

$$\ddot{z} = \frac{F_{total}(cos(\phi)cos(\theta))}{mass} - 9.81 \tag{11}$$

$$\ddot{\phi} = \frac{\tau_x + (I_{yy} - I_{zz})\dot{\theta}\dot{\psi}}{I_{xx}} \tag{12}$$

$$\ddot{\theta} = \frac{\tau_y + (I_{zz} - I_{xx})\dot{\phi}\dot{\psi}}{I_{yy}} \tag{13}$$

$$\ddot{\psi} = \frac{\tau_z + (I_{xx} - I_{yy})\dot{\theta}\dot{\phi}}{I_{zz}} \tag{14}$$

Based on the above relations, one can derive the continuous states of the quadrotor given an input angular velocity of the propeller or the thrust and moments about corresponding axes. The state of the quadrotor i.e., position and orientation can be obtained through these relation. Proportional Integral Derivative (PID) control is commonly used for quadrotor to follow a trajectory. The simple nature of the control helps to

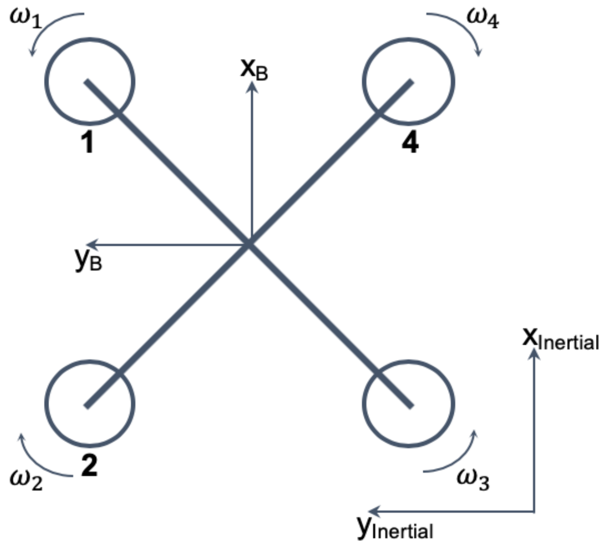


Fig. 1. Quadrotor coordinate frame

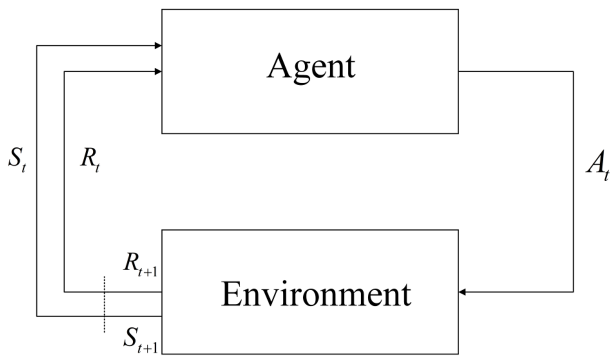


Fig. 2. Agent-environment interaction in RL

build control strategies for complex system models. The PID gains can be tuned manually or by using algorithms like Ziegler-Nichols methods. Quadrotor requires two cascaded PID loops to make it followa required trajectory. One to control its position in space (POS control) and using the information about where it is in space, the required orientation is controlled with another PID loop (Attitude control). This complex cascaded model makes tuning of PID parameters an arduous process. A lot of methods are implemented to make this problem easier but this study utilises Reinforcement learning (RL) viz., Q-learning to learn the appropriate PID parameters.

RL is an approach in machine learning which solves the problem assigned based on the end goal assigned to the statement. The basic working of RL is given in Fig.2 It works by generating a control policy that is developed from the interactions of the environment without any prior knowledge of the system model. This interactions-based problem solving observes states and actions. The updated state and action are sent as the input while the output is rewards. These rewards are usually scalar. The process works in a fashion where the end goal is to reach the desired end state by working with an optimal control policy receiving the maximum discounted rewards possible. All reinforcement learning problems are based off Markov Decision Process and the RL algorithms are broadly classified into value-based and policy-based algorithms. The approach used here is Q-learning which a value-based problem-solving algorithm. It is a model-free learning algorithm and works best on discrete state space with infinite inputs. Q-learning is used to learn a policy using the algorithm tells an agent to perform an action under certain circumstances. To compliment the learning process and the resulting output of the process, Q-learning gives rewards for the actions performed by the agent.The updating rule of Q-learning is shown below:

$$Q(S_t, A_t) = Q(S_t, A_t) + \alpha\big[R_{t+1} + \gamma \underset{a}{max}\; Q(S_{t+1}, a) - Q(S_t, A_t)\big] \tag{15}$$

In this study, we implement a Q learning algorithm to tune the PID parameters dynamically to achieve the desired trajectory. The Q model is learned for desired trajectory and implemented for two different trajectories. The simulation is observed for both normally tuned PID and auto-tuned Q learning PID.

II. Methodology

A. State-space model of the system

The continuous time state-space model of the quadrotor is obtained by by defining the state vector of the system,X, as follows,

$$X = \begin{bmatrix} x & y & z & \phi & \theta & \psi & \dot{x} & \dot{y} & \dot{z} & \dot{\phi} & \dot{\theta} & \dot{\psi} \end{bmatrix}$$

$$u = \begin{bmatrix} F_{total} & \tau_x & \tau_y & \tau_z \end{bmatrix}$$

where u is the input vector. Then, the continuous time state-space model can be written as,

$$\dot{X} = f_x(X, u)$$

where $f_x(X, u)$ is the non-linear function matrix mapping the first order derivative states to the system states and w is the process noise or disturbance in input. The non-linear function matrix, $f_x(X, u)$, can obtained. The quadrotor is modelled in Simulink as shown in Appendix.**??**. The total simulation time is 2.5s with time step, T$_s$ of 0.1s.

B. Trajectory generation

The quadrotor was desired to achieve two seperate trajectory, viz., diagonal of unit square in XZ plane and diagonal of a unit cube. The trajectory is generated by quadratic interpolation between way points defined at each individual time step. The interpolated values are considered as desired values of position and linear velocity. The interpolated values are compared with feedback values every 0.01s and given as error value for POS control.

C. Cascaded PID control

The cascaded PID control is implemented by having position control and attitude control blocks, shown in Fig.3. The feedback of the current states are obtained from the dynamic model.

1) POS control: The position control (POS control) is implemented by following,

$$\begin{bmatrix} err_x \\ err_y \\ err_z \end{bmatrix} = \begin{bmatrix} k_{px} & 0 & 0 \\ 0 & k_{py} & 0 \\ 0 & 0 & k_{pz} \end{bmatrix} \left(\vec{r}_{des} - \vec{r}_{cur} \right) + \begin{bmatrix} k_{dx} & 0 & 0 \\ 0 & k_{dy} & 0 \\ 0 & 0 & k_{dz} \end{bmatrix} \left(\dot{\vec{r}}_{des} - \dot{\vec{r}}_{cur} \right) \tag{16}$$

$$\Delta U_1 = err_z \tag{17}$$

$$R = \begin{bmatrix} sin(\phi) & cos(\phi) \\ -cos(\phi) & sin(\phi) \end{bmatrix} \tag{18}$$

$$\begin{bmatrix} \phi_{des} \\ \theta_{des} \end{bmatrix} = \frac{0.25}{\delta U_1} inv(R) \begin{bmatrix} err_x \\ err_y \end{bmatrix} \tag{19}$$

2) Attitude control: The attitude is implemented using the following equations,

$$\begin{bmatrix} \Delta U_2 \\ \Delta U_3 \\ \Delta U_4 \end{bmatrix} = \begin{bmatrix} k_{p\phi} & 0 & 0 \\ 0 & k_{p\theta} & 0 \\ 0 & 0 & k_{p\psi} \end{bmatrix} \left(\vec{\Omega}_{des} - \vec{\Omega}_{cur} \right) + \begin{bmatrix} k_{d\phi} & 0 & 0 \\ 0 & k_{d\theta} & 0 \\ 0 & 0 & k_{d\psi} \end{bmatrix} \left(\dot{\vec{\Omega}}_{des} - \dot{\vec{\Omega}}_{cur} \right) \tag{20}$$

The inputs to the system, ΔU_1, ΔU_2, ΔU_3 and ΔU_4, are used as simulation inputs and current states are generated by the dynamic model which are taken as feedback for tuning the PID. The parameters sets for traditional PID controls are given in Table.4.

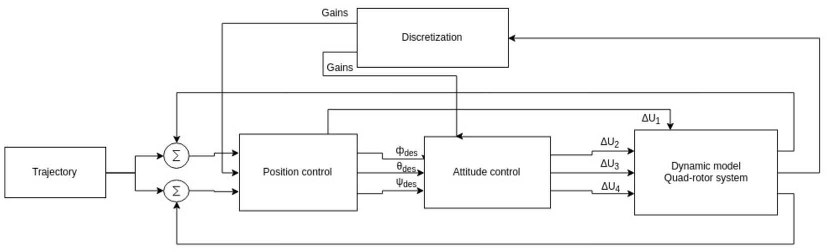


Fig. 3. PID control

Gain	Value	**Gain**	Value
$\mathbf{k}_{px}$	10	$\mathbf{k}_{p\phi}$	90
$\mathbf{k}_{py}$	10	$\mathbf{k}_{p\theta}$	95
$\mathbf{k}_{pz}$	20	$\mathbf{k}_{p\psi}$	20
$\mathbf{k}_{dx}$	20	$\mathbf{k}_{d\phi}$	58
$\mathbf{k}_{dy}$	20	$\mathbf{k}_{d\theta}$	58
$\mathbf{k}_{dz}$	20	$\mathbf{k}_{d\psi}$	0

Fig. 4. Gain values for PID control

D. Q learning training process

1) States and Actions: The Q-learning algorithm implemented in this study using 12 states and 12 actions in 26 time-steps. The states defined for the study can broadly be classified into four sets of three states which are given in Subsection.A. The actions used in this study are the gains used for the quadrotor. These gains can be broadly classified into two types which are the proportional gains and derivative gains. Twelve Q-tables are created for the 12 actions that are defined. The actions are a vector with elements that are chosen randomly. This random action assigned is based off the greedy-policy method. Though these vectors are randomly chosen, a bounding value is set for each of the actions based on the gains used in tuning the PID prior to RL implementation.

2) Discretization: It is essential to discretize the continuous inputs in quad-copter system considered because, as mentioned earlier, Q-learning works on discrete state spaces only. The process of discretization was done by dividing the continuous input variables into various buckets. The bucket size and the bucket limits were defined based on simulation performances. These limits were compared with the current input value and accordingly the input value was assigned a discretized value. The discretization condition is shown below,

$$n = \begin{cases} 1, & x_{con} < X_{min} \\ 26, & x_{con} > X_{max} \\ \lfloor \frac{x_{con}}{X_{max}-X_{min}} \times N \rfloor + 1, & X_{min} \leq x_{con} \leq X_{max} \end{cases} \tag{21}$$

where [x] = max {n $\in$ **Z** | n $\leq$ x}, n denotes the discrete variable, x$_{con}$ denoted the continuous variable, X$_{max}$ and X$_{min}$ are the lower and upper bound of x$_{con}$ and N denotes the number of buckets each variable is divided into.

3) ϵ - greedy method: All twelve Q-tables generate the actions according to ϵ - greedy method given the current state. This method is defined by

$$A = \begin{cases} random\ action, & \eta < \epsilon \\ arg\ \underset{a}{max}\ Q(s,a), & otherwise, \end{cases} \tag{22}$$

where η is a random number generated normally between 0 and 1. The exploration and exploitation is taken care by setting the following relation for ϵ based on episode,

$$\epsilon(eps) = \begin{cases} \frac{1}{1+e^{eps}} + 0.001, & eps < 0.6 \times maxepisodes \\ 0, & otherwise, \end{cases} \tag{23}$$

4) Reward scheme: The reward schemes are associated with the gains of the PID controllers, and are defined as follows,

$$R = \begin{cases} -2, & if\ |S_{t+1}| < |S_{des}| - S_{lim}\ or\ |S_{t+1}| > |S_{des}| + S_{lim} \\ 10, & |S_t| - S_{lim} < |S_{t+1}| < |S_{des}| + S_{lim} \\ 100, & |S_{t+1}| == |S_{des}| \end{cases} \tag{24}$$

where, |S$_{des}$| denotes the desired state values at that time, |S$_{t+1}$| denotes the state values for the next time step and |S$_t$| denotes the state values obtained for the current time step. The overall implemented Q-learning algorithm is given by Algorithm.1. The training hyper-parameters are listed in Table.6. The Q-learning training process is illustrated in Fig.5. The Simulink model of both the training process of Q-learning and auto-tuning of PID parameters are given in Appendix.A respectively.

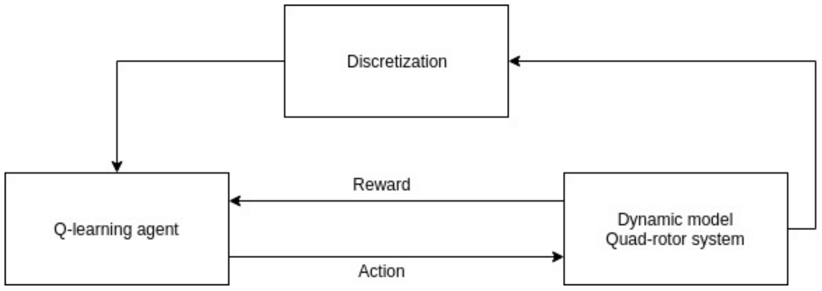


Fig. 5. Q learning training process

Parameters	Value
α	0.4
γ	0.99
S$_{lim}$	0.075
maxepisodes	50
Bin size, N	26
Total time, tottime	2.5s
Time step, T$_s$	0.1s

Fig. 6. Q learning hyper-parameters

The trained Q table was utilized in choosing the desired gain value for the discretized states. The results obtained in utilizing these values resulted in achieving the desired trajectory.

Algorithm 1 Q Learning Training Process

```
Initialize $Q_i$(s,a) = 0, $\forall$ a $\in$ A, $\forall$ s $\in$ S, i = 1,2,...,12
Initialize exploration rate $\epsilon$, learning rate $\alpha$ and discount rate $\gamma$

while episode < maxepisode do
    t = 0
    Initialize $S_t$(x(t),y(t),....,$\dot{\psi}$(t))
    Decay $\epsilon$ according to Eqn.23
    for t = 1;t $\leq$ maxtime;t++ do
        Discretization of state $S_t$, obtain $n_1$(t),...,$n_1$2(t)
        for i = 1;i $\leq$ 6;i++ do
            According to $n_1$(t),...,$n_6$(t), choose action $A_i$ following $\epsilon$ greedy policy
        end
        for i = 1;i $\leq$ 6;i++ do
            According to $n_7$(t),...,$n_1$2(t), choose action $A_i$ following $\epsilon$ greedy policy
        end
        Obtain simulation input u(t) according to Eqn.17 and Eqn.20
        Observe new state $S_{t+1}$(x(t),y(t),....,$\dot{\psi}$(t))
        Receive the reward R based on Eqn.24
        Discretization of states $S_{t+1}$(x(t),y(t),....,$\dot{\psi}$(t)), obtain $n_1$(t+1),...,$n_{12}$(t+1)
        Update $Q_{1,2,...,12}$(s,a) using R and $\alpha$ using Eqn.15
        $S_t$ = $S_{t+1}$
    end
end
```

E. Results

1) Traditional PID control: The traditional PID controller was implemented initially for the quadrotor, with its gains manually tuned. The end results obtained from the PID controller is as shown in the TABLE I.

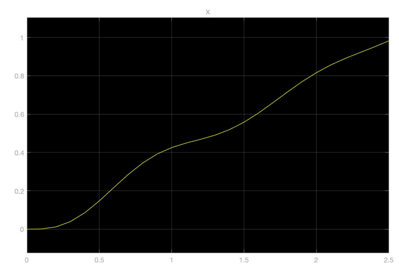
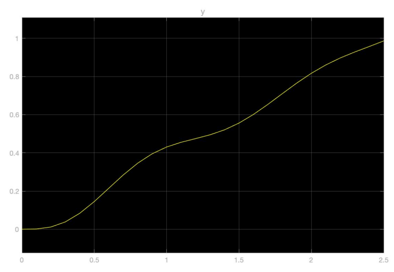
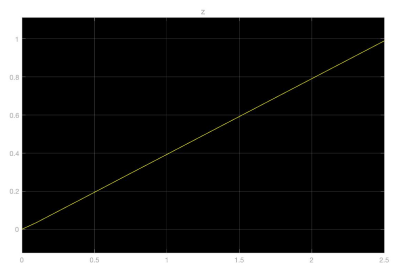
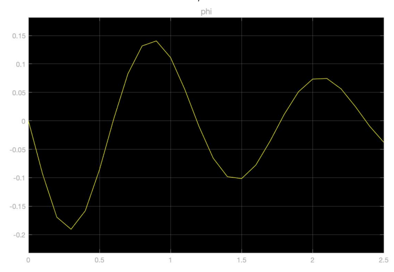
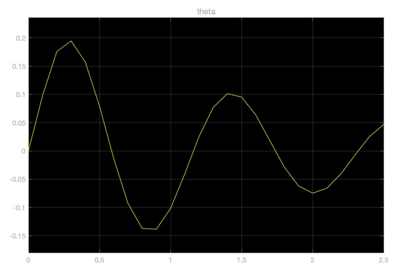
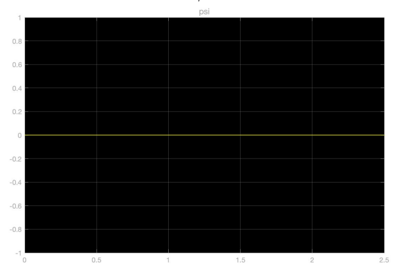
X	Y	Z
		
ϕ	θ	ψ
		

TABLE I
STATE VALUE OBTAINED USING TRADITIONAL PID CONTROL

2) Q-learning tuned PID control for quadrotor in normal environment: A Q-learning agent was implemented which is used to tune the PID controller gains for the quadrotor in the normal environment conditions with no external disturbances. The obtained end results for the learned model is as shown in the TABLE II.

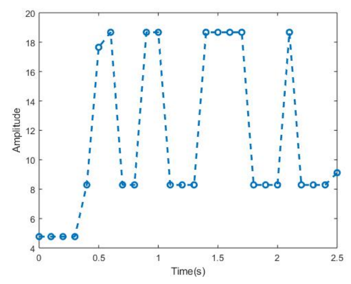
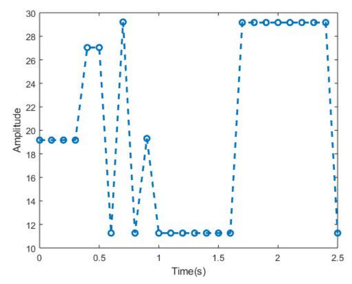
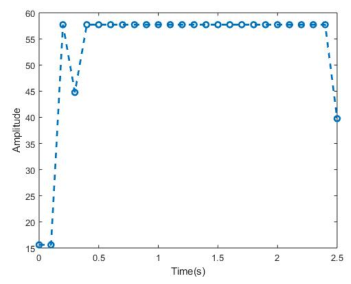
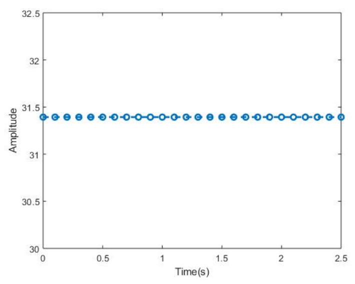
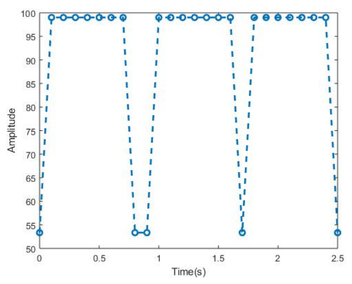
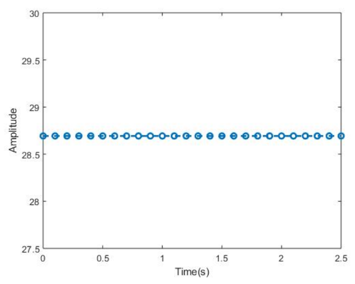
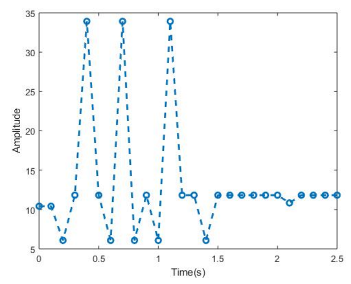
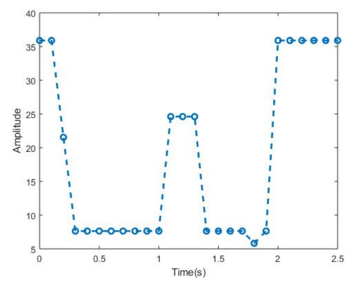
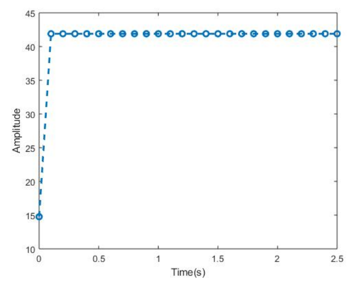
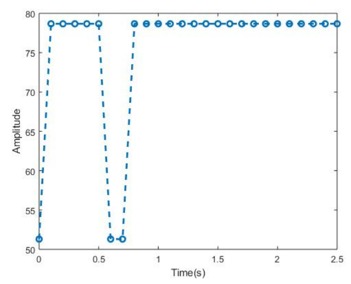
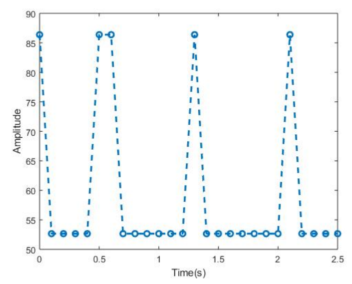
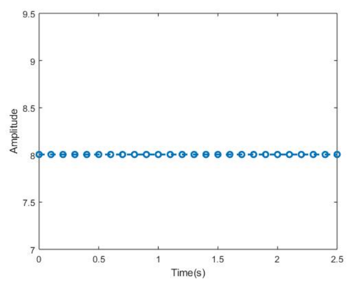
k_{px} 	k_{py} 	k_{pz} 
$k_{p\phi}$ 	$k_{p\theta}$ 	$k_{p\psi}$ 
k_{dx} 	k_{dy} 	k_{dz} 
$k_{d\phi}$ 	$k_{d\theta}$ 	$k_{d\psi}$ 

TABLE II
GAIN VALUE OBTAINED USING Q LEARNING TUNED PID CONTROL

The above plots show the respective k_p and k_d values chosen at each time interval during the simulation. These k_p and k_d values are the chosen with the indices with maximum Q value from respective Q matrix, at each time interval. We can observe that the gain values are dynamically chosen using the above mentioned method for a better performance of the quadrotor, unlike the traditional PID controller with constant gain values.

3) Q-learning tuned PID control for quadrotor in disturbed environment: The Q-learning agent implemented to auto-tune the PID controller gains was developed for the quadrotor to be able to function optimally in an environment with external disturbances. The end results obtained from the performance of the learned model on the controller gains are as shown in the TABLE III.

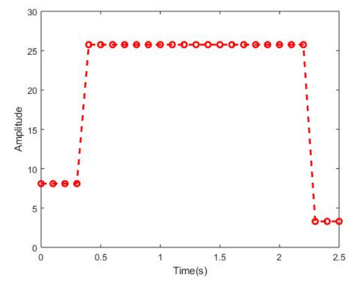
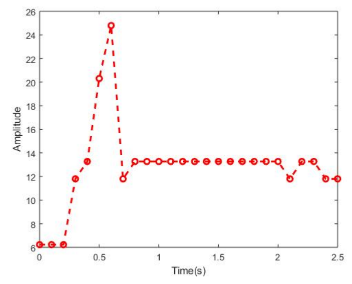
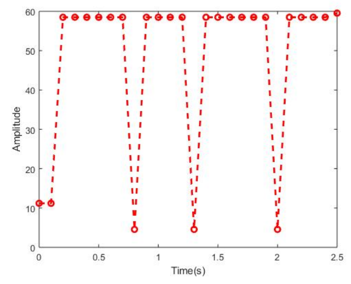
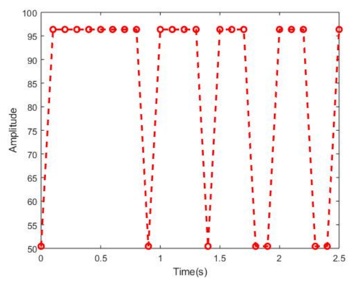
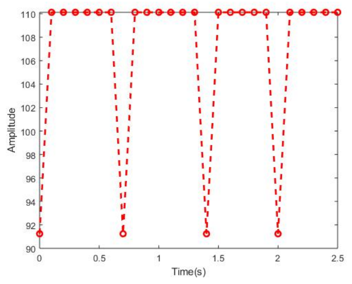
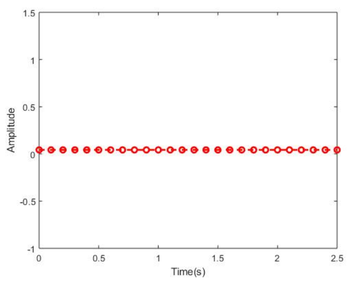
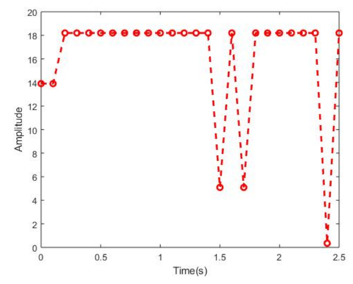
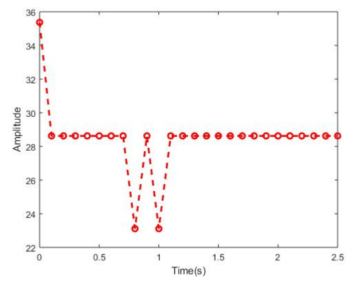
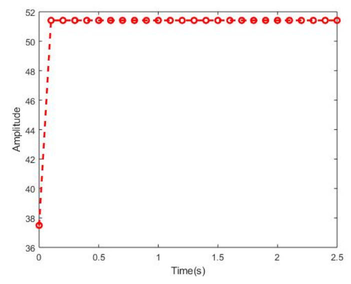
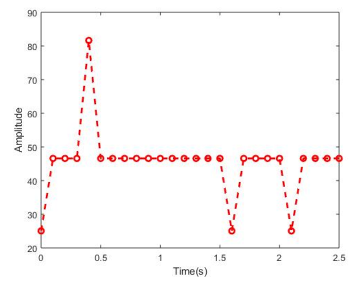
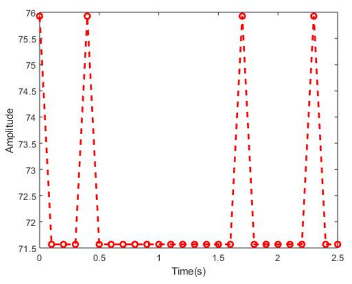
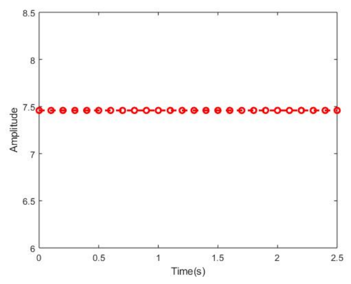
k_{px} 	k_{py} 	k_{pz} 
$k_{p\phi}$ 	$k_{p\theta}$ 	$k_{p\psi}$ 
k_{dx} 	k_{dy} 	k_{dz} 
$k_{d\phi}$ 	$k_{d\theta}$ 	$k_{d\psi}$ 

TABLE III
STATE VALUE OBTAINED USING Q LEARNING TUNED PID CONTROL

The above plots show the respective k_p and k_d values for the disturbed environment condition are chosen at each time interval during the simulation for the same. These k_p and k_d values are the chosen with the indices with maximum Q value from respective Q matrix obtained during the training with abnormal conditions of the quadcopter, at each time interval. We can observe that the gain values are dynamically chosen using the above mentioned method for a better performance of the quadrotor, unlike the traditional PID controller with constant gain values.

F. Conclusion

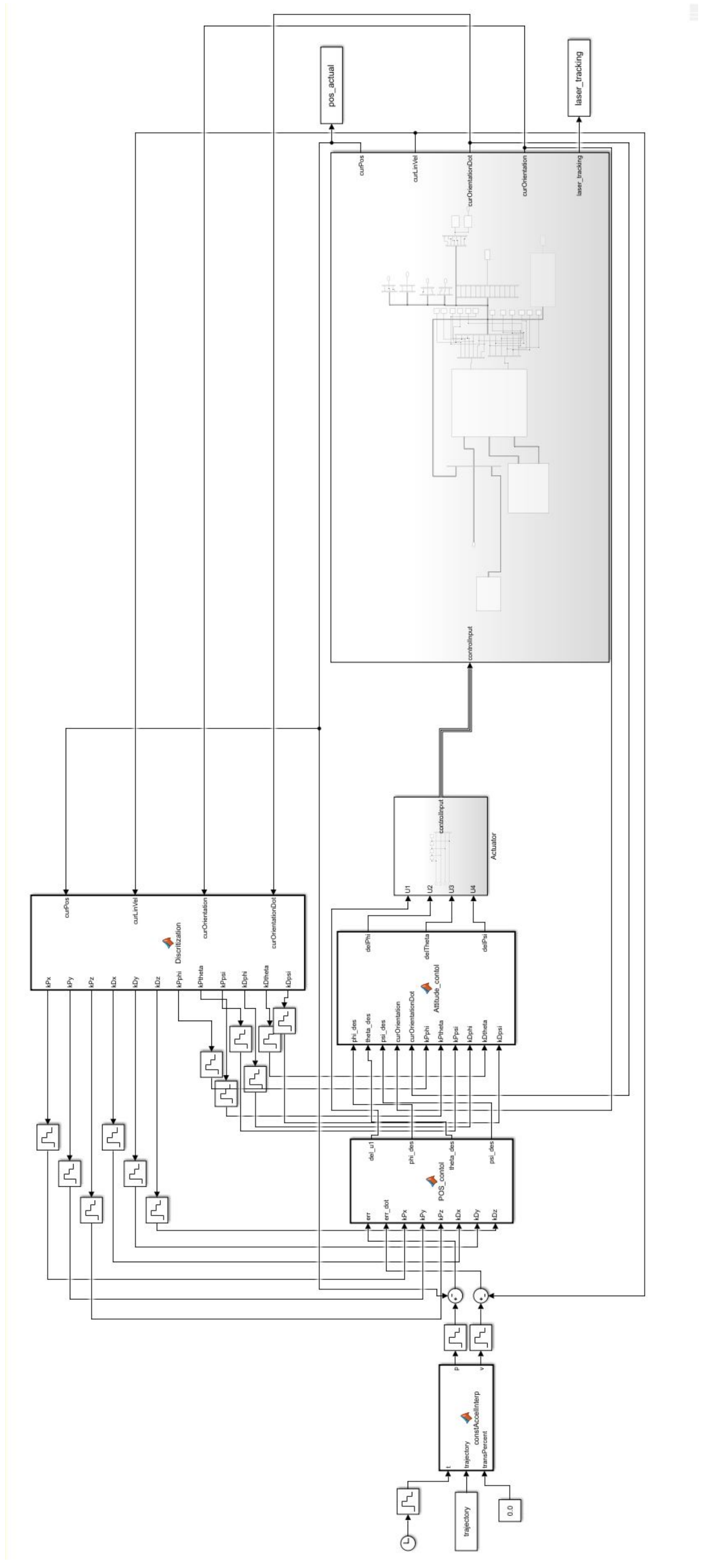
The objective of the project done was to tune the PID parameters using Q-learning for a quadrotor system. This auto-tuning of the PID parameters, using the RL technique Q-learning, was effective in traversing the quadrotor from the initial state to the goal state. The effectiveness of the attempt in involving reinforcement learning for the intended process was justified by the ability incorporated into the quadrotor, which is to maintain the trajectory even when perturbed with disturbances enroute. The conventional PID control lacks the ability to self-tune the PID parameters in a dynamic environment and hence would not be able to tackle the situation deliberated in the problem statement. The results also prove the above statement that the PID control used for controlling the quadrotor would not be sufficient, unless tuned manually online, during the travel if the quadrotor system encounters non-ideal state inputs. From this stage of the project, the future additions that could be made include the ability of completing the intended task of reaching the goal state even if physical failures are encountered by the system during the travel. Another improvement that could be done for the suggested system is to quicken the learning of the system as a whole. Better performance can be obtained by increasing the number of episodes and decreasing the time steps during the Q-learning training process.

References

[1] Shi, Qian, et al. "Adaptive PID controller based onQ-learning algorithm." CAAI Transactions on Intelligence Technology 3.4 (2018): 235-244.

[2] Kurak, Sevkuthan, and Migdat Hodzic. "Control and Estimation of a Quadcopter Dynamical Model." Periodicals of Engineering and Natural Sciences (PEN) 6.1 (2018): 63-75.

APPENDIX A
Q LEARNING TUNED PID CONTROL SIMULINK MODEL


pos_actual
laser_tracking
curPos
curLinVel
curOrientationDot
curOrientation
laser_tracking
controlInput
controlInput
Actuator
U1
U2
U3
U4
curPos
curLinVel
curOrientation
curOrientationDot
Discritization
kPx
kPy
kPz
kDx
kDy
kDz
kPphi
kPtheta
kPpsi
kDphi
kDtheta
kDpsi
delPhi
delTheta
delPsi
Attitude_contol
phi_des
theta_des
psi_des
curOrientation
curOrientationDot
kPphi
kPtheta
kPpsi
kDphi
kDtheta
kDpsi
del_u1
phi_des
theta_des
psi_des
POS_contol
err
err_dot
kPx
kPy
kPz
kDx
kDy
kDz
p
v
constAccelInterp
t
trajectory
transPercent
trajectory
0.0