**REINFORCEMENT LEARNING BASED LINEAR QUADRATIC REGULATOR FOR THE CONTROL OF A QUADCOPTER**

**By Vishal Kashyap**

**DENM100 Extended Research Project – Final Report**

**September 2022**

**Supervisor: Dr Ranjan Vepa**

**DECLARATION**

**DENM100 Extended Research Project – Final Report**

This report entitled **REINFORCEMENT LEARNING BASED LINEAR QUADRATIC REGULATOR FOR THE CONTROL OF A QUADCOPTER** was composed by me and is based on my own work. Where the work of others has been used, it is fully acknowledged in the text and in captions to tables and illustrations.

This report has not been submitted for any other qualification.

Name Vishal Kashyap

Signed ………………..

Date ………………..

**ABSTRACT**

In a practical implementation of the Linear Quadratic Regulator (LQR) for the control of a quadrotor drone, a key problem is the choice of the state and control weighting matrices, Q and R, respectively. In this paper, we propose a Reinforcement Learning based LQR for quadrotor control. Leveraging the advances in deep reinforcement learning, Deep Deterministic Policy Gradient (DDPG) model is used to reset the elements of the Q matrix to achieve a faster response while minimizing the integral square error (ISE). Following the properties of the LQR control law, the LQR-DDPG controller is optimal and asymptotically stable. The proposed controller is compared with four other extensively used methods to choose the Q matrix. In the first method, that Q matrix is initially selected to be an identity matrix. Bryson’s rule is used to set the Q matrix in the second method but not updated subsequently. Similar to the second method, the third method uses Bryson’s rule to set the Q matrix but uses a proportional derivative controller along with LQR. The third method uses an iterative optimization algorithm that minimizes the integral square error (ISE) over the training trajectories to select the Q matrix. The simulation results show that LQR-DDPG is better than all benchmarking cases in terms of rise time, settling time and time of flight, all by a margin of 10% or more.

**ACKNOWLEDGEMENTS**

I am extremely grateful to my esteemed supervisor, Dr Ranjan Vepa, for his invaluable advice, continuous support and endless patience throughout my dissertation. His experience and immense knowledge have been a huge motivating factor in all the time of my academic research and daily life. I would also like to thank Dr Adrian Briggs and Dr Yousef Issa Zawahreh for the valuable support sessions organised. Finally, a special thanks to my family, without their support completing my research would have been impossible for me.

Table of Contents

[1. Introduction 7](#_Toc113751290)

[2. Literature survey 8](#_Toc113751291)

[2.1. Choosing weighting matrices in LQR 8](#_Toc113751292)

[2.2. Reinforcement Learning (RL) for continuous action space 9](#_Toc113751293)

[3. Theory 11](#_Toc113751294)

[3.1. Quadrotor dynamics 11](#_Toc113751295)

[3.1.1. Dynamic Equations of Motion 12](#_Toc113751296)

[3.1.2. Linearization 14](#_Toc113751297)

[3.1.3. State space representation 15](#_Toc113751298)

[3.2. Linear Quadratic Regulator 16](#_Toc113751299)

[3.3. PID control 16](#_Toc113751300)

[3.3.1. Finding PID gains using ISE optimization 17](#_Toc113751301)

[3.4. Levenberg-Marquardt algorithm 17](#_Toc113751302)

[3.5. Reinforcement Learning 18](#_Toc113751303)

[3.5.1. Off-policy vs On-policy 19](#_Toc113751304)

[3.5.2. Learning the Q function 20](#_Toc113751305)

[3.5.3. Learning the policy function 20](#_Toc113751306)

[3.5.4. Actor-Critic Algorithms 20](#_Toc113751307)

[3.5.5. Deep Deterministic Policy Gradient (DDPG) 21](#_Toc113751308)

[4. Methodology 21](#_Toc113751309)

[4.1. LQR with Identity Q matrix 22](#_Toc113751310)

[4.2. LQR with Bryson’s rule to tune Q matrix 23](#_Toc113751311)

[4.3. LQR with Bryson’s rule to tune Q matrix and additional PD controller 23](#_Toc113751312)

[4.4. Levenberg-Marquardt algorithm to tune Q matrix 23](#_Toc113751313)

[4.5. LQR with RL agent to tune Q matrix 24](#_Toc113751314)

[5. Numerical Experiments and Discussion 26](#_Toc113751315)

[5.1. Test 1: Reaching a set point. 26](#_Toc113751316)

[5.1.1. Overview of training process. 27](#_Toc113751317)

[5.1.2. Results 27](#_Toc113751318)

[5.2. Test 2: Following a given trajectory 31](#_Toc113751319)

[5.2.1. Overview of training process 31](#_Toc113751320)

[5.2.2. Results 32](#_Toc113751321)

[5.3. Practical feasibility of controllers 35](#_Toc113751322)

[6. Conclusion 35](#_Toc113751323)

[References 37](#_Toc113751324)

**LIST OF SYMBOLS**

|  |  |
| --- | --- |
| **Symbol** | **Meaning/Description** |
|  | Action-value function in Reinforcement Learning |
|  | State of control system |
|  | Action given by RL agent |
|  | Probability function |
|  | Parameters of policy function in RL |
|  | Policy function in RL |
|  | Cosine rotation matrix |
|  | Transfer matrix |
|  | Mass of quadrotor |
|  | Identity matrix |
|  | Linear velocity of quadrotor |
|  | Rotational velocity of quadrotor w.r.t body frame |
|  | External force acting on quadrotor |
|  | External torque acting on quadrotor |
|  | Rotation speed of rotors |
|  | Thrust force |
|  | roll torque |
|  | pitch torque |
|  | yaw torque |
|  | Lift factor |
|  | Distance between the rotation axis of two opposite propellers |
|  | Drag factor |
|  | Gyroscopic torque |
|  | Moment of inertia of propellor |
|  | Angular rates w.r.t body frame |
| and | Principal moments of inertia |
|  | Gravitational acceleration constant |
|  | Proportional gain value in PID controller |
|  | Derivative gain value in PID controller |
|  | Integral gain value in PID controller |
|  | Gradient of vector |
|  | Transition function in RL |
|  | Discount factor in RL |
|  | Reward function in RL |
|  | Expectation |

# Introduction

The Linear Quadratic Regulator (LQR) is a well-known linear feedback controller that provides optimal feedback gains to ensure system stability and high performance (Kálmán, 1960). The performance of a system is defined using a cost function which is a function of control inputs and deviations from desired state. LQR algorithm has been successfully applied to control a quadrotor and has resulted in better performance compared to PID controller (Zulu and John, 2014). Cowling *et al.*, (2007) applied LQR controller on a quadrotor to achieve accurate path following in the presence of wind and other disturbances.

One of the main challenges in implementing LQR is to choose the cost function parameters. This is usually done by trial and error to achieve the best response, which is tedious and not optimal. To be useful, drones need to be quick. Because of the limited battery life, drones need to accomplish the assigned task in minimum time. Applications like search-rescue and medical equipment delivery (figure 1) have an inherent urgency which require a quick and stable response.

A drone flying over a mailbox

Description automatically generated with low confidence

Figure 1. Medical equipment delivery using drones (Snouffer, 2022)

Several methods have been tested to tune the weighting matrices in the LQR cost function to achieve faster response for various applications. However, RL has not been explored for tuning the weighting matrices in LQR. Compared to other optimisation-based and heuristic methods for setting the weighting matrices, RL tunes the weighting matrices based on the current state of the quadrotor. This should allow the controller to focus on certain states or inputs at different points in the trajectory. Compared to end-to-end RL algorithms for drone control (Song *et al.*, 2021), using an LQR-RL controller is optimal and asymptotical stable.

The aim of this research was to design an RL algorithm to specify the cost function parameters based on the state of the quadrotor without human intervention to achieve a faster and stable response. In this research, a simulation environment for a quadrotor was created in MATLAB. To control the quadrotor in 3 dimensions, an LQR controller with the quadrotor’s linearised dynamic equations of motion was employed. Using this simulation on MATLAB, five LQR-based controllers with different methods for choosing the Q matrix were compared in terms of their rise time, settling time, ISE and flight time. This comparison was carried out for two different cases, setpoint tracking and trajectory tracking. Out of the five methods for choosing the weighting matrix, three use heuristics, one is optimisation based, and one employs RL.

The rest of the paper is structured as follows. In section 2, we present a literature survey of methods used to set the LQR weighting matrices. This is followed by a brief survey of state-of-the-art RL algorithms that could be used to tune the weighting matrices. In section 3, we discuss the theory behind all controllers tested in this research. Section 4 describes the working of all five controllers in greater detail. Next, in section 5 we present and discuss the numerical experiments carried out to test the controllers' ability to reach a setpoint and follow a trajectory. Finally, the article is concluded in section 6.

Note: All code used in this research can be found in the following GitHub repository:

# Literature survey

Linear Quadratic regulator (LQR) lies at the centre of optimal control theory and has been investigated since the advent of control theory. LQR has been successfully applied to a wide range of applications documented in the literature. Muhando *et al.* (2008) proposed an LQR-based control strategy for the operation of a wind turbine generator. Pang, Zheng and Luo (2011) used LQR for the control of ball and beam system and demonstrated its effectiveness using practical experiments. Kedjar and Al-Haddad (2009) used LQR with integral action to control a three-phase three-wire shunt active filter.

LQR has been extensively tested and proven to be an effective control method for aerial vehicles as well. (Liu, Pan and Chang, 2016) solved the trajectory tracking problem for the Quanser Qball-X4 quadrotor (figure 2) using LQR. Sasongko *et al.* (2011) applied an LQR controller to a micro air vehicle for surveillance applications. Oner *et al.* (2009) and Hajiyev and Vural (2013) proved the effectiveness of LQR in vertical flight models. In this research, LQR is used to control the quadrotor. In section 2.1, we will review the algorithms used in literature to tune the weighting matrices in LQR and the reason why RL could be a better approach. Section 2.2 will discuss the state-of-the-art RL algorithms suitable for tuning the weighting matrices.

Chart, radar chart

Description automatically generated

Figure 2. Quanser Qball-X4 quadrotor used for trajectory tracking (Liu, Pan and Chang, 2016)

# Choosing weighting matrices in LQR

One of the key challenges in all practical applications of LQR is the choice of Q and R weighting matrices. These matrices can be adjusted based on user preference to give more importance to specific states or to control the cost of control inputs. The methods found in the literature to set the weighting matrices can be classified into two categories, non-optimisation and optimisation-based. Optimisation-based methods define a cost function and minimize that to select the weighting matrices. Non-optimisation-based methods use simple heuristics. The most common non-optimisation-based method is to choose identity Q and R matrices or tune the weight matrices by trial and error. Bryson’s rule is another example of non-optimisation methods, which has shown better performance than identity Q and R matrices (Okyere *et al.*, 2019).

Optimisation-based methods have been used to set weighting matrices to enhance a particular aspect of the controller performance, like settling or rise time (Branch, 2011). Several methods have been investigated in the literature to choose the optimal weighting matrices. Bhushan and Chatterjee (2017) used Genetic algorithm (GA) to tune the weighting matrices and found it to perform better than Proportional-Integral (PI) and vanilla LQR controllers in terms of rise time, settling time, peak value and steady state values. GA has been used in several other studies and showed similar improvements (Marada, Matousek and Zuth, 2017; Ghoreishi and Nekoui, 2012). Kambushev *et al.* (2019) chose Integral square error (ISE) as the objective function and used a Quasi-Newtonian method (MATLAB’s fsolve) to solve the optimisation problem. They tested their method on a lateral motion model of F-16 and found it to give better results than Bryson’s rule. Several other optimisation algorithms have been used to select the weighting matrices for a variety of applications, including ant colony (Jacknoon and Abido, 2017) and artificial bee colony (Wang *et al.*, 2013) for controlling an inverted pendulum system, differential evolution (Deng *et al.*, 2017) for a 5-D linear system, particle swarm for the control of Helicopter system (Yu and Hsieh, 2019) and simulated annealing for a lane-keeping system (Sherif *et al.*, 2019).

All the above optimization algorithms result in better system response than their non-optimization counterparts. However, these optimization algorithms keep the weighting matrices fixed as the system’s state changes. This approach does not give the freedom to give more importance to certain states or control inputs for different system states. The proposed method uses an RL algorithm that tunes the weighting matrices based on the system’s state. We will benchmark the RL algorithm against four methods to select the weighting matrices, which are as follows:

1. Set Q and R weighting matrices to identity
2. Bryson’s rule.
3. LQR-Proportional Derivative controller with Bryson’s.
4. Minimise ISE to obtain weighting matrices using a quasi-Newtonian method similar to Kambushev *et al.* (2019).

# Reinforcement Learning (RL) for continuous action space

To set LQR’s weighting matrices using RL, the agent’s action space has to be continuous as we want to select the Q and R matrices from a continuous range of values. Classic RL algorithms like Q learning (Watkins and Dayan, 1992) fail in such settings. Q learning trains using the method of Temporal difference (TD) (Sutton, 1988), in which the agent updates the *value* (also known as Q value or action-value) of taking a particular action in a state based on the immediate reward received and the *value* of the state to which it is taken. The optimal policy is then found by taking the maximum of the Q function over all actions in a state (. There are a couple of major issues with Q learning. First, Q learning often overestimates the action values under certain conditions (van Hasselt, Guez and Silver, 2015). Second, finding the maximum value of the Q function to get the optimal policy in continuous action space is challenging as the chosen algorithm may get stuck in a local maximum and/or be computationally expensive.

Another challenge while modelling an RL agent to set LQR’s weighting matrices is the choice of appropriate function approximators for the Q function. We want to choose a function approximator that can use raw, high-dimensional sensory data to solve complex tasks. With the recent advances in deep learning (Krizhevsky, Sutskever and Hinton, 2017), significant strides have been made in reinforcement learning with algorithms like “Deep Q Network” (DQN) (Mnih *et al.*, 2015). DQN uses a neural network to approximate the Q function and is capable of playing video games like Atari using raw input from pixel data. DQN showed that non-linear function approximators like neural networks could be successfully used to learn the value function in a robust and stable manner.

However, like Q learning, DQN is also not applicable to problems with high dimensional continuous action spaces. DQN selects the action that maximises the action-value function, e.g. turn left or right in a video game). In control problems, we need to select the best action within a range of continuous values. One potential solution to this could be discretizing the action space, but this results in many problems, including the curse of dimensionality.

The policy gradient algorithm is one widely researched algorithm to predict actions in continuous space. Policies can be modelled as either stochastic or deterministic. Stochastic policies are approximated as a parametrised probabilistic function, , which selects an action given a state based on parameters . Using the data sampled by the interaction between agent and environment, the policy function is adjusted toward the gradient of higher cumulative rewards.

Deterministic policies select an action value based only on the state (), unlike stochastic policies, which output a probabilistic distribution over the action space. Consequently, in a stochastic policy, the policy gradient has to integrate over the action and state spaces. Whereas, in a deterministic policy the policy gradient integrates over only the state space. This means deterministic policies will require fewer samples to converge compared to stochastic policies.

Silver *et al.* (2014) proposed an RL algorithm, Deterministic policy gradient (DPG) that can learn policies in high dimensional, continuous action space. They used an actor-critic algorithm and modelled both value and policy functions. Prior to DPG, it was believed that a deterministic policy gradient did not exist (Peters, 2010). However, (Silver *et al.*, 2014) showed that the deterministic policy gradient is equal to the expected gradient of the value function. They found DPG to outperform stochastic policies in high-dimensional action spaces significantly.

Lillicrap *et al.* (2019) combined the learnings from DPG and DQN to propose “Deep Deterministic Policy Gradient” (DDPG), an actor-critic algorithm that uses neural networks for action-value and policy function approximation. Like DPG, DDPG uses the policy gradient to update the value and policy functions. The DDPG algorithm robustly solves challenging control problems such as the cartpole swing-up (Manrique Escobar, Pappalardo and Guida, 2020). It requires 20 times fewer samples than DQN. Due to its data-efficient nature and state-of-the-art performance across several control tasks, we used DDPG in this research to select the weighting matrices in LQR.

# Theory

This section will cover the theoretical background behind the control methods developed and tested for the quadrotor. We will begin by deriving the state space equation for the quadrotor dynamics in section 3.1. In section 3.2, we will look at the theory behind LQR. Section 3.3 will describe a PID controller and how the optimal PID gains can be found using ISE optimization. Finally, section 3.4 discusses the Deep Deterministic Policy Gradient algorithm (DDPG) used for designing the RL agent that chooses the Q matrix in LQR based on ISE rewards.

# Quadrotor dynamics

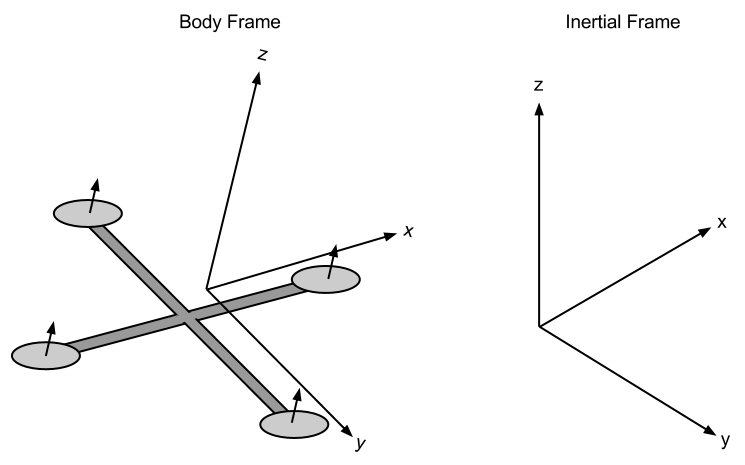


Figure 3. Plus configuration quadrotor with body and inertial frames

A quadrotor should be able to execute a range of motion with four rotors attached symmetrically to its four arms, as shown in figure 3. The quadrotor system is underactuated as it has to achieve six degrees of freedom with four inputs. This can be achieved with either plus or cross configurations. Comparing the two configurations, the cross configuration decouples the yaw moment from pitch/roll control inputs, which is not the case for the plus configuration (Partovi *et al.*, 2012). However, there are no significant differences in the flight dynamic characteristics of the two configurations (Niemiec and Gandhi, 2017). A plus configuration is used for all experiments presented in this paper (figure 3). For plus configuration, sideways motion is achieved by rotating left (right) rotors faster than right (left) rotors. This can be seen in figures 4.d and 4.h, where the arrow's thickness represents the rotation speed. Vertical motion is achieved by rotating all four rotors. Similarly, yaw motion (turning left or right) is also achieved by varying rotation speed on individual rotors (Cheon *et al.*, 2018).

Shape

Description automatically generated

|  |  |  |  |
| --- | --- | --- | --- |
| (a) | Yaw (anticlockwise direction) | (e) | Backward motion ( ) |
| (b) | Yaw (clockwise direction) | (f) | Forward motion ( ) |
| (c) | Take-off or take-up | (g) | Land or take-down |
| (d) | Sideways motion (right) | (h) | Sideways motion (left) |

Figure 4. Execution of different motions in plus configuration.

# Dynamic Equations of Motion

In this section, we will derive the rotation and linear equations of motion for the quadrotor and put these equations in a structure that is easy to work with while modelling a controller. To simplify the derivation, the rotation equations of motion will be derived in the body fixed frame, while the linear equations will be formulated in the inertial frame (see figure 3). The origin of the body fixed frame is at the centre of mass of the quadrotor, and the axes of the body fixed frame are parallel to the principle axes of inertia of the quadrotor. We can use the cosine matrix to transform the linear velocity vector from the body to the inertial frame.

Here, and are substitutes for and . Similarly, the transfer matrix ) can be used to convert rotational velocity from the body frame to the inertial frame.

Using this representation, the dynamics of the quadrotor can be written as

Where and are the linear and rotation velocities of the quadrotor w.r.t to the inertial and body frame, respectively. is the mass of the quadrotor. The external forces acting on the quadrotor in the inertial frame are represented by , and gives the external torque in the body frame. is the identity matrix, and is the mass of the quadrotor.

The inputs to the quadrotor will be the rotation speed of the four quadrotors, . We can represent these inputs in terms of the thrust force , roll torque , pitch torque , and yaw torque .

Due to this direct conversion between and , could be considered to be the input to the quadrotor. Using the inputs defined above, we can write as the combination of thrust and gravity forces,

The external torque is the sum of input torques and gyroscopic torque ( defined below.

Where is the moment of inertia of a propeller. To sum up, we can write the equations of motion of the quadrotor using equations 1-6 in the following form.

Where and are the principle moments of inertia of the quadrotor We define as and and are the angular rates w.r.t the body frame of the quadrotor.

# Linearization

The dynamic equations of motion for the quadrotor (equation 7) are highly non-linear. In order to control the quadrotor using optimal controllers like LQR, we need to linearize these equations. Different linearization techniques have been proposed and tested in the literature (Belkheiri *et al.*, 2012). A widely used method is to linearize the equations of motion (equation 7) at the hover state of the quadrotor. At the hover state, roll and pitch angles are small, and all linear and rotational velocities are zero (Hua *et al.*, 2013). These assumptions can be mathematically represented as

The yaw angle is taken as zero. With this assumption, the roll angle will be responsible for motion in the direction, and the pitch angle will cause the motion in the direction.

Using the above assumptions, we write the equations of motion (equation 8) in the following form.

# State space representation

The state space representation consists of input, output, and state variables related by a first-order differential equation. A "state space" can be described as a space having the state variables as the axes. Vectors can be used to represent the system's state in that space. For the quadrotor system, we can take the state vector and output to be the following (Araar and Aouf, 2014)

The state space equation then becomes

Where

# Linear Quadratic Regulator

Quadrotors are **dynamically unstable** due to changing parameters and disturbances such as wind (Zhang *et al.*, 2014). A suitable controller is required to make the quadrotor stable. The following equations govern finite horizon LQR,

Equation 13 is the state space equation of the quadrotor from previous section, and is the cost objective of the optimal controller. The first term of the cost function drives the state to the desired value weighted by the Q matrix. The second term limits the use of control inputs based on the weighting in the R matrix. To obtain a closed form solution, it is required that and are symmetric positive (semi-) definite matrices. The optimal control that minimizes the cost function is given by

Where is the feedback gain and is the solution of the Riccati equation

Under steady-state conditions, the above equation for *P* reduces to the algebraic Riccati equation, which is solved using MATLAB Eigenvalue decomposition after it is reduced to a pair of Hamiltonian equation. In steady state, the gain matrix *K* is a constant.

# PID control

The Proportional-integral-derivate controller is a widely used feedback control mechanism. This controller uses the difference between the current and desired state (the error) to drive the system to a zero error state. Three parameters that need to be tuned in this controller are the proportional (), derivative () and integral () gain values. Figure 5 shows the block diagram of the drone system containing a PID controller.

Diagram

Description automatically generated

Figure 5 Block diagram of PID control system for quadrotor

The error is calculated as the difference between the output and the input . This error is sent as the input to the PID control block, which calculates the integral and derivate of the error signal and weights them based on the gains defined above. The output of the PID block is sent as input to the drone system. The generalised transfer function of this system is given a

# Finding PID gains using ISE optimization

The performance index employed by Turkoglu *et al.* (2008) for longitudinal UAV flight control was used to find the PID gain values.

Minimizing the above function will result in a fast and stable response. However, the error function in equation 17 is in the s domain, whereas is in the time domain. To convert from time to s domain, we will use Parseval’s theorem (Kealy and O’Dwyer, 2003).

Where n is the degree of system dynamics and is the error signal transfer function (equation 17). We can find the optimal gain values using precalculated integral for equation 19 (Gradshteĭn, Ryzhik and Jeffrey, 2000) and minimizing the obtained expression.

# Levenberg-Marquardt algorithm

The task of selecting the optimum weighting matrices in LQR could be posed as an optimization problem with ISE as the objective function. This cost function can be written as

Where is the state of the control system, is the desired state and is the L2 norm. The state is a function of the Q matrix, and the objective is to choose the Q matrix such that the ISE is minimum.

Levenberg-Marquardt (LM) algorithm (Marquardt, 1963; Levenberg, 1944) can be used to find the minimum of this cost function. LM algorithm is a blend of gradient descent and Newton-Gauss algorithms. The gradient descent algorithm updates the parameters in each iteration using the following expression

Where is the gradient of w.r.t to the Q matrix. As such, the gradient descent algorithm suffers from convergence problems due to first-order approximation of the cost function (Mandic, 2004). The situation could be improved by using Newton’s method to minimize . Newton’s method uses the second order approximation of using the Taylor series expansion and gives the following update rule

Levenberg proposed an algorithm combining gradient descent and Newton’s method,

This update rule has advantages of both gradient descent and Newton’s method. The factor is chosen based on the performance of the update rule. If the function decreases after an update, our quadratic assumption on is working, and we decrease lambda. On the other hand, if the error goes up, we increase to increase the effect of gradient descent.

# Reinforcement Learning

**Diagram

Description automatically generated**

Figure 6. Agent-environment interaction in Reinforcement Learning

The study of RL is concerned with the way agents learn. Agents are components responsible for taking actions based on the state of the environment it interacts with. The environment, in turn, rewards the agent for taking actions that help it reach a goal state (see figure 6). Through this experience, the agent can determine the optimal action suited to every state of the environment to maximise the expected cumulative reward in the future. This set of learned (optimal) actions for all states in the environment is known as an (optimal) policy.

To mathematically model the interaction between agent and environment, Markov decision processes (MDPs) are used extensively (Kober, Bagnell and Peters, 2013). MDPs are stochastic processes that satisfy the Markov property. An environment is said to satisfy the Markov property if the environment’s response to taking action in a particular state depends only on the current state of the environment (Sutton and Barto, 1998). An MDP process can be described as a tuple where is the set of states agents can occupy, and  is the set of actions the agent can take. is the transition function which satisfies the Markov property and gives the probability an agent will arrive at state from state after taking an action at time . is the reward the agent receives after making the transition. A sequence of such transitions is called a trajectory.

On following a trajectory, an agent collects rewards , where  is the discount factor that gives more weight to rewards accumulated in the near future. The total reward collected by an agent starting at a state-action pair and following a policy is known as the Q value. The function that gives the Q value for all state and action pairs is known as a Q function.

Equation 21 gives the Q value of the agent, starting at state , taking action and following a policy after that. The objective of the RL agent is to learn the optimal policy

In the following sub-sections (3.5.1 to 3.5.5), we will describe the main features and give an overview of the training steps of the DDPG algorithm.

# Off-policy vs On-policy

In an RL algorithm, the policy used by the agent to explore the environment (known as behaviour policy) can be different from the policy the agent is learning. Such algorithms are classified as off-policy. Algorithms in which behaviour and learned policy are the same are known as on-policy algorithms. A key advantage of off-policy algorithms is that while updating Q value or policy functions using gradient ascent, the gradient can be calculated by sampling all previous experiences of the agent. This is known to give a better gradient estimate than using a single noisy transition (Lillicrap *et al.*, 2019).

# Learning the Q function

The Bellman equation is the starting point for approximating a Q function. Suppose the agent is at state , and after taking action it reaches state . Intuitively, the Bellman equation says that if the agent is following an optimal policy, the Q value at state after taking action ( should be equal to the sum of reward collected by the agent on reaching state and . This can be mathematically represented by the equation below

If we want to learn an approximator to the optimal Q function, say , we can set up a cost function L that tells us how closely the approximated function follows the Bellman equation.

# Learning the policy function

A policy function takes the state as input and gives the optimal action to maximise the expected cumulative reward. The policy function can be modelled as any function with parameters . (Silver *et al.*, 2014).

To learn a policy using the Q function approximated in the previous section, we simply need to find the action that will result in the maximum Q value in each state.

# Actor-Critic Algorithms

**Diagram

Description automatically generated**

Figure 7. Actor-critic RL agent architecture

An actor-critic algorithm combines the two types of RL algorithms, value-based and policy-based. The actor gives an action based on the state of the environment. The critic learns a Q value function (), given the action from the actor. The updates in both Q value and policy functions are driven by the output of the critic (Grondman *et al.*, 2012).

# Deep Deterministic Policy Gradient (DDPG)

The DDPG algorithm employs a model-free, off-policy actor-critic algorithm to model the agent (Lillicrap *et al.*, 2019). The actor and critic are both neural networks. The critic learns the Q function by minimizing Bellman equation error using the cost function defined in equation 24. Since the action space is continuous, it is not feasible to find the optimal action using equation 25. Instead, we learn a policy function which gives the optimal action that maximizes the Q function approximated by the critic.

# Methodology

Five methods were tested for controlling the quadrotor using the state space equation defined in equation 11 and the algorithms discussed in section 3 (see table 1). Each method was tested for its ability to reach the desired set point and follow a trajectory. This section will describe all five control.

Table 1. A list of controllers tested with type of algorithm used to select Q matrix (see section 2.1)

|  |  |  |
| --- | --- | --- |
| **Controller** | **Method of choosing Q matrix** | **Type of algorithm** |
| LQR | Identity matrix | Non-optimization based |
| LQR | Bryson's rule | Non-optimization based |
| LQR + PD | Bryson's rule | Non-optimization based |
| LQR | LM | Optimization based |
| LQR | DDPG | Reinforcement Learning |

# LQR with Identity Q matrix

In this controller, LQR, as described in section 3.2, is used for feedback control. To generalise the controller for set points other than zero, we will use the error state, , instead of the state vector (equation 10). Where is the desired state vector. The state dynamics then follow from equation 13

Where . The optimal control (equation 15) for the error state is given by the following equation (Lewis, Vrabie and Syrmos, 2012)

Where is the feedback gain and is the input required by the quadrotor to stay at the desired state. The Q matrix for this controller will have the following form

Figure 8 shows the control block for this controller.

A screenshot of a computer

Description automatically generated with low confidence

Figure 8. Control block for LQR controller

# LQR with Bryson’s rule to tune Q matrix

This controller is similar to the LQR controller defined in section 4.1. The only difference between the two controllers is the way we set the Q matrix. Bryson’s rule will be used to set the diagonal elements of the Q matrix in this controller. Intuitively, Bryson’s rule scales each diagonal term in the Q matrix such that the maximum value it can take is one. This is achieved by dividing each diagonal value by the square of the expected maximum value its corresponding state can take. This is particularly useful if the units of elements in the state vector are different (Hespanha, 2009). We will apply Bryson’s rule only on state variables that we want to control: . This will allow other state variables to vary freely to reach desired .

# LQR with Bryson’s rule to tune Q matrix and additional PD controller

A PD control, in addition to the controller defined in section 4.2, is used in the third controller. The PD gains are found using the ISE optimization method defined in section 3.3.1. Due to the introduction of a PD control along with LQR, the optimal gain matrix K will change. After accounting for the input due to PD control, the error dynamics for LQR (equation 31) will have the following modified form

In the above equation, is the input from the PD controller and is defined as follows

Using the state vector notation, equation 4 can be written as

Where , is a matrix that contains the PD gain values,

# Levenberg-Marquardt algorithm to tune Q matrix

In this controller, ISE is chosen as the objective function, and MATLAB’s fsolve is used to set the four diagonal elements of the Q matrix (see section 4.2) for the LQR controller. After each training episode, ISE is calculated, and the Q matrix weights are adjusted using the Levenberg-Marquardt Algorithm (available as part of MATLAB’s fsolve function package) such that ISE is minimised (see section 3.4). The complete procedure is as follows:

1. LQR requires a positive semi-definite (PSD) Q matrix to allow the solution for the Ricatti equation (equation 16). If the Q matrix is PSD proceed to step 1. Else return a high negative cost function value.
2. Use the updated Q matrix with the LQR controller to reach the desired state and return ISE for the episode.
3. Update the Q matrix values using the Levenberg-Marquardt Algorithm and ISE from step 2. Return to step 1 with updated Q values and proceed in the loop till minimum is reached.

Following the above algorithm, Q matrix values were set to the following in this controller,

# LQR with RL agent to tune Q matrix

This section will introduce our implementation of the DDPG algorithm discussed in section 3.5.5. We will look at our choice of the state given as input to the RL agent, the output (action) of the actor-network, the reward function and the actor-critic neural network architecture.

**State:** The state of the drone will be represented using a 12-dimensional vector, . The state vector will have the following structure,

Where are the position coordinates and are the angular displacements. A single element from the state vector will be indexed as . For example, the velocity in the direction can be notated as . A waypoint is an intermediate state configuration the drone should reach while following a trajectory. The waypoint on a trajectory will be notated using . A notation similar to the state vector is followed to notate individual elements in the waypoint. For example, the position of a waypoint will be shown as .

The drone starts from an initial state, . In each iteration of the algorithm, the RL agent receives the drone's position relative to the next waypoint and current velocities. For example, if the drone has reached waypoint , the input containing state information to the RL agent () will be as follows:

Defining the input in this way allows us to facilitate learning by generalising the goal of the RL agent to a zero vector for all inputs received.

**Action:** The RL agent outputs the values of four diagonal elements in the Q matrix, : . This is similar to the controller in section 4.2.

**Reward function:** The reward function used to train the RL agent is composed of two components. If the drone is between waypoints, the RL agent is given a penalty equal to the ISE. This incentivises the agent to reach its goal (or subgoal) in the minimum number of steps. The agent is given a positive reward if it reaches a goal (final waypoint) or subgoal (intermediate waypoint).

**Actor-Critic NN architecture:** As mentioned in section 3.5.5, both the actor and the critic are neural networks whose weights are updated to learn the optimal policy (equation 30) and the Q function for the optimal policy (equation 27), respectively. Figure 9 shows the neural network architecture for the actor. A 1×24 dimension vector, which is a concatenation of the state vector (1×12) and the goal vector (the final desired state, 1×12) is given as input to the actor. The input is then scaled to bring it within the range [-1,1]. This is required because the input vector has values of different units and the learning is better after normalisation (Ioffe and Szegedy, 2015). The scaling layer is followed by four fully connected layers with an output dimension of four. Next, sigmoid and scaling layers are used to get the values of the four diagonal elements of the Q matrix. The final values are kept within the range of [0.01,100] to avoid having singular Q matrix.

Diagram

Description automatically generated

Figure 9. Actor Neural network design

The critic network takes two vectors as inputs, the state vector and the action vector, which is the output of the actor (see figure 10). For this state-action pair, the critic gives as output the expected Q value (equation 27). First, the state input is scaled to the range [-1,1] while the action input is scaled between [0,1]. Next, these inputs are concatenated into a single 1×16 vector and passed through four fully connected layers similar to the actor-network. Finally, the tanh layer and scaling layer output a value between -1000 and 1000.

Diagram

Description automatically generated

Figure 10: Critic Neural network design

# Numerical Experiments and Discussion

The learning agent’s ability to reach the desired set point and follow a trajectory was tested on MATLAB. This section discusses the results obtained for both the above cases. We begin by describing the notation and quantitative metrics used to compare different controllers.

Four quantitative metrics will be used to compare the performance of different controllers, rise time, settling time, integral square error (ISE) and time of flight. Rise time is the time the quadrotor takes to go from 10% to 90% of its goal position. Settling time is the time the quadrotor takes to reach and stay within 2% of its steady state position. Flight time is the total time the quadrotor takes to reach its goal position from the initial state. ISE is formed by integrating the square of the error (difference between desired and current state of the drone) over its total flight time.

# Test 1: Reaching a set point.

Experiments in this section were designed to answer the following questions:

1. Is the learning-based approach able to reach a random desired set point?
2. How does the learning-based approach compare with the methods listed in table 1?

# Overview of the training process.

In each training episode, the MATLAB RL environment described in section 4.5 generates a random set point for the quadrotor to reach. This new desired set point has a random value for x, y and z coordinates between the closed interval [-1m, 1m]. The desired angular displacements and velocities are set to zero. Overall, the desired state vector has the following structure:

The quadrotor starts from an initial state of zero position and velocity. In each iteration of the episode, the RL agent receives the current state of the quadrotor and outputs the four diagonal elements of the Q matrix, as described in section 4.5. With the updated Q matrix, the controller outputs the thrust for each rotor which drives the quadrotor’s state towards the desired state.

# Results

*Training*

The RL agent was trained for 100 episodes, and the reward per episode was recorded. It is worth noting that a lower reward does not necessarily represent a better trajectory, as each episode has a different desired state. The average reward till the current episode is a more robust measure to monitor the agent's learning. Figure 11 shows these two quantities after each episode.



Figure 11. Training progress of the RL agent to reach a desired set point

The RL agent collects better rewards as the training progresses. As the agent learns a meaningful Q function and policy, it is expected that the quadrotor should be able to reach the desired state more often and in less number of iterations. Figure 12 shows the number of iterations taken by the drone to reach the desired set point in a given episode. The number of iterations required reduces to lower values as the training progresses.



Figure 12. Number of iterations taken by RL agent to reach the desired state per episode

*Testing*

After training the agent for 100 episodes, it was tested on 50 new random set points. The agent successfully reached all the test set points within the allocated time of 10 seconds. Figure 13 shows the trajectory taken by different controllers for one of the test cases.



Figure 13. Sample trajectory comparing the four controllers on a test set point

The five controllers were compared for their time of flight, ISE, rise time and settling time averaged across the 50 test cases (Table 2). All controllers perform better than the vanilla LQR (identity Q matrix). LQR combined with RL agent to choose the Q matrix gives the lowest average flight time, ISE, rise time, and settling time. Using LQR with PD controller gives significant improvement over using just LQR controller. The ISE values for both the optimization-based methods, LM and RL DDPG, are the lowest.

Table 2. Comparison of controllers for reaching a set point

|  |  |  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- | --- | --- |
| **Controller** | **Method of choosing  Q matrix** | **Average Time of flight  (seconds)** | **Average Integral Square Error** | | | | **Average Rise  time (seconds)** | **Average  settling time (seconds)** |
|  |  |  | **x** | **y** | **z** | **psi** |  |  |
| LQR | Identity matrix | 9.83 | 0.37 | 0.35 | 0.38 | 2.44E-07 | 2.45 | 4.38 |
| LQR | Bryson's rule | 6.11 | 0.36 | 0.33 | 0.35 | 8.36E-07 | 1.21 | 2.00 |
| LQR + PD | Bryson's rule | 5.20 | 0.27 | 0.27 | 0.23 | 2.00E-06 | 1.41 | 2.25 |
| LQR | LM | 5.41 | 0.19 | 0.22 | 0.16 | 6.00E-06 | 0.90 | 2.33 |
| LQR | RL DDPG | 4.64 | 0.15 | 0.14 | 0.11 | 4.08E-05 | 0.52 | 1.71 |

# Test 2: Following a given trajectory

There are two main differences between the tasks of reaching a set point and following a trajectory. First, after reaching a waypoint, the drone starts moving towards the next waypoint with non-zero linear and angular velocities. Second, the drone does not have to stop at intermediate waypoints and must only come to a halt at the final waypoint. These differences make the problem of following a trajectory worth exploring. The experiments in this section aim to answer the following questions:

1. Is the learning-based approach able to follow the desired trajectory?
2. How does the performance of the learning-based approach compare with methods listed in table 1?

# Overview of the training process

At the beginning of each episode, a random trajectory with ten waypoints is generated. Each waypoint is sampled using the equations defined below:

Where and are two consecutive waypoints and and are sampled from a uniform distribution with -1 and 1 as the lower and upper bound, respectively.

The drone starts from an initial state, . In each iteration of the algorithm, the RL agent receives the drone's position relative to the next waypoint and current velocities. The actor outputs the values of the Q matrix as actions using the state of the drone provided as input (see section 4.5). These values are then fed into the LQR, which gives control inputs for the drone.

# Results

*Training*

With the architecture and reward function described in section 4.5 and the training process described above, the RL agent was trained for 100 episodes. The figure below shows the reward obtained per episode and the average reward obtained till the current episode.

Figure 14. Training progress of the RL agent to follow a trajectory

From the above figure, it can be seen that the agent earns progressively better rewards. As the agent learns, it is able to complete the trajectory more often in the given time limit. During the first few episodes, the agent explores (this is achieved by adding decaying noise to the actor’s output), and after around 15 episodes, it exploits the learned policy and the reward obtained per episode plateaus. This can also be seen in figure 15, where the number of iterations taken to complete an episode reaches a stable lower value after 15 episodes.



Figure 15. Number of iterations taken by RL agent to complete the trajectory per episode

*Testing*

The RL agent’s ability to follow a trajectory was tested on fifty new random trajectories that were not seen during training. This performance was compared with the four benchmark approaches for selecting the Q matrix in LQR. Figure 16 shows the performance of all four methods for one such test trajectory. The RL agent takes a significantly lower time compared to the other controllers.



Figure 16. Sample trajectory comparing the four controllers on a test trajectory

We calculated the average time of flight and ISE for all four controllers over the fifty test trajectories (table 2). Like test 1, the RL agent has the lowest flight time and ISE. The RL agent is 28.5% better than the second best method (LM). Bryson’s rule also gives significantly better results than vanilla LQR, which uses an identity matrix. Unlike test 1, the LQR controller gives a lower flight time when compared with the LQR + PD controller.

Table. 3 Comparison of controllers for following a trajectory

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **Controller** | **Method of**  **choosing Q matrix** | **Average Time of flight (seconds)** | **Average Integral Square Error** | | | |
|  |  |  | **x** | **y** | **z** | **psi** |
| LQR | Identity matrix | 51.64 | 3.60 | 3.06 | 3.36 | 0.000002 |
| LQR | Bryson's rule | 26.98 | 3.40 | 2.73 | 2.93 | 0.000007 |
| LQR + PD | Bryson's rule | 28.67 | 2.64 | 2.31 | 2.06 | 0.000010 |
| LQR | LM | 24.31 | 1.86 | 1.82 | 1.42 | 0.002300 |
| LQR | RL | 18.92 | 1.41 | 1.16 | 1.02 | 0.000340 |

# Practical feasibility of controllers

There are practical limits to the values of the true inputs and the rotation speed of rotors (), which must be considered in a real design. Using equation 4, which provides the conversion between and , we calculated the maximum rotation speed required for each rotor across all fifty test trajectories described in section 5.2.1 (table 4).

Table 4. Maximum inputs required by controllers across fifty test trajectories

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
|  | **LQR** | **BLQR** | **BLQRPD** | **LM** | **RL** |
| Ω1 (rpm) | 131.21 | 139.29 | 192.96 | 206.63 | 289.61 |
| Ω2 (rpm) | 162.76 | 140.79 | 248.93 | 251.24 | 420.15 |
| Ω3 (rpm) | 163.68 | 141.23 | 236.26 | 292.89 | 423.84 |
| Ω4 (rpm) | 124.10 | 149.24 | 165.17 | 161.81 | 256.32 |

It can be observed from the above table that a lower flight time usually requires a higher rotation speed (more energy). This is true for all controllers except for LQR with Bryson's rule (BLQR). BLQR and LQR have similar rotation speeds, but BLQR has half the fight time.

# Conclusion

In this study, a reinforcement learning-based LQR controller is proposed and designed to control a quadrotor in three dimensions. The dynamic equations of motion of the quadrotor are linearized at hover conditions to derive the state-space model used to design the controller. The actor-critic RL algorithm, DDPG, is used to select the Q matrix in LQR optimally. The proposed controller is compared against four other methods used extensively in the literature to select the Q matrix in LQR, three heuristics based and one optimization based. The DDPG algorithm achieves 28.4% lower flight time while following a trajectory than all other methods. During setpoint tracking, it also achieves 73.1% lower rise time and 17% lower settling time. However, this increase in performance requires a higher rotor rotation speed than the vanilla LQR controller. Bryson’s rule to select the Q matrix gives the best performance if one wants to keep the maximum rotor rotation speed similar to vanilla LQR.

# References

Araar, O. and Aouf, N. (2014) ‘Full linear control of a quadrotor UAV, LQ vs H&#x221E’;, in *2014 UKACC International Conference on Control (CONTROL)*. *2014 UKACC International Conference on Control (CONTROL)*, Loughborough, UK: IEEE, pp. 133–138. Available at: https://doi.org/10.1109/CONTROL.2014.6915128.

Belkheiri, M. *et al.* (2012) ‘Different linearization control techniques for a quadrotor system’, in *CCCA12*. *2012 2nd International Conference on Communications, Computing and Control Applications (CCCA)*, Marseilles, France: IEEE, pp. 1–6. Available at: https://doi.org/10.1109/CCCA.2012.6417914.

Bhushan, R. and Chatterjee, K. (2017) ‘Mathematical modeling and control of DFIG-based wind energy system by using optimized linear quadratic regulator weight matrices’, *International Transactions on Electrical Energy Systems*, 27(11), p. e2416. Available at: https://doi.org/10.1002/etep.2416.

Branch, S.T. (2011) ‘Optimal design of LQR weighting matrices based on intelligent optimization methods’, *International Journal of Intelligent Information Processing*, 2(1), pp. 63–74.

Cheon, J.H. *et al.* (2018) ‘Toward a Secure Drone System: Flying With Real-Time Homomorphic Authenticated Encryption’, *IEEE Access*, 6, pp. 24325–24339. Available at: https://doi.org/10.1109/ACCESS.2018.2819189.

Cowling, I.D. *et al.* (2007) ‘A prototype of an autonomous controller for a quadrotor UAV’, in *2007 European Control Conference (ECC)*. *European Control Conference 2007 (ECC)*, Kos: IEEE, pp. 4001–4008. Available at: https://doi.org/10.23919/ECC.2007.7068316.

Deng, X. *et al.* (2017) ‘Optimal analysis of the weighted matrices in LQR based on the differential evolution algorithm’, in *2017 29th Chinese Control And Decision Conference (CCDC)*. *2017 29th Chinese Control And Decision Conference (CCDC)*, Chongqing, China: IEEE, pp. 832–836. Available at: https://doi.org/10.1109/CCDC.2017.7978635.

Ghoreishi, S.A. and Nekoui, M.A. (2012) ‘Optimal Weighting Matrices Design for LQR Controller Based on Genetic Algorithm and PSO’, *Advanced Materials Research*, 433–440, pp. 7546–7553. Available at: https://doi.org/10.4028/www.scientific.net/AMR.433-440.7546.

Gradshteĭn, I.S., Ryzhik, I.M. and Jeffrey, A. (2000) *Table of integrals, series, and products*. 6th ed. San Diego: Academic Press.

Grondman, I. *et al.* (2012) ‘A Survey of Actor-Critic Reinforcement Learning: Standard and Natural Policy Gradients’, *IEEE Transactions on Systems, Man, and Cybernetics, Part C (Applications and Reviews)*, 42(6), pp. 1291–1307. Available at: https://doi.org/10.1109/TSMCC.2012.2218595.

Hajiyev, C. and Vural, S.Y. (2013) ‘LQR Controller with Kalman Estimator Applied to UAV Longitudinal Dynamics’, *Positioning*, 04(01), pp. 36–41. Available at: https://doi.org/10.4236/pos.2013.41005.

van Hasselt, H., Guez, A. and Silver, D. (2015) ‘Deep Reinforcement Learning with Double Q-learning’. Available at: https://doi.org/10.48550/ARXIV.1509.06461.

Hespanha, J.P. (2009) *Linear systems theory*. Princeton: Princeton University Press.

Hua, M.-D. *et al.* (2013) ‘Introduction to feedback control of underactuated VTOLvehicles: A review of basic control design ideas and principles’, *IEEE Control Systems*, 33(1), pp. 61–75. Available at: https://doi.org/10.1109/MCS.2012.2225931.

Ioffe, S. and Szegedy, C. (2015) ‘Batch Normalization: Accelerating Deep Network Training by Reducing Internal Covariate Shift’. arXiv. Available at: http://arxiv.org/abs/1502.03167 (Accessed: 7 September 2022).

Jacknoon, A. and Abido, M.A. (2017) ‘Ant Colony based LQR and PID tuned parameters for controlling Inverted Pendulum’, in *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*. *2017 International Conference on Communication, Control, Computing and Electronics Engineering (ICCCCEE)*, Khartoum, Sudan: IEEE, pp. 1–8. Available at: https://doi.org/10.1109/ICCCCEE.2017.7867652.

Kálmán, R.E. (1960) ‘Contributions to the Theory of Optimal Control’, in.

Kambushev, M. *et al.* (2019) ‘Optimizing the Selection of the Weighting Matrices Q and R in Linear Quadratic Regulator’, in *2019 II International Conference on High Technology for Sustainable Development (HiTech)*. *2019 II International Conference on High Technology for Sustainable Development (HiTech)*, Sofia, Bulgaria: IEEE, pp. 1–3. Available at: https://doi.org/10.1109/HiTech48507.2019.9128263.

Kealy, T. and O’Dwyer, A. (2003) ‘Analytical ISE calculation and optimum control system design’. Available at: https://doi.org/10.21427/J1HZ-4Q93.

Kedjar, B. and Al-Haddad, K. (2009) ‘DSP-Based Implementation of an LQR With Integral Action for a Three-Phase Three-Wire Shunt Active Power Filter’, *IEEE Transactions on Industrial Electronics*, 56(8), pp. 2821–2828. Available at: https://doi.org/10.1109/TIE.2008.2006027.

Kober, J., Bagnell, J.A. and Peters, J. (2013) ‘Reinforcement learning in robotics: A survey’, *The International Journal of Robotics Research*, 32(11), pp. 1238–1274. Available at: https://doi.org/10.1177/0278364913495721.

Krizhevsky, A., Sutskever, I. and Hinton, G.E. (2017) ‘ImageNet classification with deep convolutional neural networks’, *Communications of the ACM*, 60(6), pp. 84–90. Available at: https://doi.org/10.1145/3065386.

Levenberg, K. (1944) ‘A method for the solution of certain non-linear problems in least squares’, *Quarterly of Applied Mathematics*, 2(2), pp. 164–168. Available at: https://doi.org/10.1090/qam/10666.

Lewis, F.L., Vrabie, D.L. and Syrmos, V.L. (2012) *Optimal control*. 3rd ed. Hoboken: Wiley.

Lillicrap, T.P. *et al.* (2019) ‘Continuous control with deep reinforcement learning’. arXiv. Available at: http://arxiv.org/abs/1509.02971 (Accessed: 24 August 2022).

Liu, C., Pan, J. and Chang, Y. (2016) ‘PID and LQR trajectory tracking control for an unmanned quadrotor helicopter: Experimental studies’, in *2016 35th Chinese Control Conference (CCC)*. *2016 35th Chinese Control Conference (CCC)*, Chengdu, China: IEEE, pp. 10845–10850. Available at: https://doi.org/10.1109/ChiCC.2016.7555074.

Mandic, D.P. (2004) ‘A Generalized Normalized Gradient Descent Algorithm’, *IEEE Signal Processing Letters*, 11(2), pp. 115–118. Available at: https://doi.org/10.1109/LSP.2003.821649.

Manrique Escobar, C.A., Pappalardo, C.M. and Guida, D. (2020) ‘A Parametric Study of a Deep Reinforcement Learning Control System Applied to the Swing-Up Problem of the Cart-Pole’, *Applied Sciences*, 10(24), p. 9013. Available at: https://doi.org/10.3390/app10249013.

Marada, T., Matousek, R. and Zuth, D. (2017) ‘Design of Linear Quadratic Regulator (LQR) Based on Genetic Algorithm for Inverted Pendulum’, *MENDEL*, 23(1), pp. 149–156. Available at: https://doi.org/10.13164/mendel.2017.1.149.

Marquardt, D.W. (1963) ‘An Algorithm for Least-Squares Estimation of Nonlinear Parameters’, *Journal of the Society for Industrial and Applied Mathematics*, 11(2), pp. 431–441. Available at: https://doi.org/10.1137/0111030.

Mnih, V. *et al.* (2015) ‘Human-level control through deep reinforcement learning’, *Nature*, 518(7540), pp. 529–533. Available at: https://doi.org/10.1038/nature14236.

Muhando, E.B. *et al.* (2008) ‘Augmented LQG controller for enhancement of online dynamic performance for WTG system’, *Renewable Energy*, 33(8), pp. 1942–1952. Available at: https://doi.org/10.1016/j.renene.2007.12.001.

Niemiec, R. and Gandhi, F. (2017) ‘Multirotor Controls, Trim, and Autonomous Flight Dynamics of Plus- and Cross-Quadcopters’, *Journal of Aircraft*, 54(5), pp. 1910–1920. Available at: https://doi.org/10.2514/1.C034165.

Okyere, E. *et al.* (2019) ‘LQR controller design for quad‐rotor helicopters’, *The Journal of Engineering*, 2019(17), pp. 4003–4007. Available at: https://doi.org/10.1049/joe.2018.8126.

Oner, K.T. *et al.* (2009) ‘Lqr And Smc Stabilization Of A New Unmanned Aerial Vehicle’. Available at: https://doi.org/10.5281/ZENODO.1056268.

Pang, Z.-H., Zheng, G. and Luo, C.-X. (2011) ‘Augmented state estimation and LQR control for a ball and beam system’, in *2011 6th IEEE Conference on Industrial Electronics and Applications*. *2011 6th IEEE Conference on Industrial Electronics and Applications (ICIEA)*, Beijing, China: IEEE, pp. 1328–1332. Available at: https://doi.org/10.1109/ICIEA.2011.5975793.

Partovi, A.R. *et al.* (2012) ‘Development of a Cross Style Quadrotor’, in *AIAA Guidance, Navigation, and Control Conference*. *AIAA Guidance, Navigation, and Control Conference*, Minneapolis, Minnesota: American Institute of Aeronautics and Astronautics. Available at: https://doi.org/10.2514/6.2012-4780.

Peters, J. (2010) ‘Policy gradient methods’, *Scholarpedia*, 5(11), p. 3698. Available at: https://doi.org/10.4249/scholarpedia.3698.

Sasongko, R.A. *et al.* (2011) ‘Path following system of small unmanned autonomous vehicle for surveillance application’, in *2011 8th Asian Control Conference (ASCC)*, pp. 1259–1264.

Sherif, M.M. *et al.* (2019) ‘Optimal Control of Lane Keeping System Using Simulated Annealing and Linear Quadratic Regulator’, in *2019 15th International Computer Engineering Conference (ICENCO)*. *2019 15th International Computer Engineering Conference (ICENCO)*, Cairo, Egypt: IEEE, pp. 1–5. Available at: https://doi.org/10.1109/ICENCO48310.2019.9027294.

Silver, D. *et al.* (2014) ‘Deterministic Policy Gradient Algorithms’, *ICML* [Preprint].

Snouffer, E. (2022) ‘Six places where drones are delivering medicines’, *Nature Medicine*, 28(5), pp. 874–875. Available at: https://doi.org/10.1038/d41591-022-00053-9.

Song, Y. *et al.* (2021) ‘Autonomous Drone Racing with Deep Reinforcement Learning’. Available at: https://doi.org/10.48550/ARXIV.2103.08624.

Sutton, R.S. (1988) ‘Learning to Predict by the Methods of Temporal Differences’, *Machine Learning*, 3(1), pp. 9–44. Available at: https://doi.org/10.1023/A:1022633531479.

Sutton, R.S. and Barto, A.G. (1998) *Reinforcement learning: an introduction*. Cambridge, Mass: MIT Press (Adaptive computation and machine learning).

Turkoglu, K. *et al.* (2008) ‘Pid Parameter Optimization Of An Uav Longitudinal Flight Control System’. Available at: https://doi.org/10.5281/ZENODO.1079352.

Wang, H. *et al.* (2013) ‘Optimization of LQR controller for inverted pendulum system with artificial bee colony algorithm’, in *Proceedings of the 2013 International Conference on Advanced Mechatronic Systems*. *2013 International Conference on Advanced Mechatronic Systems (ICAMechS)*, China: IEEE, pp. 158–162. Available at: https://doi.org/10.1109/ICAMechS.2013.6681769.

Watkins, C.J.C.H. and Dayan, P. (1992) ‘Q-learning’, *Machine Learning*, 8(3–4), pp. 279–292. Available at: https://doi.org/10.1007/BF00992698.

Yu, G.-R. and Hsieh, P.-H. (2019) ‘Optimal Design of Helicopter Control Systems Using Particle Swarm Optimization’, in *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*. *2019 IEEE International Conference on Industrial Cyber Physical Systems (ICPS)*, Taipei, Taiwan: IEEE, pp. 346–351. Available at: https://doi.org/10.1109/ICPHYS.2019.8780133.

Zhang, X. *et al.* (2014) ‘A Survey of Modelling and Identification of Quadrotor Robot’, *Abstract and Applied Analysis*, 2014, pp. 1–16. Available at: https://doi.org/10.1155/2014/320526.

Zulu, A. and John, S. (2014) ‘A Review of Control Algorithms for Autonomous Quadrotors’, *Open Journal of Applied Sciences*, 04(14), pp. 547–556. Available at: https://doi.org/10.4236/ojapps.2014.414053.