**Title: Reinforcement learning based Linear quadratic Regulator for the Control of a Quadcopter**

**Contents**

Title Page (see above)

Declaration (see above)

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 three 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. 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 40% or more.

Acknowledgements

List of Contents (sections and sub-section with page numbers)

List of Symbols

Note a list of Figures and Tables is NOT required. Neither is it necessary to include units in your nomenclature.

Main Body of work

This will be split into any number of sections, with appropriate titles and numbered 2, 3 etc.

1. **Introduction**
   1. Motivation: Linear Quadratic regulator(LQR) and difficulty is choosing Q and R matrices (design matrices).
   2. Application: Drones! Benefits of faster response.
2. **Literature survey**: *Analyse advantages and disadvantages of literature in each section below.*
   1. LQR: its application to various systems (including drones).
      1. Survey of methods for choosing design matrices in LQR.
   * Example: Bryson’s rule, reinforcement learning (RL).
   * Note: Mention that using LQR with RL still guarantees optimality whereas using standalone RL agent to produce control inputs provides no such guarantee
     1. Methods for combining LQR with Proportional Integral Derivative (PID) control.
   * This should include Integral Square Error (ISE) optimization
   1. Reinforcement Learning for continuous action space
   * Include RL algorithm used in code: <https://arxiv.org/abs/1509.02971>
3. **Theory**
   1. Quadrotor dynamics, state space equation
   2. LQR
   3. PID control
      1. Deriving PID gains using ISE optimization
   4. Reinforcement Learning background
4. **Methodology (can make a separate section for benchmarks)**
   1. Control method for drone: LQR/ LQR+ PD control
   2. Bryson’s rule for selecting design matrices
      1. Changing all diagonal elements
      2. Changing only 4 diagonal elements.
   3. Reinforcement learning for selecting design matrices
      1. Task formulation: state, transition, action and reward in our context. (mention we are tuning only 4 diagonal elements)
      2. Actor, Critic neural network architecture. More details about the implementation like batch normalization.
5. **Numerical Experiments**

Few words about Numerical Experiments setup: Training and test conditions and how set point or trajectory is generated.

* 1. Test 1: Reaching a set point.

Results should include:

1. Training curve for RL agent: mean reward, reward per episode, number of failures, average length of episode.
2. Number of failures while training with RL agent. (the drone should be able to reach the set point more consistently as the training episodes increase)
3. ISE of all 4 methods over 100 random tests. Report number of failures when testing with RL agent.
4. Sample plot with all 4 methods.
   1. Test 2: Following a trajectory
5. Training curve for RL agent.
6. Number of failures while training with RL agent. (the drone should be able to reach the desired goal more consistently as the training episodes increase)
7. ISE of all 4 methods over 100 random tests. Report number of failures when testing with RL agent.
8. Sample plot with all 4 methods.
   1. Test 3: TODO: any additional tests.

This could include:

1. giving extra features like next two waypoints to the RL agent.
2. Different reward functions:
   1. Penalty due to limits on maximum input values (due to limit on maximum torque generated by motors).
   2. Penalty due to singularity of Q matrix: The Q matrix in LQR should well behaved.
3. Robustness of RL agent to noise in sensor measurements or model.
4. Robustness and response of RL agent to sudden change in trajectory due to obstacle.
5. **Conclusion**

**References** Not a numbered section

**Appendices** (designated Appendix A, B etc. with subsections A1, A2, B1 if appropriate)

1. **Methodology**

Using the state space equation defined in equation 11 and the algorithms discussed in section 3, four methods were tested for controlling the quadrotor. Each method was tested for its ability to reach a desired set point and follow a trajectory. This section will describe all four control methods.

|  |  |
| --- | --- |
| **Controller** | **Method of choosing Q matrix** |
| LQR | Identity matrix |
| LQR | Bryson's rule |
| LQR | RL |
| LQR + PD | Bryson's rule |

* 1. **LQR with Identity Q matrix**

In this controller, LQR as describe 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 follows 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 quadrotor to stay at desired state. The Q matrix for this controller will have the following form

Figure 1 shows the control block for this controller.

Figure 1.

* 1. **LQR with Bryson’s rule to tune Q matrix**

This controller is similar to the LQR controller defined in section 1.1. The only difference between the two controllers is the way we set Q matrix. Bryson’s rule will be used to set the diagonal elements of 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 with the square of the expected maximum value its corresponding state can take. This is particularly useful if the units of elements in 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 .

* 1. **LQR with Bryson’s rule to tune Q matrix and additional PD controller**

A PD control in addition to the controller defined in section 1.2 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 1) will have the following modified form

In the above equation is the input from 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,

* 1. **LQR with RL agent to tune Q matrix**

This section will introduce our implementation of the DDPG algorithm discussed in section 3.4.5. We will look at our choice of state which is 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,

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 1.2.

**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 (ref). This incentivizes the agent to reach its goal (or subgoal) in 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 3.4.5, both the actor and the critic are neural networks whose weights are updated to learn the optimal policy (equation 25) and the Q function for the optimal policy (equation 26), respectively. Figure 1 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 normalization (ref). 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 1. 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 2). For this state-action pair the critic gives as output the expected Q value (equation 23). The inputs are first scaled. 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 2: Critic Neural network design

Scale input

16×64

Fully connected Layer

FC 1

Input

1×12

(State input)

1×12

Scale the input vector

Concatenation

Layer

1×1

Range: [-1000,1000]

1×16

64×64

Fully connected Layer

FC 2

64×64

Fully connected Layer

FC 3

1×1

Range: [0,1]

64×1

Fully connected Layer

Scale output

Tanh Layer

FC 4

Scale Action

Action

1×4

Scale the action vector

1×4

(Action input)

64×64

Fully connected Layer

1×24

Scale the input vector

4×4

Range: [0.01,100]

4×4

Range: [0,1]

4×64

Fully connected Layer

FC 3

Scale output

Sigmoid Layer

FC 4

1×24

(State input + goal input)

24×64

Fully connected Layer

FC 2

Input

Scale input

FC 1

64×64

Fully connected Layer