**Contents**

Title Page (see above)

Declaration (see above)

Abstract

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: 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.
   * 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 PID control.
   * This should include 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**
   1. Control method for drone: LQR/ LQR+ PD control
   2. Bryson’s rule for selecting design matrices
   3. Reinforcement learning for selecting design matrices
      1. Task formulation: state, transition, action and reward in our context.
      2. Actor, Critic neural network architecture. More details about the implementation like batch normalization.
5. **Experiments**
   1. Few words about Experimental setup: Training and test conditions and how set point or trajectory is generated.
   2. Test 1: Reaching a set point.

Results should include:

1. Training curve for RL agent.
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)

**Started writing about the code to generate more ideas. You may ignore this.**

REINFORCEMENT LEARNING BASED DESIGN OF LQR CONTROLLER WITH APPLICATION TO DRONES

**Motivation**

The Linear Quadratic Regulator (LQR) is a well-known linear feedback controller that provides optimal feedback gains to ensure system stability and high performance. The performance of a system is defined using a cost function which is a function of control inputs and deviations from desired state. 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. The aim of this research is to design a method to specify the cost function parameters (Q and R matrices) without human intervention to achieve a faster and stable response.

**Application**

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 have an inherent urgency which require a quick and stable response. In the proposed method LQR is used along with an actor-critic Reinforcement Learning (RL) algorithm to set the values in Q and R matrices. This method is benchmarked with the following two approaches:

1. Set Q and R to identity matrices.
2. Use Bryson’s rule to set Q and R matrices.
3. Use Bryson’s rule to set Q and R matrices and use a PD controller along with LQR.

**Implementation Details**

* **Simulation environment**

**Diagram

Description automatically generated**

Figure 1: Drone environment Design

A simulation has been created in MATLAB to carry out the tests. *DroneEnvironment*\_*BaseClass* is built on top of *rl.env* class provided by MATLAB to help build simulation environments for RL projects. *DroneEnvironment*\_*BaseClass* implements the dynamic equations of motion and functions to generate random trajectories for testing and training. Sub classes for *DroneEnvironment*\_*BaseClass* are created to test the proposed method and the three benchmark approaches as listed above.

* **RL architecture**

An actor-critic model-free RL algorithm has been chosen to determine the Q and R matrices. This algorithm is based on deterministic policy gradient and has shown to work well over continuous action space. More details can be found at: <https://arxiv.org/abs/1509.02971>. Figure 2 shows the Neural Network architecture used to implement the *critic* module. The *critic* predicts the Q value of given state-action pair. The input (state and action), is scaled and passed through four fully connected layers. The output is scaled to appropriate bounds.

**Chart, line chart

Description automatically generated**

Figure 2: Critic Design

Figure 3 shows the *actor* architecture. The *actor* choses an action (Q and R matrices for our application) given the state of drone to maximise reward. Similar to the *critic’s* design, the input is first scaled and then passed through four fully connected layers.

Chart, radar chart

Description automatically generated

Figure 3: Actor Design

The scaling parameters for both *actor* and *critic* networks are the average values observed by running LQR over 100 episodes. These episodes are different from test and train episodes. A projection-based path progress reward is used along with positive reward for reaching a waypoint.

**Tests and Results**

The table below shows the Integral Square Error (ISE) values for the four methods tested. The values have been summed over 100 test episodes. RL method shows the fastest convergence and hence the lowest time of flight. On average using an RL tuned LQR reduces the time of flight by 50% when compared with vanilla LQR method.

TABLE 1: ISE values over 100 test episodes

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| **LQR** | **Bryson's rule + LQR** | | **Bryson' rule + LQR + PD** | | **RL + LQR** | |
| -25500.574 | -18240.70288 | | -30348.22103 | | -6075.979609 | |
| -27880.956 | -20342.35863 | | -19448.26556 | | -6336.997954 | |
| -29102.431 | -20833.46896 | | -26231.57935 | | -5775.982727 | |
| -16.233905 | -45.15833491 | | -4557.43565 | | -72.29666477 | |
|  | |  | |  | |

A picture containing diagram

Description automatically generated

Figure 4: Example of a test trajectory using RL + LQR

**Future work**

These promising results have motivated further research into designing an even better RL agent to tune LQR. Current work is looking at:

1. Adding additional waypoint data into the state variable. The motivation is to allow the agent to see future waypoints while deciding the parameters for current waypoint.
2. Test different reward functions.

**How to run the project through MATLAB command line:**

Command to train the model run: *run\_DDPG*

Command to test all 4 methods: *get\_score\_controller(environment, method)*

* Here environment is the object of method you want to test.
* Example command: *get\_score\_controller(DroneEnvironment\_LQR('test'),”LQR”))*

Plot test trajectories:

* For RL method: *predict\_rl*
* For other methods: *predict\_classic*

**ADDITIONAL DOCUMENTS**

The following additional documents can be found in the *Documentation* folder:

* Bryson\_Rule.docx: Motivation behind Bryson’s rule for tuning Q and R matrices with results for step input
* Error\_Response\_with\_PID\_controller.docx: Analysis of the response of system with PID controller to step input.
* PID\_gains\_from\_ISE\_optimization.docx: Solution of closed form equations to obtain optimal PID gains to minimise the integral square error.
* Reinforcement\_Learning.docx: Response of RL tuned LQR to step input and comparison with other methods to tune LQR.