**Title: Application of Reinforcement learning to select design matrices in LQR for Drone control**

**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: 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. **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.

* 1. **Quadrotor dynamics**

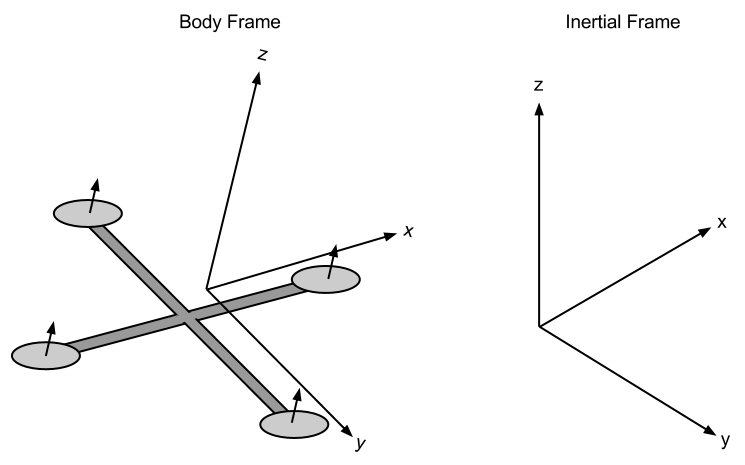


Figure 1. Plus configuration quadrotor with body and inertial frames

A quadrotor should be able to execute a range of motion with four rotors attached symmetrically to its four arms, as shown in figure 1. 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, while this 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 1). For plus configuration, sideways motion is achieved by rotating left (right) rotors faster than right (left) rotors. This can be seen in figures 2.d and 2.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 2. Execution of different motions in plus configuration.

* + 1. **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 1). 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 form 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 thrust force , roll torque , pitch torque , and yaw torque . These inputs can be represented in terms of the rotation speed of the propellers as shown in the equation below.

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

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

* + 1. **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

* + 1. **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

* 1. **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,

Where 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 Ricatti equation

* 1. **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, the proportional (), derivative () and integral () gain values. Figure 3 shows the block diagram of the drone system containing a PID controller.

Diagram

Description automatically generated

Figure 3 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

* + 1. **Finding PID gains using ISE optimization**

To find the PID gain values, the performance index employed by Turkoglu *et al.*, 2008 for longitudinal UAV flight control was used.

Minimizing the above function will result in a fast and stable response. However, the error function in equation 17 is in s domain whereas is in time domain. To convert from time to s domain we will use the 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). Using precalculated integral for equation 19 (Gradshteĭn, Ryzhik and Jeffrey, 2000) and minimizing the obtained expression we can find the optimal gain values.

* 1. **Reinforcement Learning**

**Diagram

Description automatically generated**

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 4). Through this experience, the agent can determine the optimal action suited to every state of the environment to maximize 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 on taking an 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 known as 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 thereafter. The objective of the RL agent is to learn the optimal policy

In the following sub-sections (3.4.1 to 3.4.5), we will describe the main features and give an overview of the training steps of the DDPG algorithm.

* + 1. **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).

* + 1. **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, 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 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.

* + 1. **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.

* + 1. **Actor-Critic Algorithms**

**Diagram

Description automatically generated**

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).

* + 1. **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.

**Agent**

action

At

state

St

reward

rt

Rt+1

**Environment**

St+1

Policy

*Actor*

action

*Critic*

Value

Function

state

reward

**Environment**