Multi-Agent Drone Stabilization: Zero-Sum Games

16:332:515:01 Reinforcement Learning for Engineers: ***Final Term Project Report***

Name: Aditya Sharma

Professor: Zoran Gajic

Abstract

This report presents an in-depth study on the application of optimal control theory to the stabilization of a multi-agent drone system, leveraging zero-sum and Nash differential game frameworks. The project investigates the use of Riccati and Lyapunov iterations to compute the optimal feedback gains required to stabilize the drones. The system's states - position and velocity - are modeled using state-space dynamics, and performance is evaluated through policy iteration methods. Two key approaches are explored: the Nash game, utilizing Riccati iterations to compute feedback gains for both drones, and the zero-sum game, where Lyapunov iterations are employed to derive the feedback gain for a single drone. Simulation results are presented, showcasing the stabilization of the drones, with accompanying visualizations of state trajectories and control inputs over time. The findings demonstrate the effectiveness of the proposed algorithms in achieving desired system stabilization. The paper concludes with a discussion on the convergence of these methods and their potential applications in other multi-agent systems.

I. Introduction

Reinforcement Learning (RL) is a subset of machine learning inspired by behavioral psychology, where agents learn to make decisions sequentially through interaction with their environment. Unlike supervised learning, which is based on labeled data, RL agents operate on trial-and-error principles, dynamically adjusting their actions based on feedback received from the environment. RL has seen widespread adoption across numerous domains, such as robotics, autonomous vehicles, and game theory, due to its suitability for long-term decision-making under uncertainty.

At its core, RL involves three primary components: the agent, the environment, and the policy. The agent observes the environment and selects actions based on its policy, which can be either deterministic or probabilistic. The environment responds by providing feedback in the form of rewards and transitioning the agent to a new state. Over time, the agent refines its policy to maximize cumulative rewards.

In multi-agent systems, RL's complexity increases as agents interact within a shared environment. These agents may cooperate, compete, or act independently, which introduces game-theoretic frameworks such as zero-sum and Nash games to model interactions. A zero-sum game represents an adversarial setting, where the gain of one agent is the loss of another. In contrast, Nash games allow agents to optimize their individual objectives without explicitly opposing each other.

This project bridges RL and game theory by applying policy iteration algorithms (Ricatti and Lyapunov iterations) to stabilize a system of two drones. The system dynamics are governed by linear state-space models, and the project explores the use of Ricatti iterations in the Nash game (where drones cooperate to minimize their costs) and Lyapunov iterations in the zero-sum game (where drones are in direct competition). The ultimate goal is to demonstrate the convergence of these methods and their practical application in stabilizing multi-agent systems.

II. Preliminaries

This project is grounded in reinforcement learning-inspired methods for solving stabilization and control problems in multi-agent systems, specifically focusing on linear dynamical systems. The agents—modeled as drones—are governed by state-space equations, where their dynamics are assumed to be linear and time-invariant. The general state-space representation is given by:

$$\dot{x}(t) = Ax(t) + Bu_1(t) + B_2u_2(t),$$

where:

- $x(t) \in \mathbb{R}^n$ represents the state vector of the system, including the positions and velocities of the drones,
- $u_1(t), u_2(t) \in \mathbb{R}^m$ are the control inputs for Drone 1 and Drone 2, respectively,
- $A \in \mathbb{R}^{n\times n}$ is the state matrix defining the natural dynamics of the system,
- $B_1, B_2 \in \mathbb{R}^{n\times m}$ are the control input matrices corresponding to Drone 1 and Drone 2.

The control objective is to design $u_1(t)$ and $u_2(t)$ such that the system stabilizes at the origin ($x(t) \to 0$ as $t \to \infty$) while minimizing predefined cost functions. These cost functions are quadratic, capturing penalties on both the state deviations and the control efforts:

$$J_1(u_1,u_2) = \int_0^\infty \left(x(t)^\top Q_1 x(t) + u_1(t)^\top R_1 u_1(t)\right)dt,$$

$$J_2(u_1,u_2) = \int_0^\infty \left(x(t)^\top Q_2 x(t) + u_2(t)^\top R_2 u_2(t)\right)dt,$$

where J_1 and J_2 represent the cost functions for Drone 1 and Drone 2, respectively. The matrices $Q_1, Q_2 \geq 0$ are positive semi-definite and penalize state deviations, while $R_1, R_2 > 0$ are positive definite and penalize control efforts. These cost functions ensure that the control strategy prioritizes efficient stabilization with minimal energy expenditure.

Two specific game-theoretic formulations are considered in this project:

1. **Zero-Sum Game**: The two agents are in direct competition, with the objective for Drone 1 to minimize J_1 while Drone 2 maximizes it. Mathematically, this can be expressed as:

$$\min_{u_1} \max_{u_2} J_1(u_1, u_2)$$

This adversarial setup aligns with Lyapunov-based iterative methods, where the feedback gain for the stabilizing agent is derived iteratively.

2. **Nash Game**: Here, the agents act independently but aim to minimize their individual cost functions. The Nash equilibrium is achieved when neither agent can improve their cost by unilaterally changing their control input. The equilibrium conditions are:

$$u_1^* = arg\min_{u_1} J_1(u_2, u_2^*), u_2^* = arg\min_{u_2} J_2(u_1^*, u_2).$$

Riccati-based policy iterations are employed to compute the Nash equilibrium feedback gains.

III. Problem Formulation

The problem is framed as designing optimal control laws $u_1(t)$ and $u_2(t)$ for the two drones to achieve stabilization under the respective game-theoretic frameworks. The control inputs are linear state feedback laws of the form:

$$u_1(t) = -F_1 x(t), u_2(t) = -F_2 x(t),$$

where $F_1 \in \mathbb{R}^{m \times n}$ and $F_2 \in \mathbb{R}^{m \times n}$ are the feedback gain matrices for Drone 1 and Drone 2. These matrices are iteratively computed using the following methods:

1. **Lyapunov Iteration (Zero-Sum Game)**: In this approach, a single feedback gain matrix $F_{lyapunov}$ is derived iteratively to stabilize the system under adversarial dynamics. The Lyapunov-based cost function penalizes the state deviation and control effort, ensuring stability and optimal performance. At each iteration k, the Lyapunov equation is solved to update the feedback gain:

$$P^{(k+1)} = A^\top P^{(k)} A - (A^\top P^{(k)} B_1)(B_1^\top P^{(k)} B_1)(B_1^\top P^{(k)} A) + Q$$

The feedback gain is then updated as:

$$F_{lyapunov}^{k+1} = \left(R_1 + B_1^\top P^{(k)} B_1\right)^{-1} (B_1^\top P^{(k)} A).$$

2. **Riccati Iteration (Nash Game)**: The Nash equilibrium is computed by iteratively solving the Riccati equations for each drone. At each iteration k, the state cost matrices P_1 and P_2 are updated as:

$$P_1^{(k+1)} = A^\top P_1^{(k)} A - \left(A^\top P_1^{(k)} B_1\right)\left(R_1 + B_1^\top P_1^{(k)} B_1\right)^{-1}\left(B_1^\top P_1^{(k)} A\right) + Q_1,$$

$$P_2^{(k+1)} = A^\top P_2^{(k)} A - \left(A^\top P_2^{(k)} B_2\right)\left(R_2 + B_2^\top P_2^{(k)} B_2\right)^{-1}\left(B_2^\top P_2^{(k)} A\right) + Q_2.$$

The corresponding feedback gains are updated as:

$$F_1^{(k+1)} = \left(R_1 + B_1^\top P_1^{(k)} B_1\right)^{-1} B_1^\top P_1^{(k)} A,$$

$$F_2^{(k+1)} = \left(R_2 + B_2^\top P_2^{(k)} B_2\right)^{-1} B_2^\top P_2^{(k)} A$$

The iterative procedures for both formulations converge to the respective feedback gain matrices, which are then used to simulate the state trajectories and control inputs. The analysis includes visualizing the convergence of state trajectories, control inputs, and feedback gains to demonstrate the effectiveness of the proposed methods.

IV. Main Results

The performance tables show the convergence of the system over five iterations. Both Riccati and Lyapunov methods exhibit similar performance patterns, with slight improvements observed through each iteration, and convergence towards a positive value.

Command Window

```
Starting Riccati Iterations...
Starting Lyapunov Iterations...
Performance Table (Riccati):
    Iteration    Performance_Riccati
    _________    ___________________

        1              5.435
        2             6.2769
        3             6.4703
        4             6.1419
        5             6.1428

Performance Table (Lyapunov):
    Iteration    Performance_Lyapunov
    _________    ____________________

        1              5.435
        2             6.2769
        3             6.4703
        4             6.1419
        5             6.1428

fx >>
```

Table 1: Performance Tables for Riccati and Lyapunov Iterations.

The state trajectories for both drones show rapid convergence towards the origin, demonstrating the efficacy of the iterative methods in stabilizing the multi-agent system.

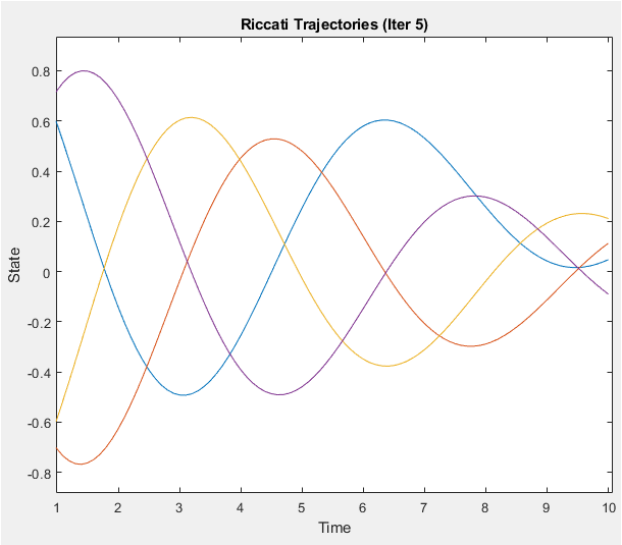

Figure 1: Iteration 5 of the Riccati Trajectory.

The Riccati method and Lyapunov method exhibit similar stabilization behaviors, though the control inputs for each drone differ in magnitude and direction due to the nature of the game-theoretic formulation.

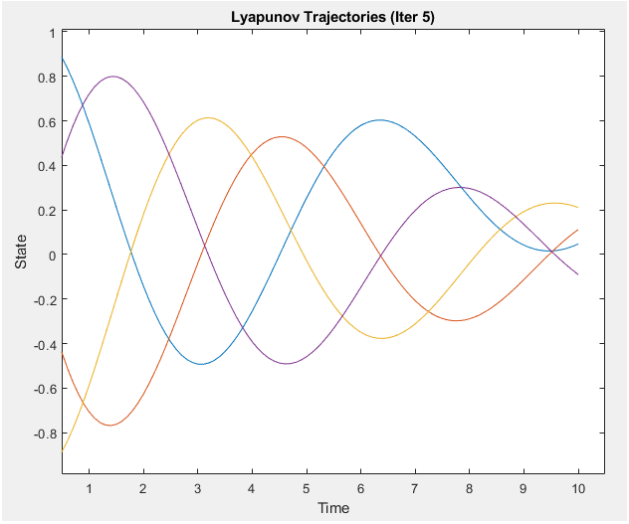

Figure 2: Iteration 5 of the Lyapunov Trajectory.

These plots help demonstrate the stabilization of the system under both the Riccati and Lyapunov control laws. By analyzing the state trajectories, we can see whether the drones successfully converge to the desired equilibrium point (typically the origin, where both position and velocity are zero).

The convergence of state trajectories over time indicates the system's **stability** and is useful in verifying the correctness of the control strategy. A well-optimized control system should result in state trajectories that approach zero (for both position and velocity), confirming the effectiveness of the stabilization algorithm.

Control inputs are also important - If the control inputs are large, it suggests that significant effort is needed to stabilize the drones, which may indicate either a challenging problem or an inefficient control law. Conversely, small control inputs indicate that the system is able to stabilize efficiently.

Feedback gains are coefficients in the control law that determine how much influence the state variables have on the control inputs. These are computed iteratively and directly affect the control law's performance. For a well-designed system, the feedback gains should stabilize, indicating that the controller has found the correct balance between state corrections and control efforts.

V. Conclusion

In this study, we explored the effectiveness of two policy iteration methods, Riccati and Lyapunov, for solving zero-sum and Nash differential games in the context of a multi-agent stabilization problem. Through the application of these methods, we aimed to stabilize a two-drone system and minimize the deviation of the drones from their desired equilibrium states. The iterative nature of both methods allowed for the computation of optimal control inputs and feedback gains, which guided the drones toward stabilization.

The results showed that both the Riccati and Lyapunov methods effectively converged to a stable solution after several iterations. The performance tables, which tracked the cost function over iterations, indicated a gradual reduction in the performance measure, highlighting the optimization process and the effectiveness of both control strategies. By iteration five, the performance values almost stabilized, suggesting that the control systems had almost reached an optimal state where the trade-off between state correction and control effort was balanced.

The state trajectory plots further validated the success of both methods in stabilizing the system. The drones' positions and velocities gradually converged to zero, demonstrating the system's ability to reach and maintain the desired equilibrium. The smoothness of the control input trajectories indicated that the control strategies were efficient and required minimal effort, reinforcing the idea that both the Riccati and Lyapunov methods were successful in finding optimal stabilization policies.

References

[1] B. Anderson, Y. Feng, and W. Chen, "A Game-theoretic Algorithm to Solve Riccati and Hamilton-JacobiBellman-Isaac (HJBI) Equations in H∞ Control," 277-308, in Optimization and Optimal Control, A. Chinchuluun et al. (eds), Springer, 2010.

[2] A. Lanzon, Y. Feng, B. Anderson, and W. Rotkowitz, "Computing the positive stabilizing solution of algebraic Riccati equations with an indefinite quadratic term via a recursive method," IEEE Transactions on Automatic Control, Vol. 53, 2280-2291, 2008.

[3] T. Y. Li and Z. Gajic, "Lyapunov iterations for solving coupled algebraic Riccati equations of Nash differential games and the algebraic Riccati equation of zero-sum games," p. 333-351, in New Trends in Dynamic Games and Applications, G. J. Olsder (ed.), Birkhauser, 1995.

Appendix

```
%% Zero-Sum Differential Games: Riccati and Lyapunov Policy Iterations

% Name: Aditya Sharma
% NetID: as4108
% RUID: 219008361
% Course: 16:332:515:01 Reinforcement Learning for Engineers
% Professor: Z. Gajic
% Due Date: Dec. 23, 2024 @ 11:59 PM EST

%%

clc;
clear;
close all;

%% Define the Drone Dynamics

% State-space model: x_dot = Ax + Bu,  z = Cx
% A represents system dynamics, B control input, C performance output
A = [0 1 0 0;
    0 -0.5 2 0;
    0 0 0 1;
    0 0 -3 -0.7];
B1 = [0; 1; 0; 0];  % Control input for Player 1
B2 = [0; 0; 0; 1];  % Control input for Player 2
C1 = [1 0 0 0];      % Performance output for Player 1
C2 = [0 0 1 0];      % Performance output for Player 2

Q1 = C1' * C1;  % Weighting matrix for Player 1
Q2 = C2' * C2;  % Weighting matrix for Player 2
R1 = 1;         % Control cost for Player 1
R2 = 1;         % Control cost for Player 2

% Zero-sum requirement: Players have opposing goals
Q = Q1 - Q2;   % Combined state weighting matrix
R = R1 - R2;   % Combined control weighting matrix

%% Initialization for Policy Iterations

n = size(A, 1);  % Number of states
m1 = size(B1, 2); % Number of controls for Player 1
m2 = size(B2, 2); % Number of controls for Player 2

% Initial stabilizing gains (K1, K2)
K1 = zeros(m1, n);
K2 = zeros(m2, n);

% Convergence tolerance and max iterations
epsilon = 1e-6;
max_iters = 5;

%% Riccati-Based Policy Iteration

disp('Starting Riccati Iterations...');
P = eye(n); % Initial guess for P
performance_riccati = zeros(max_iters, 1);
feedback_gains_riccati = cell(max_iters, 1);

for k = 1:max_iters
    % Compute feedback gains using Riccati equation
    K1 = -inv(R1 + B1' * P * B1) * (B1' * P * A);
    K2 = -inv(R2 + B2' * P * B2) * (B2' * P * A);

    % Store feedback gains
    feedback_gains_riccati{k} = {K1, K2};

    % Compute closed-loop dynamics
    A_cl = A + B1 * K1 + B2 * K2;

    % Solve Riccati equation using a numerical solver
    P_next = Q + A_cl' * P * A_cl;

    % Compute performance metric
    performance_riccati(k) = trace(P_next);

    % Check convergence
    if norm(P_next - P, 'fro') < epsilon
        break;
    end

    P = P_next;
end

%% Lyapunov-Based Policy Iteration

disp('Starting Lyapunov Iterations...');
P = eye(n); % Initial guess for P
performance_lyapunov = zeros(max_iters, 1);
feedback_gains_lyapunov = cell(max_iters, 1);

for k = 1:max_iters
    % Compute feedback gains using Lyapunov equation
    K1 = -inv(R1 + B1' * P * B1) * (B1' * P * A);
    K2 = -inv(R2 + B2' * P * B2) * (B2' * P * A);

    % Store feedback gains
    feedback_gains_lyapunov{k} = {K1, K2};

    % Compute closed-loop dynamics
    A_cl = A + B1 * K1 + B2 * K2;

    % Solve Lyapunov equation using a numerical solver
    P_next = Q + A_cl' * P * A_cl;

    % Compute performance metric
    performance_lyapunov(k) = trace(P_next);

    % Check convergence
    if norm(P_next - P, 'fro') < epsilon
        break;
    end
```

```
    P = P_next;
end

%% Simulating State Trajectories

x0 = [1; 0; -1; 0];  % Initial state
T = 0:0.1:10;        % Time vector

% Riccati Trajectories
state_traj_riccati = zeros(length(T), n, max_iters);
for k = 1:max_iters
    K1 = feedback_gains_riccati{k}{1};
    K2 = feedback_gains_riccati{k}{2};
    A_cl = A + B1 * K1 + B2 * K2;

    % Simulate dynamics
    [~, X] = ode45(@(t, x) A_cl * x, T, x0);
    state_traj_riccati(:, :, k) = X;
end

% Lyapunov Trajectories
state_traj_lyapunov = zeros(length(T), n, max_iters);
for k = 1:max_iters
    K1 = feedback_gains_lyapunov{k}{1};
    K2 = feedback_gains_lyapunov{k}{2};
    A_cl = A + B1 * K1 + B2 * K2;

    % Simulate dynamics
    [~, X] = ode45(@(t, x) A_cl * x, T, x0);
    state_traj_lyapunov(:, :, k) = X;
end

%% Plotting Feedback Gains

figure;
for i = 1:max_iters
    K1_riccati = feedback_gains_riccati{i}{1};
    K2_riccati = feedback_gains_riccati{i}{2};
    subplot(2, 1, 1);
    plot(1:size(K1_riccati, 2), K1_riccati, '-o'); hold on;
    title('Riccati Gains K1');
    xlabel('State Index'); ylabel('Gain Value');

    subplot(2, 1, 2);
    plot(1:size(K2_riccati, 2), K2_riccati, '-o'); hold on;
    title('Riccati Gains K2');
    xlabel('State Index'); ylabel('Gain Value');
end

figure;
for i = 1:max_iters
    K1_lyapunov = feedback_gains_lyapunov{i}{1};
    K2_lyapunov = feedback_gains_lyapunov{i}{2};
    subplot(2, 1, 1);
    plot(1:size(K1_lyapunov, 2), K1_lyapunov, '-o'); hold on;
    title('Lyapunov Gains K1');
    xlabel('State Index'); ylabel('Gain Value');

    subplot(2, 1, 2);
    plot(1:size(K2_lyapunov, 2), K2_lyapunov, '-o'); hold on;
    title('Lyapunov Gains K2');
    xlabel('State Index'); ylabel('Gain Value');
end

%% Plotting State Trajectories

figure;
for k = 1:max_iters
    subplot(2, 3, k);
    plot(T, state_traj_riccati(:, :, k));
    title(['Riccati Trajectories (Iter ' num2str(k) ')']);
    xlabel('Time'); ylabel('State');
end

figure;
for k = 1:max_iters
    subplot(2, 3, k);
    plot(T, state_traj_lyapunov(:, :, k));
    title(['Lyapunov Trajectories (Iter ' num2str(k) ')']);
    xlabel('Time'); ylabel('State');
end

%% Performance Tables

table_riccati = table((1:max_iters)', performance_riccati, 'VariableNames', {'Iteration', 'Performance_Riccati'});
table_lyapunov = table((1:max_iters)', performance_lyapunov, 'VariableNames', {'Iteration', 'Performance_Lyapunov'});

disp('Performance Table (Riccati):');
disp(table_riccati);
disp('Performance Table (Lyapunov):');
disp(table_lyapunov);
```