ME550 Final Report

Audrey Gunawan, Yenpang Huang

November 2023

1 Introduction

The control of multi-agent systems is crucial, given the wide applicability of formation problems in diverse scenarios like spacecraft coordination, earth observation, and deep space exploration [1].

In practical applications, multi-agent systems are often tasked with achieving specific team performance objectives, including overall efficiency, minimal energy consumption, and optimization of efficiency. Integrating optimal tracking into these systems provides a means to address and fulfill these performance goals effectively.

Furthermore, the incorporation of time-varying formations enhances the adaptability of systems to changing conditions, showcasing the versatility of modern control systems. Time-varying formations, especially under time constraints, offer benefits such as improved energy efficiency—making them particularly practical for applications where multiple agents, such as drones or coordinated robotic manipulators, are deployed.

As mentioned in "Distributed time varying formation optimal tracking for uncertain Euler-Lagrange systems with time-varying cost functions", Multi agent Euler Lagrange systems are nonlinear models that can have tracking issues, and much of the research o these systems tend to focus on formation tracking, with the optimal performance not fully taken into account. This therefore highlights the importance of using time varying optimization. Using uncertain Euler-Lagrange systems are also needed when the systems are more likely to be subject to uncertainty, which therefore models cases that might happen in the real world. In this case, the Hamiltonian method is applied to a linear time invariant system, to simplify the system due to time constraints.

2 System Model and Optimization Objective

In the paper [1], an uncertain EL system with 12 agents is used. The article studies the distributed formation optimal tracking problem for the multi agent EL system. The multi agent system is used as $M_i(q_i)\ddot{q} + C_i(q_i)q_i + G_i(q_i) = \omega_i + \tau_i$.

However, in order to be able to simulate the system within the given time constraints, the Euler Lagrange system is simplified to a Linear Time Invariant (LTI) system and simulated by using a first order system with identifiable A, B, C and D matrices. The system used is

$$\dot{x}_1(t) = 2x_2(t)$$

$$\dot{x}_2(t) = -3x_1(t) - x_2(t) + 2u(t)$$

with the optimization of the local cost function:

$$J = \int_0^{10} \frac{1}{2}[(p_{ri} - q_i)^2 + u_i^2]dt$$

where the tracking parameter p_{ri} can be expressed as

$$p_{ri} = \begin{bmatrix} 0.1 \\ 0.07 \end{bmatrix} t + \begin{bmatrix} cos(0.02t) & -sin(0.02t) \\ cos(0.02t) & cos(0.02t) \end{bmatrix} v_i$$

v_i stands for disturbances in the system. For this paper, Gaussian noises with a 0.1 standard deviation are chosen for the simulation.
The global cost function is defined as

$$f[q(t), t] = \sum_{i=1}^{N} f_i[q_i(t), t]$$

Hence, the optimal values of global cost function can then be expressed as

$$q^*(t) = argmin f[q(t), t]$$

The circle pattern is chosen as the desired trajectory path, the formation pattern of the circle is given as:

$$h_i(t) = [cos([2(i-1)\pi/12] + 2t), sin([2(i-1]\pi/12] + 2t)]^T$$

with the boundary conditions being random initial states between the given range $q(0)\epsilon[-4, 4]$ and the goal state $q(tf) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$.

Formation vectors are vectors that encode the optimal positions of the agents in the system. These formation vectors will allow it so that the optimal tracking algorithms can use these vectors as reference points. These agents are also interdependent and will avoid collision with each other while it reaches the given goal state. With the goal of optimizing the state, this is applicable to different systems and can be used for fuel consumption optimization.
The optimization objective will be to analyze optimal tracking performance, where formation tracking and optimization is implemented.

3 Calculus of Variations and Solution Approach

The example equation will output the matrices.

$$\dot{x}(t) = \underbrace{\begin{bmatrix} 0 & 2 \\ -3 & 1 \end{bmatrix}}_{\text{A}} x(t) + \underbrace{\begin{bmatrix} 0 \\ 2 \end{bmatrix}}_{\text{B}} u(t)$$

$$R = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}, Q = \begin{bmatrix} 0.5 & 0 \\ 0 & 0.5 \end{bmatrix}$$

$$H = \frac{1}{2}(p_{ri} - q_i)^2 + \frac{1}{2} r u_i^2 + \lambda^T (Ax + Bu)$$

derivatives:

$$\frac{dH}{du} = rU + B^T \lambda = 0 \rightarrow u_k(t) = -r^{-1} B^T \lambda_k(t)$$

$$\dot{x} = \frac{dH}{d\lambda} = Ax - BR^{-1}B^T \lambda_*$$

$$\dot{\lambda} = \frac{-dH}{dx} = -C^T QCx - A^T \lambda + C^T Qz$$

$$\begin{bmatrix} \dot{x} \\ \dot{\lambda} \end{bmatrix} = \begin{bmatrix} A & -BR^{-1}B^T \\ -C^T QC & -A^T Q \end{bmatrix} \begin{bmatrix} x \\ \lambda \end{bmatrix} + \begin{bmatrix} 0 \\ C^T Q \end{bmatrix} z$$

boundary conditions: $x(t_0) = t_0$ are random initial states and $x(t_f) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$ is fixed,

$$\lambda(t_f) = \frac{dS}{dx}|_{t_f} = 0$$

With the Riccati method,

$$\lambda_*(t) = P(t)x(t) - g(t)$$

$$P(t_f) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}, g(t_f) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$$

$$[\dot{P} + PA + A^T P - PEP + V]x^*(t) - [\dot{g} + A^T g - PEg + wz]$$

where $E = BR^{-1}B, V = C^T QC, w = C^T Q$
where it is solved

$$\dot{P} + PA + A^T P - PEP + V = 0$$

$$\dot{g} + A^T g - PEg + wz$$

where the final condition is $P(t_f) = \begin{bmatrix} 0 & 0 \\ 0 & 0 \end{bmatrix}$ and $g(t_f) = \begin{bmatrix} 0 \\ 0 \end{bmatrix}$
The results are then plotted in section 5.

4 Numerical Simulation

The system simulated is divided into several steps.

Step 1: The system is first initialized by defining the number of agents, time boundaries, initial positions, and goal states.

Step 2: The system dynamics are then defined with the variables 'A', 'B', and 'C'. The optimal control is also designed, which is the objective J in the variables 'Q', 'R', and 'F'.

Step 3: The code used also uses the function 'Psolve' to get the Riccati matrices 'P' and 'g' for Riccati differential equation, which captures the information of the optimal control over time. The control gains 'K' and 'G' is calculated by solving the Riccati equation back in time.

Step 4: The formation vectors are then defined with different shapes, which includes the circle, star, lemniscate, and heart. The difference between the current positions and the desired formation vectors is then calculated.

Step 5: The states and inputs for each agent are computed using ODE integration with the control gains.

Further details can be checked in Appendix 7.

5 Simulation Results and Discussion

Figure 1 shows the Riccati matrices $P(t)$ and $g(t)$ over time. The plots show the visualization of how the optimization process adjusts its parameters to achieve optimal tracking over time. The $P(tf)$ and $g(tf)$ present in the figure matches the boundary condition mentioned in section 3 where both are equal to 0.

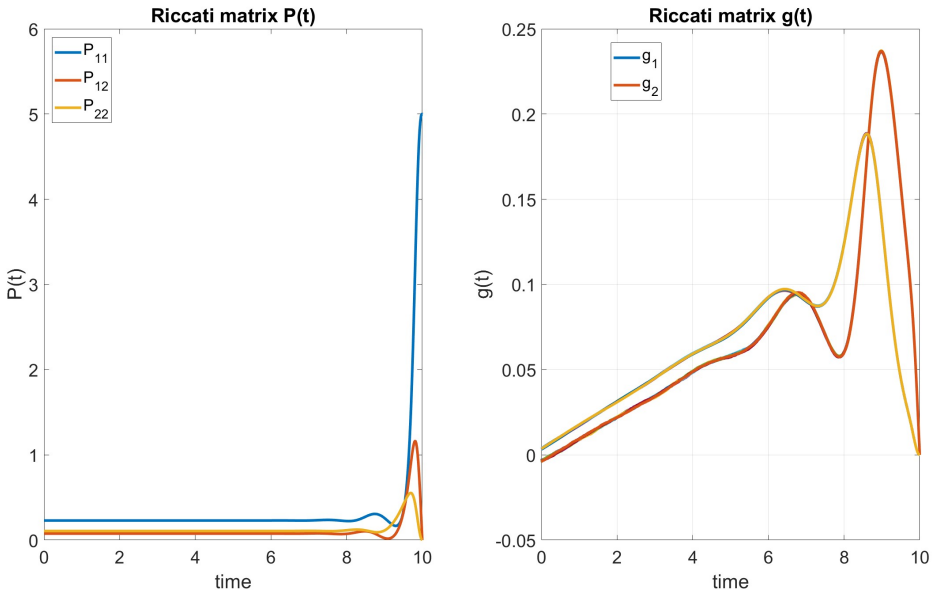

Figure 1: Riccati Matrices

Figure 2 shows the simulation result of trajectory paths for 12 agents from the randomly assigned initial states to a common goal state $(0, 0)$. With the time-varying tracking parameter p_{ri}, all agents are able to converge to the goal state within a short time. The tracking parameter takes into account the Gaussian noises, which satisfies the uncertain requirement. Noticing the fluctuating path depicted in the figure is a result of random Gaussian noises introduced into the tracking parameter.

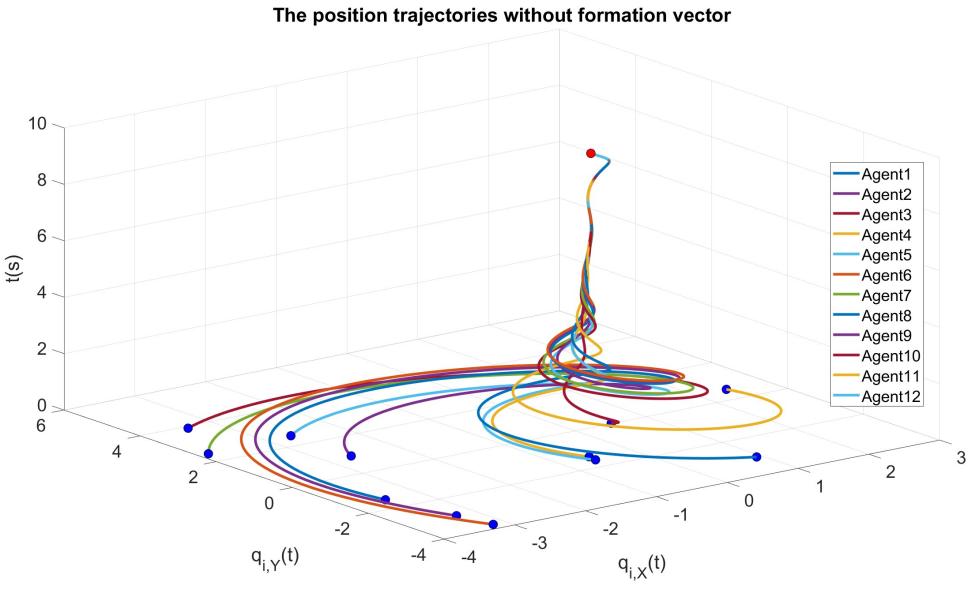

Figure 2: Position Trajectory without formation vectors

Figure 3 depicts the trajectories after the incorporation of formation vectors. By introducing interdependence among agents, the system ensures that each agent maintains a predefined distance from others, as the specified formation. In this case, the arrangements of the agents are shown to be a circle pattern. The

inclusion of these vectors enhances dynamic scalability and has the potential to optimize system efficiency.

The utilization of additional formation vectors allows for the effective distribution of workload, reducing mission completion times. Beyond facilitating efficient movement coordination, these formation vectors also play a crucial role in optimal tracking. The implementation demonstrates a marked improvement in the efficiency and smoothness of agent movements.

It is clear to see that the agents are no long overlapping on each other which achieves collision avoidance.

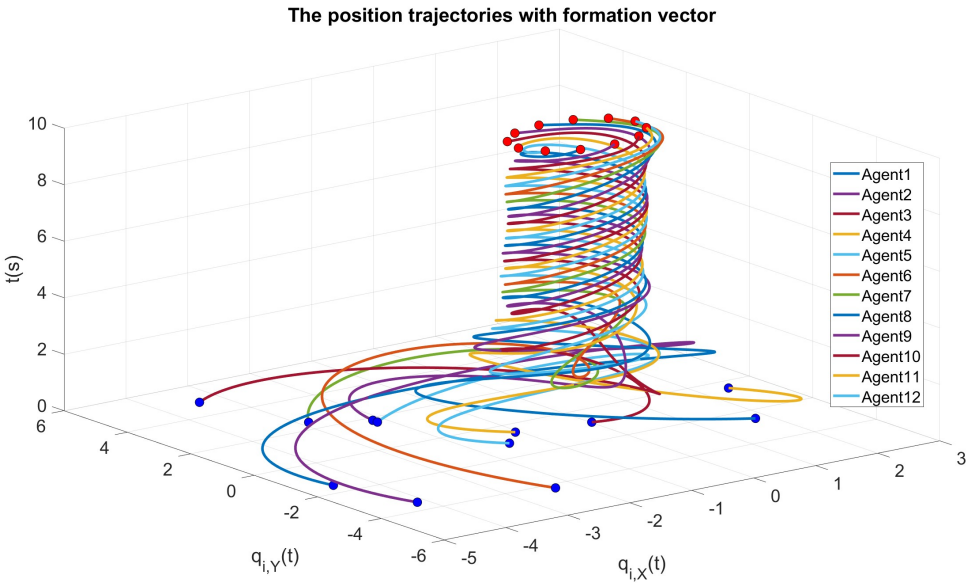

Figure 3: Position Trajectory with formation vectors.

Figure 4 5 6 7 8 are the snapshots of the states at $t = 0, 2.5, 5, 7.5, 10$. At $t = 0$, all the states are randomly located in the specified domain. The plots are showing a hint of the pattern, which is optimized to the formation pattern as desired over time. This is also a really good way to visualize to how effective the optimization is. When the weight parameter Q is increased, the agents from the randomized initial positions will follow the circular form much quicker, which essentially increases the tracking performance. On the other hand, it also gives you an insight on what is going on during the optimization process. It is also observed that the pattern is almost formed around $t = 5$ in Figure 3, however it is not true by analyzing Figure 4 5 6 7 8.

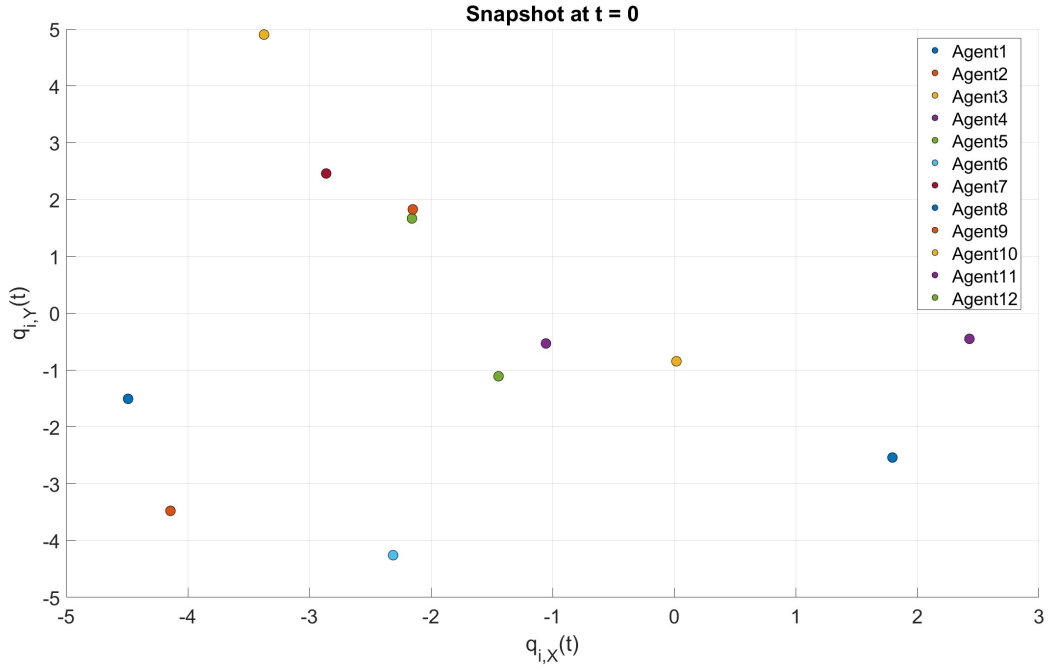

Figure 4: Snapshot at tf = 0.

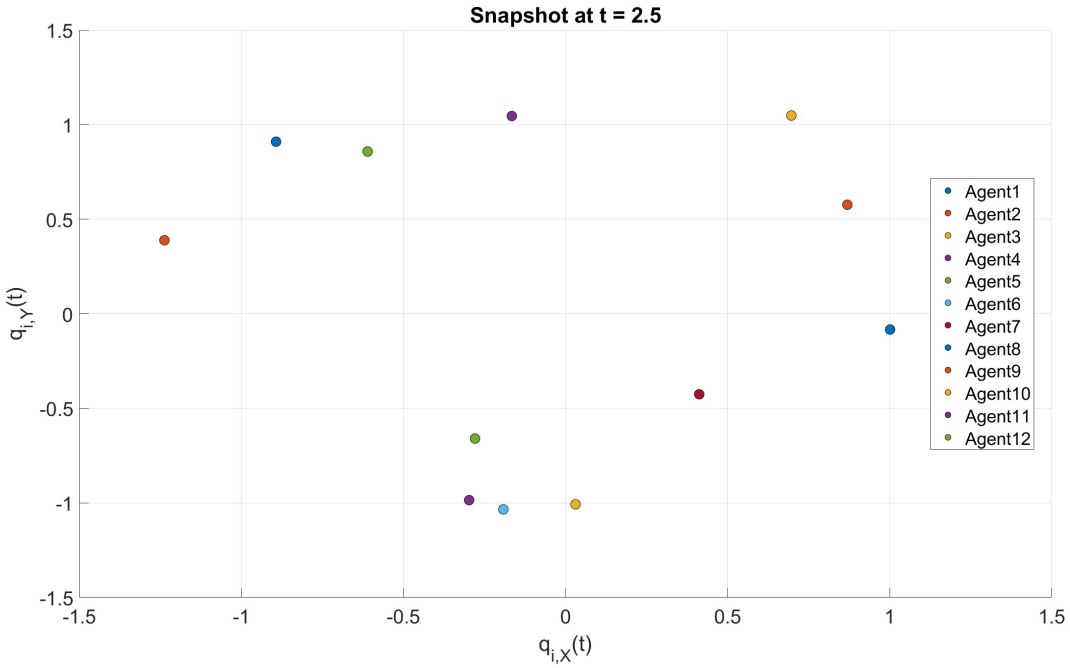

Figure 5: Snapshot at tf = 2.5.

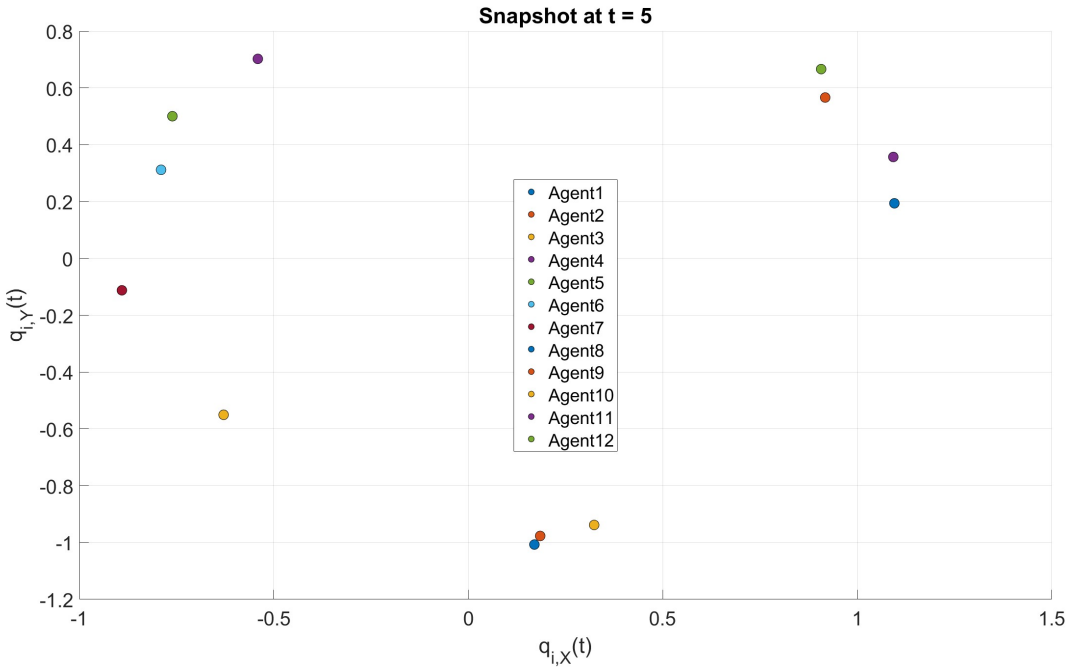

Figure 6: Snapshot at tf = 5.

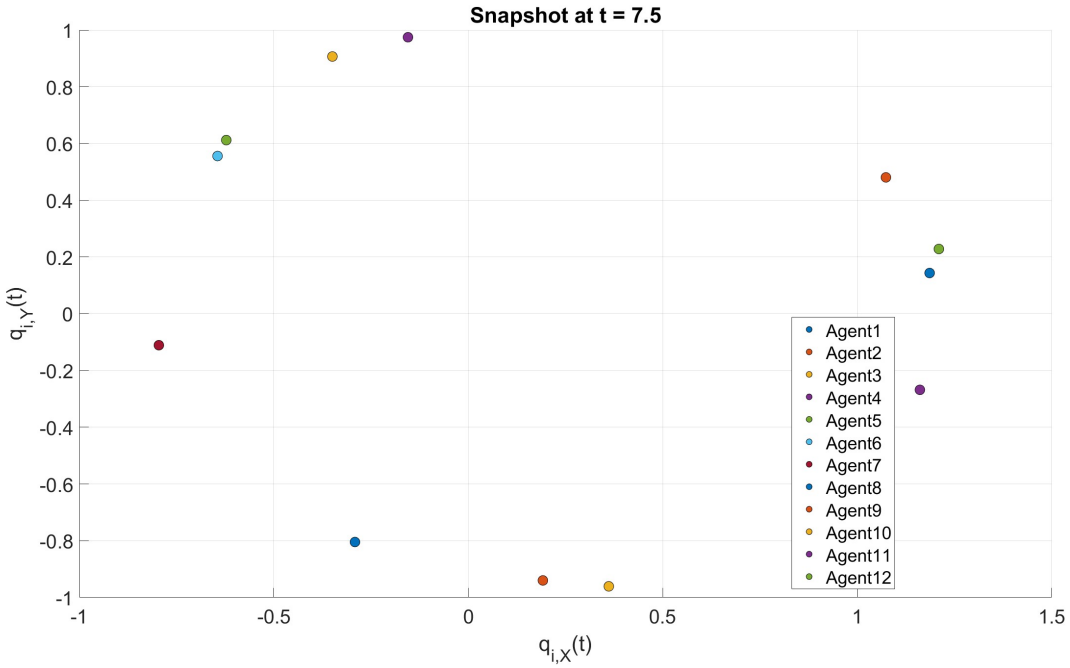

Figure 7: Snapshot at tf = 7.5.

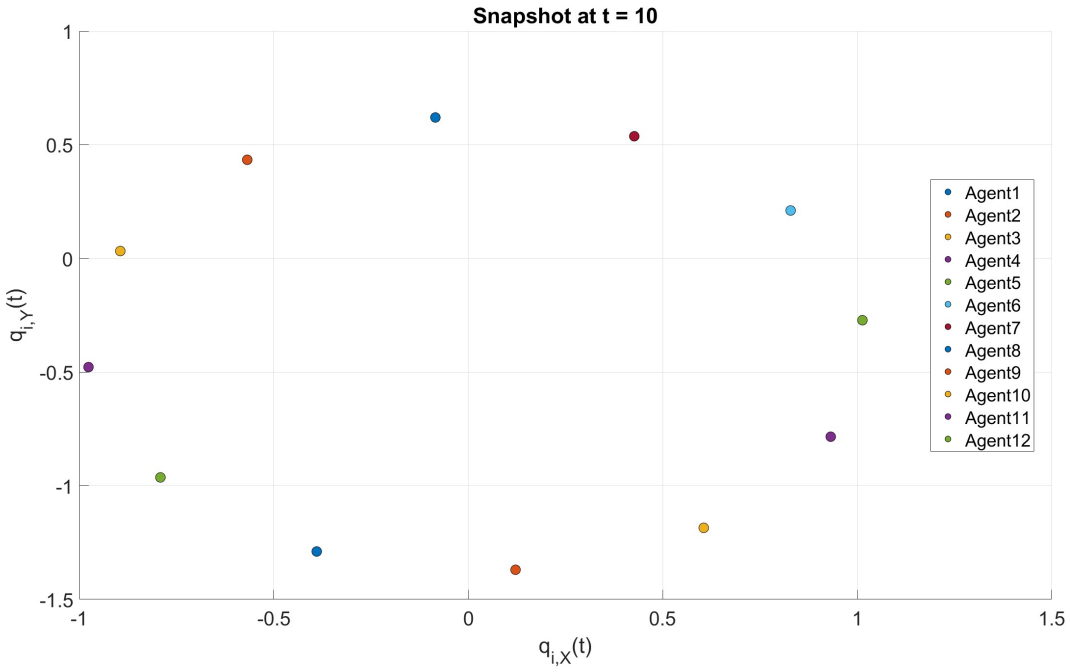

Figure 8: Snapshot at tf = 10.

Figure 9 and 10 showcase examples of the implementation of different formation vectors. In both cases, the visualization of the patterns requires additional simulation time, as these formations involve more intricate shapes. The complexity of the shapes necessitates a longer simulation duration for a clearer representation.

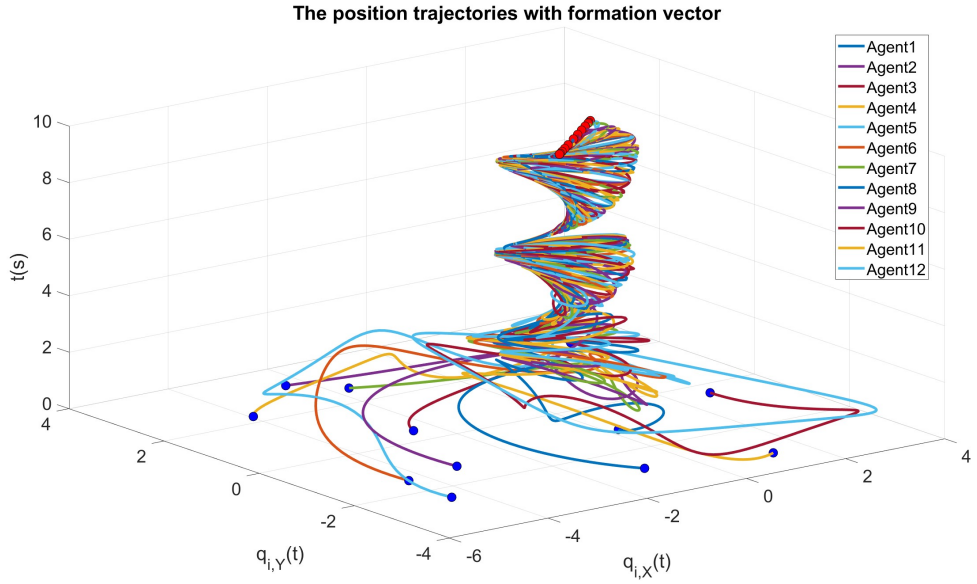

Figure 9: Position Trajectory with formation vectors in a star pattern.

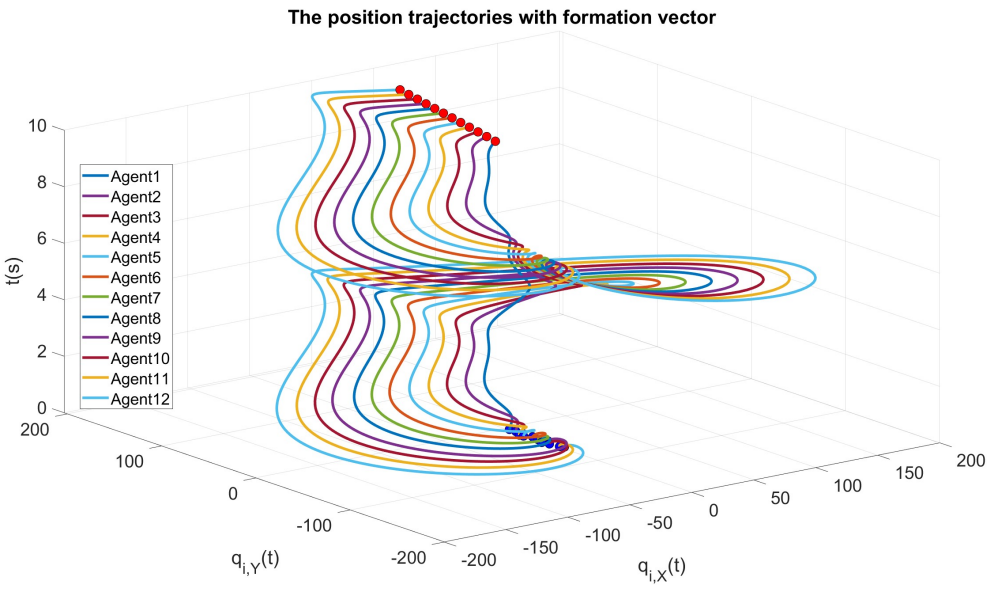

Figure 10: Position Trajectory with formation vectors in a heart pattern.

Figure 3, 9, 10 shows trajectories going in different shapes, where 9 has the trajectories follow the pattern of a star, and 10 has the trajectories follow the pattern of a heart. When the formation vectors are defined, the agents will follow the shape of the vector. It was also observed in this case that having more complex formation vectors made the simulation more time consuming than simple formation vectors.

6 Summary

This paper addresses the issue of the control of multi-agent systems, where the systems of linear time invariant systems. These systems generally have the objective to minimize energy consumption, which can be done by doing optimal tracking and incorporating a time varying aspect.

The incorporation of time varying formation vectors allow it so that the system can be more adaptable to conditions that can be applicable to real-world situations. Hence the adaptability, and the potential advantage of minimizing energy consumption.

The model used in the paper is a Linear Time Invariant system with 12 agents that is simplified due to time constraints. Several formation vectors are tested, which included the shape of a circle, a star, a heart, and a lemniscate. These formation vectors allow for the optimal pattern of trajectory for these agents. Several agents that are interdependent on each other can allow for collision avoidance.

In this project, the objective is to optimize and implement optimal tracking, where the calculus of variations and the Riccati method are used to calculate for the Riccati matrices for the trajectories. The agents are then simulated into two graphs, where one is used with no formation vectors and the other with formation vectors. The simulation was successful and demonstrates the differences and capabilities of formation vectors (in terms of efficiency and adaptability).

7 Future Work

The simulated system in this paper is a simple uncertain LTI system, in contrast to the uncertain EL system applied in [1]. Disturbances in the system are predefined instead of modeled as Gaussian noises. It would be

more compelling to assess the performance of the proposed approach with a more intricate system. For future investigations, exploring a time-varying dynamic system could provide valuable insights into the versatility of the proposed method.

Furthermore, due to time limitations, obtaining the optimal trajectory path proved challenging. To address this limitation, it is essential to visualize the performance of the tracking system by plotting curves of the time-varying formation optimal tracking errors. Ideally, one would anticipate observing a diminishing trend in the error over time. As discussed earlier, the application of time-varying formation vectors and tracking harbors significant potential for solving real-world problems. It would be particularly exciting to explore how this approach can be synergistically combined with other techniques, such as neural networks, in future research endeavors.

In summary, future work should delve into testing the proposed method with more complex and time-varying systems, as well as addressing challenges related to obtaining optimal trajectories. This exploration is expected to open avenues for incorporating the proposed approach with cutting-edge techniques, offering innovative solutions to real-world problems.

References

[1] Piaoyi Su, Jianglong Yu, Yongzhao Hua, Qingdong Li, Xiwang Dong, and Zhang Ren. Distributed time-varying formation optimal tracking for uncertain euler–lagrange systems with time-varying cost functions. *Aerospace Science and Technology*, 132:108019, 2023.

Appendix

```
clc
clear
close all



% Time
num_agents = 12;
t0 = 0; tf= 10; dt = 0.01;
t = t0:dt:tf;

% Init
x_range = [-4, 4]; % x range of the environment
y_range = [-4, 4]; % y range of the environment

initial_x = (x_range(2) - x_range(1)) * rand(1, num_agents) + x_range(1);
initial_y = (y_range(2) - y_range(1)) * rand(1, num_agents) + y_range(1);

x0_all = [initial_x; initial_y];
xf = [0; 0];

% System
A = [0 2; -3 -1];
B = [0; 2];
C = [1 0; 0 0];
Q = 0.5*eye(2); % change this to make tracking better
r = 1;
R = 0.5*r;
F = 10*Q;

%
for a = 1:num_agents
    x0 = x0_all(:, a);
    % Simplified pri with Gaussian noises instead
    noise_std_dev = 0.1; % Standard deviation for noises
    pri = @(t) [0.1; 0.07]*t + [cos(0.02*t) -sin(0.02*t); cos(0.02*t) cos(0.02*t)] *
        normrnd(0, noise_std_dev, 2, 1);

    [tK,K,G] = Psolve(A,B,C,Q,R,F,t0,tf,pri);
    options=odeset('RelTol',1e-10);
    [tx,x]=ode45(@(tx,x) xdiff(tx,x,flag,A,B,R,tK,K,G),[t0 tf],x0,options);
    [m,n]=size(x); % Length of time, x is m
    [mR,nR] = size(R); % number of inputs = mR
    Kt = interp1(tK,K,tx); % interpolate the K matrix
    Gt = interp1(tK,G,tx); % interpolate the G matrix

    u = zeros(m,mR); % initialize the input
    for jj=1:1:m
    u(jj) = -Kt(jj,:)*(x(jj,:))' + inv(R)*B'*(Gt(jj,:))'; %
    end

    % Formation vectors
    % Circle
    hx = cos((2*(a - 1)*pi/num_agents) + 2*tx);
    hy = sin((2*(a - 1)*pi/num_agents) + 2*tx);

    % Star
%     hx = cos(a * tx) .* cos(tx);
%     hy = cos(a * tx) .* sin(tx);

    % Lemniscate of Gerono
%     hx = a * cos(tx);
%     hy = a * sin(2 * tx);

    % Heart
```

```
%     hx = a * (16 * sin(tx).^3);
%     hy = a * (13 * cos(tx) - 5 * cos(2 * tx) - 2 * cos(3 * tx) - cos(4 * tx));

    % Local formation references
    xri = x(:,1) - hx;
    yri = x(:,2) - hy;

    figure(1)
    grid on
    plot3(x(:,1), x(:,2), tx, 'LineWidth', 3)
    hold on
    scatter3(x(1, 1), x(1, 2), tx(1), 100, 'MarkerEdgeColor', 'k', 'MarkerFaceColor','b
        ')
    scatter3(x(end, 1), x(end, 2), tx(end), 100, 'MarkerEdgeColor', 'k', '
        MarkerFaceColor','r')
    set(gca,'FontSize',20)

    figure(2)
    grid on
    plot3(xri, yri, tx, 'LineWidth', 3)
    hold on
    scatter3(xri(1), yri(1), tx(1), 100, 'MarkerEdgeColor', 'k', 'MarkerFaceColor','b')
    scatter3(xri(end), yri(end),tx(end), 100, 'MarkerEdgeColor', 'k', 'MarkerFaceColor
        ','r')
    set(gca,'FontSize',20)

    figure(3)
    grid on
    hold on
    scatter(xri(1), yri(1), 100, 'filled', 'MarkerEdgeColor', 'k')
    set(gca,'FontSize',20)

    figure(4)
    grid on
    hold on
    scatter(xri(round(2*end/5)), yri(round(2*end/5)), 100, 'filled', 'MarkerEdgeColor',
        'k')
    set(gca,'FontSize',20)

    figure(5)
    grid on
    hold on
    scatter(xri(round(3*end/5)), yri(round(3*end/5)), 100, 'filled', 'MarkerEdgeColor',
        'k')
    set(gca,'FontSize',20)

    figure(6)
    grid on
    hold on
    scatter(xri(round(4*end/5)), yri(round(4*end/5)), 100, 'filled', 'MarkerEdgeColor',
        'k')
    set(gca,'FontSize',20)

    figure(7)
    grid on
    hold on
    scatter(xri(round(5*end/5)), yri(round(5*end/5)), 100, 'filled', 'MarkerEdgeColor',
        'k')
    set(gca,'FontSize',20)

end

figure(1)
title('The position trajectories without formation vector')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', '', '', 'Agent2', '', '', 'Agent3', '', '', 'Agent4', '', '', 'Agent5',
```

```
     '', '', 'Agent6', '', '', 'Agent7', '', '', 'Agent8', '', '', 'Agent9', '', '', '
    Agent10', '', '', 'Agent11', '', '', 'Agent12', 'Location', 'Best')

figure(2)
title('The position trajectories with formation vector')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', '', '', 'Agent2', '', '', 'Agent3', '', '', 'Agent4', '', '', 'Agent5',
     '', '', 'Agent6', '', '', 'Agent7', '', '', 'Agent8', '', '', 'Agent9', '', '', '
    Agent10', '', '', 'Agent11', '', '', 'Agent12', 'Location', 'Best')

figure(3)
title('Snapshot at t = 0')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', 'Agent2', 'Agent3', 'Agent4', 'Agent5', 'Agent6', 'Agent7', 'Agent8', '
    Agent9', 'Agent10', 'Agent11', 'Agent12', 'Location', 'Best')

figure(4)
title('Snapshot at t = 2.5')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', 'Agent2', 'Agent3', 'Agent4', 'Agent5', 'Agent6', 'Agent7', 'Agent8', '
    Agent9', 'Agent10', 'Agent11', 'Agent12', 'Location', 'Best')

figure(5)
title('Snapshot at t = 5')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', 'Agent2', 'Agent3', 'Agent4', 'Agent5', 'Agent6', 'Agent7', 'Agent8', '
    Agent9', 'Agent10', 'Agent11', 'Agent12', 'Location', 'Best')

figure(6)
title('Snapshot at t = 7.5')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', 'Agent2', 'Agent3', 'Agent4', 'Agent5', 'Agent6', 'Agent7', 'Agent8', '
    Agent9', 'Agent10', 'Agent11', 'Agent12', 'Location', 'Best')

figure(7)
title('Snapshot at t = 10')
xlabel('q_i_,_X(t)')
ylabel('q_i_,_Y(t)')
zlabel('t(s)')
legend('Agent1', 'Agent2', 'Agent3', 'Agent4', 'Agent5', 'Agent6', 'Agent7', 'Agent8', '
    Agent9', 'Agent10', 'Agent11', 'Agent12', 'Location', 'Best')



%% Helper Functions
function [tK,K,G] = Psolve(A,B,C,Q,R,F,ti,tf,z)
[n,m] = size(B);
NT = n*(n+1)/2;
E = B*inv(R)*B';
options=odeset('RelTol',1e-12);
% Fiding final P in a vector form using F
Ptf = zeros(NT,1);
PMtf = C'*F*C;
k = 1;
for i=1:n
for j=i:n
Ptf(k) = PMtf(i,j);
k = k+1;
```

```
end
end
gtf = [0;0];
% combined vector
PGtf = [Ptf; gtf];
% Solving for P in a vector form PV
[t,PVG]=ode45(@(t,p) PVGdiff(t,p,flag,A,B,C,Q,R,F,tf,z),[tf ti],PGtf,options);
%
%
% PV is in vector form, each row corresponds to row in time t
% flip the PV vector
PVG = flipud(PVG);
% redefine the time vector
t = flipud(t);
% t = -t +(tf)*ones(size(t));
%
%
% computing the gain matrix K(t) as row vector
%
PV = PVG(:,1:NT);
G = PVG(:,NT+1:NT+n);
% plot the riccati matrix terms
% nfig = nfig+1; figure(nfig)
figure(10)
grid on
hold on
subplot(1,2,1);
plot(t,PV,'LineWidth',3)
title('Riccati matrix P(t)')
xlabel('time')
ylabel('P(t)')
legend('P_{11}','P_{12}','P_{22}','Location','best')
set(gca,'FontSize',20)
% plot the riccati matrix terms
% nfig = nfig+1; figure(nfig)
subplot(1,2,2);
plot(t,G,'LineWidth',3)
title('Riccati matrix g(t)')
xlabel('time')
ylabel('g(t)')
legend('g_{1}','g_{2}','Location','best')
set(gca,'FontSize',20)
[mP,nP] = size(PVG);
K = zeros(mP,n);
G = zeros(mP,n);
tK =t;
%
for jj = 1:1:mP
% find P matrix at time jj
Pjj = zeros(n);
for i=1:n
for j=i:n
k = i*n - i*(i-1)/2 - (n-j);
Pjj(i,j) = PVG(jj,k);
Pjj(j,i) = Pjj(i,j);
end
end
% computing the feedback gain matrix
K(jj,:) = inv(R)*B'*Pjj;
G(jj,:) = PVG(jj,NT+1:NT+n);
end
end


%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Function to find the solution to Riccati Eq
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function PVGdiff = PVGdiff(t,p,flag,A,B,C,Q,R,F,tf,z)
```

```
%
% Called by PSolve to compute the Riccati matrix P.
%
% See also PSOLVE.
%
%
% Finding the P matrix PM
[m,n] = size(A);
NT = n*(n+1)/2;
PM = zeros(n);
E = B*inv(R)*B';
%
%
% Finding the P matrix PM
for i=1:n
for j=i:n
k = i*n - i*(i-1)/2 - (n-j);
PM(i,j) = p(k);
PM(j,i) = PM(i,j);
end
end
%
%
% computing P matrix derivative
% Backward in time
PM_diff = -(A'*PM + PM*A -PM*E*PM +C'*Q*C);
%
zt = z(t);
gt = p(NT+1: NT+n,1);
%
Gdiff = -((A -E*PM)'*gt +C'*Q*zt);
%
%
% computing P vector derivative
PVdiff = zeros(NT,1);
k = 1;
for i=1:n
for j=i:n
PVdiff(k) = PM_diff(i,j);
k = k+1;
end
end
PVGdiff = [PVdiff; Gdiff];
end
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
%%%% Function to find the solution to system with optimal control
%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%%
function xdiff = xdiff(t,x,flag,A,B,R,tK,K,G)
%
% solving xdot = AX + Bu
Kt = interp1(tK,K,t);
Gt = interp1(tK,G,t);
ut = -Kt*x +inv(R)*(B')*Gt';
xdiff = A*x +B*ut;
end
```