Delft University of Technology

Estimation and Detection
ET4386

Drone swarms formation control: Project Final Report

Authors:
Taoyue Wang (5914795) and Jiarui Zhou (6017002)

January 7, 2024

1 Formation Control System

1.1 Formation control system without noise modeling

The project focuses on the formation control of multi-agent systems. The formation is described in a graph with $N = 7$ agents and $M = 12$ edges. The generalized Laplacian is given by matrix $\mathbf{L}$. To get edge weights, we need to take the opposite number of each element: $l_{ij} = -\mathbf{L_{ij}}$. To ensure the convergence of the control system, we change the total number of iterations K from 1500 to 50000.

We implement the control system without noise based on algorithm in [1]. In each control update, the first three agents do not change and the last four agents change positions according to the distributed control law and single-integrator update.

The performance measure include the error convergence plot and the trajectory plot. In order to make the performance difference of different estimators more obvious, we plot the error in a logarithmic scale. Figure 1 and figure 2(a) shows the error convergence plot and the trajectory plot without noise. The performance of the control system is perfect.

1.2 Formation control system with noisy observations

Since there is no perfect sensors, uncertainty will be introduced into measurements. In a multi-agent control system, the agent measures its relative position to its neighbors. Measurement noise is introduced on the edge states. We assume the measurement noise is independent and identically distributed. The noise $\mathbf{v}_{ij}$ is assumed Gaussian with zero mean and covariance matrix $\mathbf{R}_{ij} \in \mathbb{S}_+^D$.

We consider the agents can get access to T measurements for each edge $\mathbf{y}_k^{ij} \in \mathbb{R}^{DT\times 1}$ over a time period k. During that period, the edge state $\mathbf{z}_k^{ij} \in \mathbb{R}^{D\times 1}$ is assumed to be constant, so the estimator can use T measurements to estimate the true state. The measurement model can be written in the form of a **general Linear Gaussian model** as follow:

$$\mathbf{y}_k^{ij} = \mathbf{H}\mathbf{z}_k^{ij} + \mathbf{v}_k^{ij} \tag{1}$$

The observation matrix is $\mathbf{H} = \mathbf{1}_T \otimes \mathbf{I}_{D\times D} \in \mathbb{R}^{DT\times D}$. The measurement noise $\mathbf{v}_k^{ij} \in \mathbb{R}^{DT\times 1}$ contains T realizations of Gaussian noise with zero mean and given covariance matrix $\mathbf{R}_{ij}$. After stacking, it is a Gaussian vector with zero mean and covariance matrix $\tilde{\mathbf{R}}_{ij} = \mathbf{I}_{T\times T} \otimes \mathbf{R}_{ij} \in \mathbb{R}^{DT\times DT}$.

Figure 1 and figure 2(b) shows the error convergence plot and the trajectory plot with noise. We can clearly see the performance degrades with noisy observations.

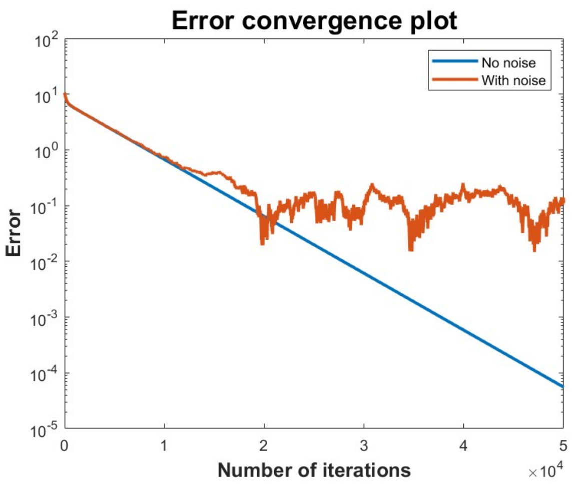
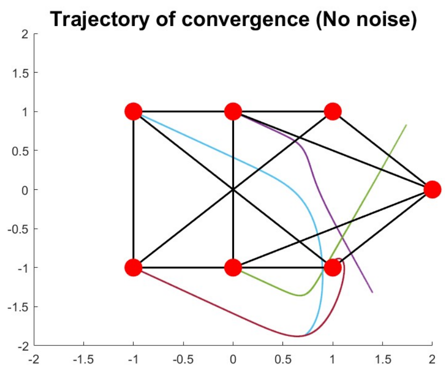
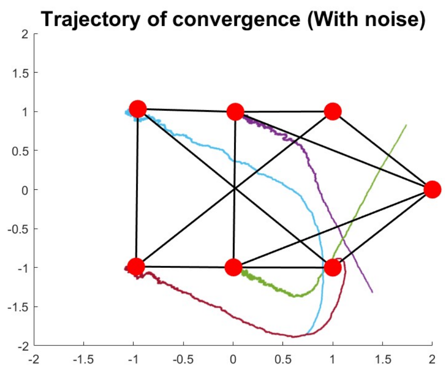


Figure 1: Error convergence plot



(a)



(b)

Figure 2: Trajectory convergence plot

2 Proposed Estimators and Results

In order to deal with the uncertainty introduced to the edge states and optimally estimate those states from noisy measurements, we derive and implement three different estimators. In this section, we will show the general principle, derivation process, and estimation performance of each estimator. We will combine the theory and practice to explain the convergence results of the formation control system.

2.1 Maximum Likelihood Estimator

Based on the measurement model, we can formulate the problem into a deterministic estimation problem. The T measurements for each edge and each time step $\mathbf{y}_k^{ij}$ contain a set of random samples drawn from probability distributions $p(\mathbf{y}_k^{ij}; \mathbf{z}_k^{ij})$. The state for each edge and each time step $\mathbf{z}_k^{ij}$ is deterministic and unknown. We wish to design an estimator to recover the state from multiple measurements.

For general linear Gaussian model, Maximum Likelihood Estimator (MLE) attains the CRLB and is the Minimum Variance Unbiased Estimator (MVUE). MLE finds the parameter which maximizes the likelihood function for fixed measurements. In our problem, finding the maximum is equivalent to solving the minimum of the cost function:

$$J(\mathbf{z}_k^{ij}) = (\mathbf{y}_k^{ij} - \mathbf{H}\mathbf{z}_k^{ij})^T \tilde{\mathbf{R}}^{-1} (\mathbf{y}_k^{ij} - \mathbf{H}\mathbf{z}_k^{ij}) \tag{2}$$

Calculate the gradient and set it to zero. We can derive the expression of MLE estimator. Next, we can plug in the observation matrix H and noise covariance matrix R and simplify the estimator:

$$\frac{\partial J}{\partial \mathbf{z}_k^{ij}} = 2\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H}\mathbf{z}_k^{ij} - 2\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{y}_k^{ij} = 0 \tag{3}$$

$$\hat{\mathbf{z}}_k^{ij} = (\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H})^{-1} \mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{y}_k^{ij} = \frac{1}{T} \mathbf{H}^T \mathbf{y}_k^{ij} \tag{4}$$

This estimator is the same as MVUE. So the estimator is unbiased and the covariance is the inverse of the Fisher Information matrix:

$$\mathbf{C}_{\hat{\mathbf{z}}_{\mathbf{k}}^{\mathbf{ij}}} = (\mathbf{H}^T \tilde{\mathbf{R}}^{-1} \mathbf{H})^{-1} = \frac{1}{T} \mathbf{R} \tag{5}$$

Therefore, for a given noise covariance matrix, as T increases, the MLE estimate has less variance and becomes more stable.

After applying the MLE estimator to the control system, we use the estimated edge states instead of the measurements to substitute in the control law and perform dynamics update. The figure 3 shows the error convergence plot with different number of measurements T. After using the estimator, the error after convergence decreases significantly. It shows that the MLE estimator makes the relative positions more accurate and enhances the stability of the control system. If we increase T, more samples can be used as input to the estimator, leading to more accurate results. However, the fluctuation of the curve is still large because MLE performs instantaneous estimation and doesn't take into account past information.

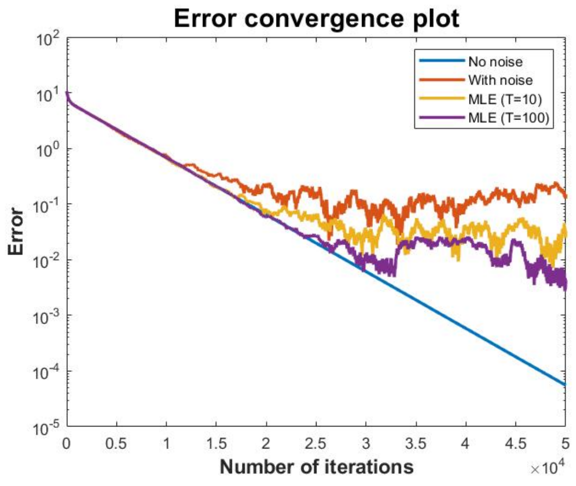
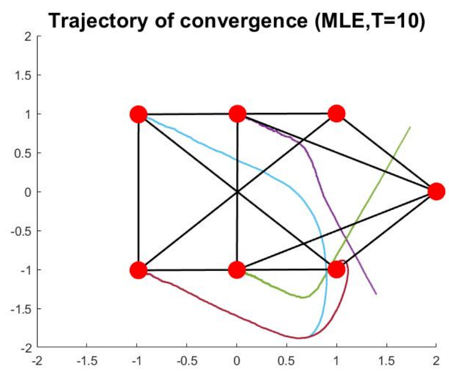
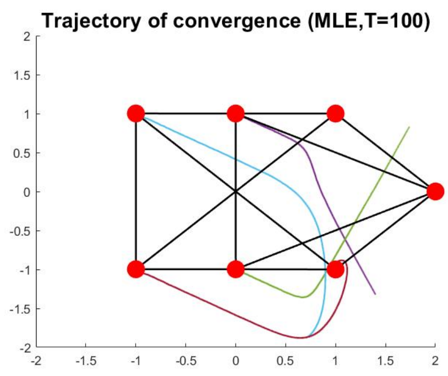


Figure 3: Error convergence plot (with MLE)

Figure 4 shows the trajectory of convergence after using MLE. Compared to the trajectory without estimator, the positions evolve in a smoother way and the final position becomes more accurate. And if we increase T from 10 to 100, the curve becomes even smoother. That's because the estimator removes the noise of edge state.



(a)



(b)

Figure 4: Trajectory convergence plot (with MLE)

2.2 Minimum Mean Square Error Estimator

The MLE estimator doesn't include temporal information of the edge state. We know the relative position between agent i and agent j will not change significantly in a short period of time. So we can use the relative position estimate at time index $k-1$ as a prior for the estimate of the relative position at time index k. This comes to the Bayesian estimation.

We assume the prior for relative position $\mathbf{z}_k^{ij}$ is Gaussian with $\hat{\mathbf{z}}_{k-1}^{ij}$ mean and fixed covariance Σ_{ij}:

$$\mathbf{z}_k^{ij} \sim \mathcal{N}(\hat{\mathbf{z}}_{k-1}^{ij}, \Sigma_{ij}) \tag{6}$$

Since the measurement model is Linear Gaussian model, through minimizing the Bayesian mean square error, we get the Minimum Mean Square Error (MMSE) estimator:

$$\hat{\mathbf{z}}_k^{ij} = \hat{\mathbf{z}}_{k-1}^{ij} + \Sigma_{ij}\mathbf{H}^\mathsf{T}(\mathbf{H}\Sigma_{ij}\mathbf{H}^\mathsf{T} + \tilde{\mathbf{R}})^{-1}(\mathbf{y}_k^{ij} - \mathbf{H}\hat{\mathbf{z}}_{k-1}^{ij}) \tag{7}$$

And the posterior covariance matrix is:

$$\mathbf{C}_{\hat{\mathbf{z}}_\mathbf{k}^\mathbf{ij}} = (\Sigma_{ij}^{-1} - \mathbf{H}^\mathsf{T}\tilde{\mathbf{R}}^{-1}\mathbf{H})^{-1} \tag{8}$$

During the iteration, MMSE estimator uses the previous estimate as the prior mean. The prior covariance Σ_{ij} is determined by experience. If choose too large, the prior knowledge plays a small role and shows no improvement over MLE. If too small, past information will be given a higher weight, thus slowing down the convergence speed. Figure 5 shows the error convergence with respect to different choice of Σ_{ij}. The priors are in the form $\Sigma_{ij} = \sigma_{prior}^2\mathbf{I}_D$. It can be observed that smaller σ_{prior}^2 indeed leads to slower initial convergence. For all the priors, the error becomes close to each other at steady state, even approaching the error of MLE. And we can see from figure 6 that the trajectory is more tortuous with smaller σ_{prior}^2.

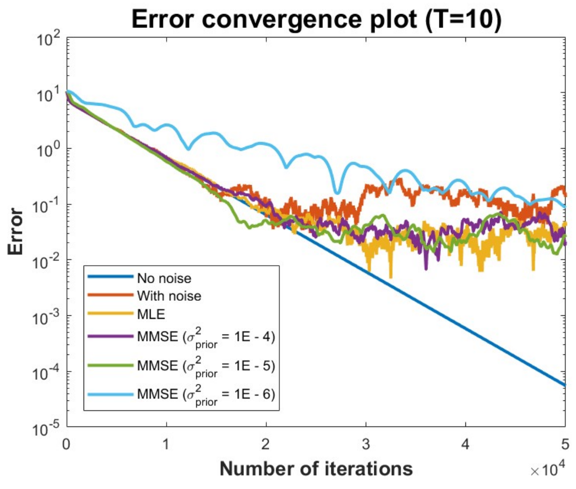
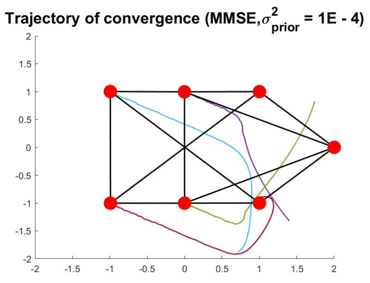
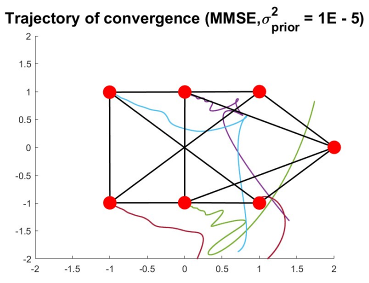


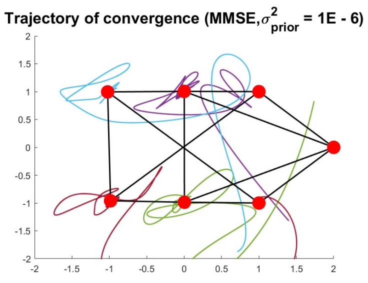
Figure 5: Error convergence plot (with MMSE, T=10)



(a)



(b)



(c)

Figure 6: Trajectory convergence plot (with MMSE, T=10)

By increasing the measurement times T, we get figure 7 and figure 8. Similar to MLE, the error significantly decreases with larger T. The trajectory of convergence also becomes smoother at this situation.

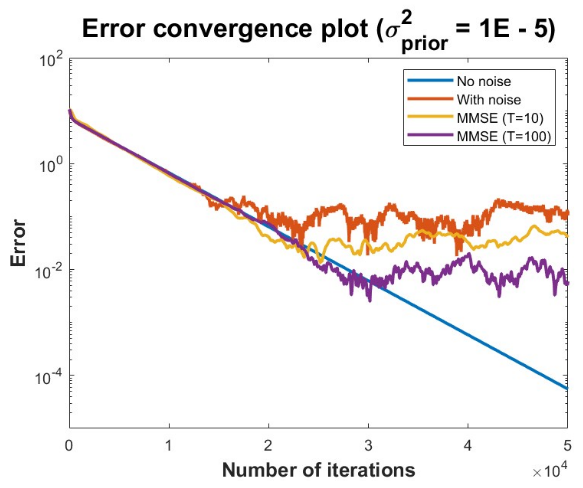


Figure 7: Error convergence plot (with MMSE, $\sigma^2_{prior} = 1E-5$)

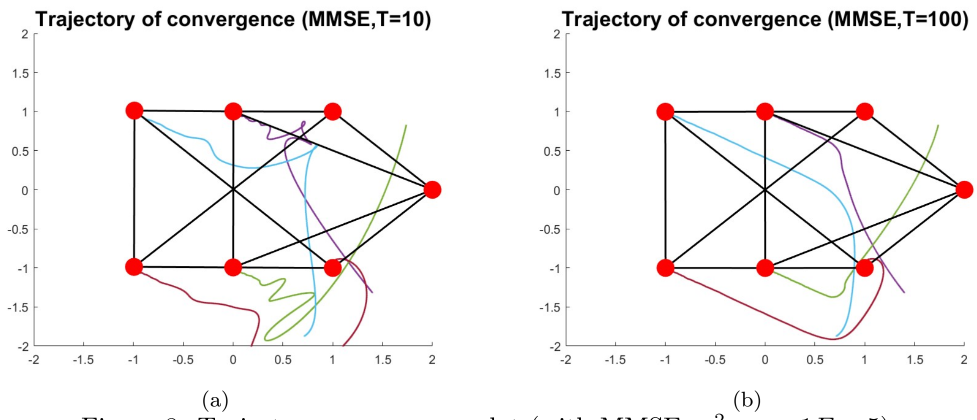


Figure 8: Trajectory convergence plot (with MMSE, $\sigma^2_{prior} = 1E-5$)

2.3 Kalman Filter

The MMSE estimator assumes the unknown edge state to be a Gaussian random variable, whose mean is the previous estimated state, and covariance is predetermined and fixed. But since the system is dynamic, we need a state model to keep track of the prior mean and covariance. The Kalman filter can be thought of a sequential MMSE estimator of a signal embedded in noise, where the signal is characterized by a state model[2].

In our problem, the state model is:

$$\mathbf{z}^{ij}_{k+1} = \mathbf{z}^{ij}_k + dt(\mathbf{u}^i_k - \mathbf{u}^j_k) \tag{9}$$

First, we initialize the expected edge state to be $\hat{\mathbf{z}}^{ij}_{0|0} = \mathbf{0}_{2\times1}$ and the covariance to be $\hat{\mathbf{C}}^{ij}_{0|0} = 2\mathbf{I}_{2\times2}$. Then we iterate over prediction step and update step. The prediction step predicts the edge state and the covariance based on the state model, only using the current state and control input. It is given by the following equations:

$$\hat{\mathbf{z}}^{ij}_{k+1|k} = \hat{\mathbf{z}}^{ij}_{k|k} + dt(\mathbf{u}^i_k - \mathbf{u}^j_k) \tag{10}$$

$$\hat{\mathbf{C}}^{ij}_{k+1|k} = \hat{\mathbf{C}}^{ij}_{k|k} \tag{11}$$

The update step incorporates the new measurements and updates the estimated edge state and covariance. In the following equations, the kalman gain $\mathbf{K}^{ij}_k$ is a correction factor that determines how much emphasis should be given to the new measurement.

$$\mathbf{K}^{ij}_{k+1} = \hat{\mathbf{C}}^{ij}_{k+1|k}\mathbf{H}^T(\mathbf{H}\hat{\mathbf{C}}^{ij}_{k+1|k}\mathbf{H}^T + \tilde{\mathbf{R}})^{-1} \tag{12}$$

$$\hat{\mathbf{z}}^{ij}_{k+1|k+1} = \hat{\mathbf{z}}^{ij}_{k+1|k} + \mathbf{K}^{ij}_{k+1}(\mathbf{y}^{ij}_{k+1} - \mathbf{H}\hat{\mathbf{z}}^{ij}_{k+1|k}) \tag{13}$$

$$\hat{\mathbf{C}}^{ij}_{k+1|k+1} = (\mathbf{I}_{2\times2} - \mathbf{K}^{ij}_{k+1}\mathbf{H})\hat{\mathbf{C}}^{ij}_{k+1|k} \tag{14}$$

The figure 9 and figure 10 show the error convergence plot and trajectory plot. First, the error after convergence is much lower than that of MLE and MMSE. The steady state error of kalman is 0.015 (T=10) and 0.002 (T=100) on the logarithmic scale, while the state error of MLE and MMSE is 0.04 (T=10) and 0.01 (T=100). That's because the kalman filter utilizes both the dynamic change of edge states and the measurement model. It is particularly effective in estimating the state of dynamic systems that evolve with time. By recursively updating the estimate as new measurements become available, it is also easier to adapt to changes in the environment, making it more robust. Second, the convergence process has the least variations or fluctuations. It can converge fast and maintain stable error. The kalman filter deals with the uncertainties in both the state model and measurement model in a statistically optimal manner for Gaussian noise. It is the best choice for our problem.

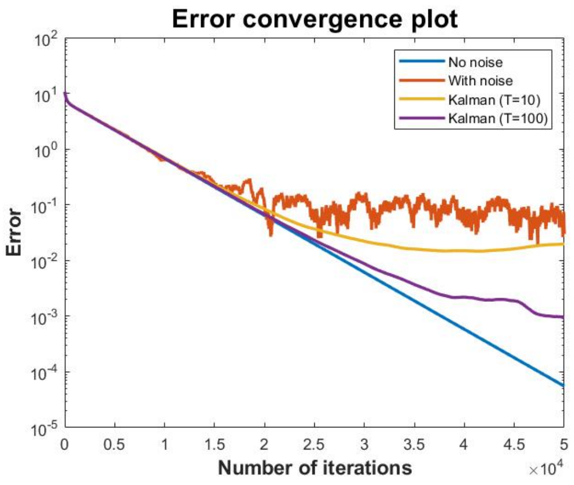
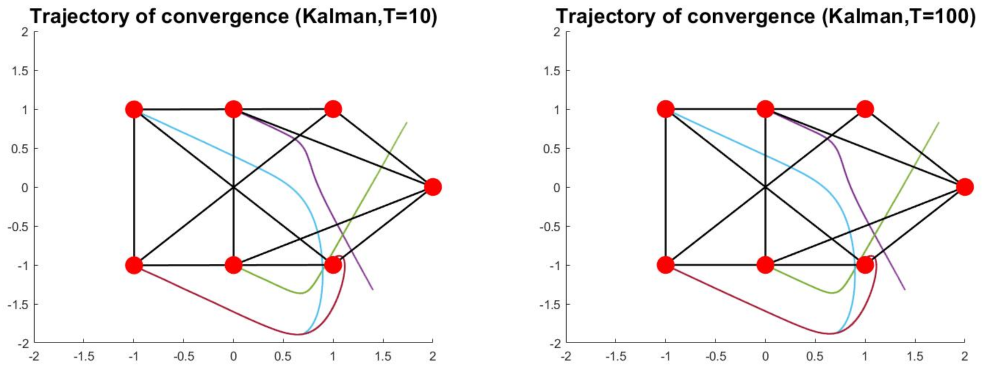


Figure 9: Error convergence plot (with Kalman)



(a) (b)

Figure 10: Trajectory convergence plot (with Kalman)

3 Discussion

3.1 Reasons for the choice of estimators

In the estimation theory, determining the Minimum Variance Unbiased Estimator (MVUE) with Cramér–Rao Lower Bound theorem (CRLB) leads to the optimal estimator. However, this approach requires the knowledge of pdf and complex computations, which is hard for practical problems. Instead, we can design practical sub-optimal estimators, including Maximum Likelihood Estimator (MLE), Best Linear Unbiased Estimator (BLUE), and Least Squares Estimator (LSE). In our problem, the measurement model is a general Linear Gaussian model. In such case, the MLE and BLUE are the same and both lead to the optimal MVU estimator. Because the MLE is the most commonly used approach and is easy to compute, we choose MLE as our first estimator.

Since we know the relative position doesn't change much in a short period, we can use previous information to improve the estimation in next iteration. The use of prior knowledge comes to the Minimum Mean Square Error (MMSE) estimator. According to the formula, the posterior covariance of MMSE is always smaller than the posterior covariance of MLE. When the covariance of the prior becomes very large, the MMSE estimator reduces to the MLE estimator.

The MMSE estimator uses previous estimated state as mean and fixed covariance for edge state Gaussian model. But the system is dynamic and the prior mean and covariance always change. Since we know the control law of the agents, we can make a prediction for next stage based on previous estimated state and predicted movement. And we may decrease the covariance near steady state as the agents are relatively stable. So we choose Kalman filter to solve the limitations of MMSE and increase the accuracy.

3.2 Comparison of three estimators

Figure 11 shows the error convergence plot for all three estimators when T=10. Firstly, in terms of the steady state error, the Kalman filter achieves the best result, while the MLE and MMSE estimators share the similar and larger errors. The Kalman filter treats the edge states as random and uses the state model to model the state change. These assumptions are closest to reality. Secondly, in terms of the fluctuations and randomness upon convergence, the Kalman filter is also the best approach for this problem, which can also be seen from trajectory plots. The Kalman filter can make use of the control law and deal with the Gaussian noise well, making the control system more stable. Thirdly, although Kalman has the best performance, it has the most computation complexity and storage requirement. So in the case of limited resources, MLE can also be a good choice. Moreover, it is worth noting that within a small time window, the error of MMSE is smaller than the noiseless case. We think it is because the randomness of this estimator. More experiments may give better results.

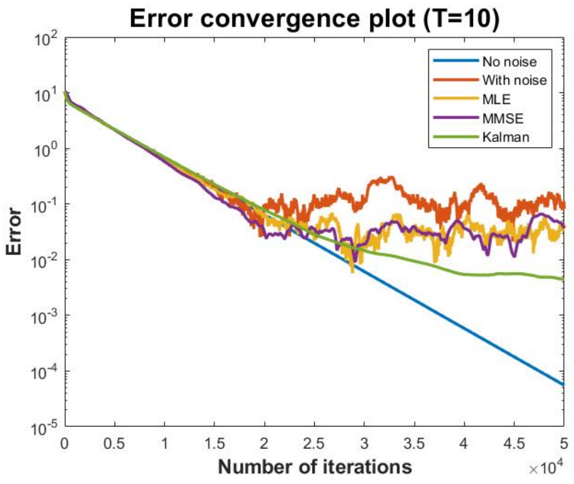


Figure 11: Error convergence plot (3 estimators, T=10)

3.3 Influence of the number of measurements T

From above results, we can see the measurement times T plays an important role in the convergence process. For all three estimators, we can see the error decreases significantly with higher T. This is because with higher T, we can have more samples to be calculated during the update stage. The noise will be removed more efficiently since the covariance of estimators decreases with more data samples.

References

[1] M. van der Marel, “Edge state kalman filtering for distributed formation control systems,” 2021.

[2] K. S. M., *Fundamentals of statistical signal processing: estimation theory.* Prentice-Hall, Inc., 1993.

A Matlab code

```
%%% The main script

close all;
clear;
clc;
%% load data
load("data.mat");
%% plot formation control system without noise and with noise
% parameter initialization
K = 50000;  % total number of iterations
mu = [0 0];
% estimate the states for each iteration
z_record = control_without_noise(N,K,z,L,dt);
z_record_noise = control_with_noise(N,K,z,L,dt,mu,R);
% calculate the estimation error for each iteration
error = calculate_error(z_record,z_star);
error_noise = calculate_error(z_record_noise,z_star);
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,'LineWidth',2);
legend('No noise','With noise');
title("Error convergence plot",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
% plot the trajectories of convergence
plot_trajectory(z_record);
title("Trajectory of convergence (no noise,T=10)",'FontSize',17,'FontWeight','bold')
plot_trajectory(z_record_noise);
title("Trajectory of convergence (with noise,T=10)",'FontSize',17,'FontWeight','bold')
%% plot MLE results
% estimate the states for each iteration
z_record_mle_T10 = control_with_mle(N,K,z,L,dt,mu,R,10);
z_record_mle_T100 = control_with_mle(N,K,z,L,dt,mu,R,100);
% calculate the estimation error for each iteration
error_mle_T10 = calculate_error(z_record_mle_T10,z_star);
error_mle_T100 = calculate_error(z_record_mle_T100,z_star);
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,1:K,error_mle_T10,1:K,error_mle_T100,'LineWidth',2);
legend('No noise','With noise','MLE (T=10)','MLE (T=100)');
title("Error convergence plot",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
% plot the trajectories of convergence
plot_trajectory(z_record_mle_T10);
title("Trajectory of convergence (MLE,T=10)",'FontSize',17,'FontWeight','bold')
plot_trajectory(z_record_mle_T100);
title("Trajectory of convergence (MLE,T=100)",'FontSize',17,'FontWeight','bold')
%% plot MMSE results
%% T=10, change the variance
% parameter initialization
var_prior_5 = 1e-5;  % for MMSE estimator only: the variance of the prior pdf for the ...
    relative position
var_prior_4 = 1e-4;
var_prior_6 = 1e-6;
% estimate the states for each iteration
z_record_mmse_4 = control_with_mmse(N,K,z,L,dt,mu,R,T,var_prior_4);
z_record_mmse_5 = control_with_mmse(N,K,z,L,dt,mu,R,T,var_prior_5);
z_record_mmse_6 = control_with_mmse(N,K,z,L,dt,mu,R,T,var_prior_6);
% calculate the estimation error for each iteration
error_mmse_4 = calculate_error(z_record_mmse_4,z_star);
error_mmse_5 = calculate_error(z_record_mmse_5,z_star);
error_mmse_6 = calculate_error(z_record_mmse_6,z_star);
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,1:K,error_mle_T10,1:K,error_mmse_4,1:K,error_mmse_5,1:K,error_mmse_6,'Line
legend('No noise','With noise','MLE','MMSE (\sigma_{prior}^2 = 1E - 4)','MMSE ...
    (\sigma_{prior}^2 = 1E - 5)','MMSE (\sigma_{prior}^2 = 1E - 6)');
title("Error convergence plot (T=10)",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
% plot the trajectories of convergence
plot_trajectory(z_record_mmse_4);
title("Trajectory of convergence (MMSE,\sigma_{prior}^2 = 1E - ...
    4)",'FontSize',17,'FontWeight','bold')
```

```
plot_trajectory(z_record_mmse_5);
title("Trajectory of convergence (MMSE,\sigma_{prior}^2 = 1E - ...
    5)",'FontSize',17,'FontWeight','bold')
plot_trajectory(z_record_mmse_6);
title("Trajectory of convergence (MMSE,\sigma_{prior}^2 = 1E - ...
    6)",'FontSize',17,'FontWeight','bold')
%% T=10 & 100
% parameter initialization
var_prior = 1e-5;  % for MMSE estimator only: the variance of the prior pdf for the relative ...
    position
% estimate the states for each iteration
z_record_mmse_10 = control_with_mmse(N,K,z,L,dt,mu,R,10,var_prior);
z_record_mmse_100 = control_with_mmse(N,K,z,L,dt,mu,R,100,var_prior);
% calculate the estimation error for each iteration
error_mmse_10 = calculate_error(z_record_mmse_10,z_star);
error_mmse_100 = calculate_error(z_record_mmse_100,z_star);
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,1:K,error_mmse_10,1:K,error_mmse_100,'LineWidth',2);
legend('No noise','With noise','MMSE (T=10)','MMSE (T=100)');
title("Error convergence plot (\sigma_{prior}^2 = 1E - 5)",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
% plot the trajectories of convergence
plot_trajectory(z_record_mmse_10);
title("Trajectory of convergence (MMSE,T=10)",'FontSize',17,'FontWeight','bold')
plot_trajectory(z_record_mmse_100);
title("Trajectory of convergence (MMSE,T=100)",'FontSize',17,'FontWeight','bold')
%% plot Kalman results
% estimate the states for each iteration
z_record_kalman_T10 = control_with_kalman(N,K,z,L,dt,mu,R,10);
z_record_kalman_T100 = control_with_kalman(N,K,z,L,dt,mu,R,100);
% calculate the estimation error for each iteration
error_kalman_T10 = calculate_error(z_record_kalman_T10,z_star);
error_kalman_T100 = calculate_error(z_record_kalman_T100,z_star);
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,1:K,error_kalman_T10,1:K,error_kalman_T100,'LineWidth',2);
legend('No noise','With noise','Kalman (T=10)','Kalman (T=100)');
title("Error convergence plot",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
% plot the trajectories of convergence
plot_trajectory(z_record_kalman_T10);
title("Trajectory of convergence (Kalman,T=10)",'FontSize',17,'FontWeight','bold')
plot_trajectory(z_record_kalman_T100);
title("Trajectory of convergence (Kalman,T=100)",'FontSize',17,'FontWeight','bold')
%% comparison of three estimators(T=10)
% plot the convergence in terms of error
figure;
semilogy(1:K,error,1:K,error_noise,1:K,error_mle_T10,1:K,error_mmse_10,1:K,error_kalman_T10,'LineWidth',2);
legend('No noise','With noise','MLE','MMSE','Kalman');
title("Error convergence plot (T=10)",'FontSize',17,'FontWeight','bold')
xlabel("Number of iterations",'FontSize',13,'FontWeight','bold')
ylabel("Error",'FontSize',13,'FontWeight','bold')
```

```
%%% control_without_noise.m
function z_record = control_without_noise(N,K,z,L,dt)
% Implementation of the formation control system without noise modeling

% Inputs:
% N: number of nodes (agents)
% K: total number of iterations
% z: the initial position vector, shape:(N,2)
% L: the generalized Laplacian matrix of the graph
% dt: the time step of one dynamics update

% Outputs:
% z_record: the position vector of each iteration, shape:(N,2,K)

    k = 1;
    z_record = zeros(N,2,K);
    z_record(:,:,1) = z;
    while k < K
        for i=4:N
            u = 0;
            for j=1:N
                if i == j
```

```
                    u = u;
                else
                    u = u+L(i,j)*(z_record(i,:,k)-z_record(j,:,k)); % control law
                end
            end
            z_record(i,:,k+1) = z_record(i,:,k)+dt*u; % dynamics update
        end
        z_record(1:3,:,k+1) = z_record(1:3,:,k); % keep the first three nodes unchanged
        k = k+1;
    end
end
```

```
%%% control_with_noise.m
function z_record = control_with_noise(N,K,z,L,dt,mu,R)
% Implementation of the formation control system with noise

% Inputs:
% N: number of nodes (agents)
% K: total number of iterations
% z: the initial position vector, shape:(N,2)
% L: the generalized Laplacian matrix of the graph
% dt: the time step of one dynamics update
% mu: the mean of the generated Gaussian noise vector, should be [0 0]
% R: the covariance matrix of the generated Gaussian noise vector

% Outputs:
% z_record: the position vector of each iteration, shape:(N,2,K)

    k = 1;
    z_record = zeros(N,2,K);
    z_record(:,:,1) = z;
    while k < K
        for i=4:N
            u = 0;
            for j=1:N
                if i == j
                    u = u;
                else
                    v = mvnrnd(mu, R, 1);  % draw observation noise for each edge
                    y = z_record(i,:,k)-z_record(j,:,k)+v;  % get noisy measurements
                    u = u+L(i,j) * y;  % control law
                end
            end
            z_record(i,:,k+1) = z_record(i,:,k)+dt*u;  % dynamics update
        end
        z_record(1:3,:,k+1) = z_record(1:3,:,k);  % keep the first three nodes unchanged
        k = k+1;
    end
end
```

```
%%% control_with_mle.m
function z_record = control_with_mle(N,K,z,L,dt,mu,R,T)
% Implementation of the Maximum Likelihood Estimator for edge state
% estimation

% Inputs:
% N: number of nodes (agents)
% K: total number of iterations
% z: the initial position vector, shape:(N,2)
% L: the generalized Laplacian matrix of the graph
% dt: the time step of one dynamics update
% mu: the mean of the generated Gaussian noise vector, should be [0 0]
% R: the covariance matrix of the generated Gaussian noise vector
% T: the number of measurements per control update

% Outputs:
% z_record: the position vector of each iteration, shape:(N,2,K)

    k = 1;
    z_record = zeros(N,2,K);
    z_record(:,:,1) = z;
    mu = repmat(mu,1,T);  % mean of the stacked noise vector
    R_hat = kron(eye(T),R);  % covariance matrix of the stacked noise vector
    H = kron(ones(T,1),eye(2)); % matrix H of the linear gaussian model

    while k < K
```

```
        for i=4:N
            u = 0;
            for j=1:N
                if i == j
                    u = u;
                else
                    v = mvnrnd(mu, R_hat, 1);  % draw observation noise of T realizations for ...
                        each edge
                    y = H*(z_record(i,:,k)-z_record(j,:,k))' + v';  % get T noisy measurements
                    z_hat = H'*y/T;  % MLE estimate of the relative position for each edge
                    u = u+L(i,j)*z_hat';  % control law
                end
            end
            z_record(i,:,k+1) = z_record(i,:,k)+dt*u;  % dynamics update
        end
        z_record(1:3,:,k+1) = z_record(1:3,:,k);  % keep the first three nodes unchanged
        k = k+1;
    end
end
```

```
%%% control_with_mmse.m
function z_record = control_with_mmse(N,K,z,L,dt,mu,R,T,prior)
% Implementation of the Minimum Mean Square Error Estimator for edge state
% estimation

% Inputs:
% N: number of nodes (agents)
% K: total number of iterations
% z: the initial position vector, shape:(N,2)
% L: the generalized Laplacian matrix of the graph
% dt: the time step of one dynamics update
% mu: the mean of the generated Gaussian noise vector, should be [0 0]
% R: the covariance matrix of the generated Gaussian noise vector
% T: the number of measurements per control update
% prior: the variance of the prior pdf for the relative position

% Outputs:
% z_record: the position vector of each iteration, shape:(N,2,K)

    k = 1;
    z_record = zeros(N,2,K);
    z_record(:,:,1) = z;
    mu = repmat(mu,1,T);  % mean of the stacked noise vector
    R_hat = kron(eye(T),R);  % covariance matrix of the stacked noise vector
    H = kron(ones(T,1),eye(2));  % matrix H of the linear gaussian model
    prior_sigma = prior*eye(2);  % the fixed covariance matrix of the prior pdf for the ...
        relative position
    z_hat_record = zeros(N,2,N);  % record the state estimate for all edges

    while k < K
        for i=4:N
            u = 0;
            for j=1:N
                if i == j
                    u = u;
                else
                    v = mvnrnd(mu, R_hat, 1);  % draw observation noise of T realizations for ...
                        each edge
                    y = H*(z_record(i,:,k)-z_record(j,:,k))' + v';   % get T noisy measurements
                    z_hat_pre = z_hat_record(i,:,j)';
                    % MMSE estimate of the relative position for each edge
                    z_hat = z_hat_pre + ...
                        prior_sigma*H'*inv(H*prior_sigma*H'+R_hat)*(y-H*z_hat_pre);
                    u = u+L(i,j)*z_hat';  % control law
                    z_hat_record(i,:,j) = z_hat';
                end
            end
            z_record(i,:,k+1) = z_record(i,:,k)+dt*u;  % dynamics update
        end
        z_record(1:3,:,k+1) = z_record(1:3,:,k);  % keep the first three nodes unchanged
        k = k+1;
    end
end
```

```
%%% control_with_kalman.m
function z_record = control_with_kalman(N,K,z,L,dt,mu,R,T)
```

```
% Implementation of the Edge-based Kalman Filter for edge state estimation

% Inputs:
% N: number of nodes (agents)
% K: total number of iterations
% z: the initial position vector, shape:(N,2)
% L: the generalized Laplacian matrix of the graph
% dt: the time step of one dynamics update
% mu: the mean of the generated Gaussian noise vector, should be [0 0]
% R: the covariance matrix of the generated Gaussian noise vector
% T: the number of measurements per control update

% Outputs:
% z_record: the position vector of each iteration, shape:(N,2,K)

    k = 1;
    z_record = zeros(N,2,K);
    z_record(:,:,1) = z;
    mu = repmat(mu,1,T);  % mean of the stacked noise vector
    R_hat = kron(eye(T),R);  % covariance matrix of the stacked noise vector
    H = kron(ones(T,1),eye(2));  % matrix H of the linear gaussian model
    z_hat_record = zeros(N,2,N);  % record the state estimate for all edges, initialize to zero
    u_record = zeros(N,2);  % record the control input for all nodes
    sigma_record = repmat(2*eye(2),N,1,N); % record the estimated covariance matrix for all edges
                                           % initialize to 2*I

    while k < K
        for i=4:N
            u = 0;
            for j=1:N
                if i == j
                    u = u;
                else
                    % prediction step
                    z_prediction = z_hat_record(i,:,j)+dt*(u_record(i,:)-u_record(j,:)); % ...
                        predict the edge state
                    sigma_prediction = sigma_record(2*i-1:2*i,:,j); % predict the covariance ...
                        matrix
                    % update step
                    v = mvnrnd(mu, R_hat, 1);
                    y = H*(z_record(i,:,k)-z_record(j,:,k))' + v';
                    K_gain = sigma_prediction*H'*inv(H*sigma_prediction*H'+R_hat);  % ...
                        calculate Kalman gain
                    z_new = z_prediction'+K_gain*(y-H*z_prediction'); % update the edge state
                    sigma_new = (eye(2)-K_gain*H)*sigma_prediction;  % update the covariance ...
                        matrix

                    u = u+L(i,j)*z_new';  % control law
                    z_hat_record(i,:,j) = z_new';
                    sigma_record(2*i-1:2*i,:,j) = sigma_new;
                end
            end
            z_record(i,:,k+1) = z_record(i,:,k)+dt*u;  % dynamics update
            u_record(i,:) = u;
        end
        z_record(1:3,:,k+1) = z_record(1:3,:,k);   % keep the first three nodes unchanged
        k = k+1;
    end
end
```

```
%%% calculate_error.m
function e = calculate_error(z_record, z_star)
% Calculate the error of each iteration

% Inputs:
% z_record: the position vector of each iteration, shape:(N,2,K)
% z_star: the target position vector, shape:(N,2)

% Output:
% e: the error of each iteration

    [N,¬,K] = size(z_record);
    e = zeros(1,K);

    for k=1:K
        error = 0;
        for i=4:N
            error = error+norm(z_record(i,:,k)-z_star(i,:));
        end
        e(1,k) = error;
    end
end
```

```
%%% plot_trajectory.m
function plot_trajectory(z_record)
% Plot the trajectories of convergence and the final position

% Input: z_record: the position vector of each iteration, shape:(N,2,K)

    [¬,¬,K] = size(z_record);

    % topology graph
    B = [1,-1,0,0,0,0,0,0,0,0,-1,0,1;
        -1,0,0,0,0,0,1,-1,0,0,0,0,0;
        0,1,-1,0,0,0,0,0,1,0,0,0;
        0,0,0,0,0,1,-1,0,0,1,-1,0;
        0,0,1,-1,0,0,0,0,0,0,1,-1;
        0,0,0,0,1,-1,0,0,-1,0,0,0;
        0,0,0,1,-1,0,0,1,0,0,0,0];
    % dimensions
    [N,M] = size(B);

    % edge set
    edges = mod(reshape(find(B≠0),2,M),N);
    edges(edges==0) = N;

    figure; hold on
    % trajectory formation
    for i=1:N
        %plot(z(i,1),z(i,2),'.','markersize',3);
        plot(reshape(z_record(i,1,:), [1,K]),reshape(z_record(i,2,:), [1,K]),'.','markersize',3);
    end

    %target formation
    z = z_record(:,:,K);
    for i=1:M
        plot(z(edges(:,i),1),z(edges(:,i),2),'k','linewidth',1.5);
    end
    for i=1:N
        plot(z(i,1),z(i,2),'r.','markersize',50);
    end
    axis([-2 2 -2 2]);
    hold off
end
```