MECH 6327 Project Report: Examining Discrete-Time Polytopic Linear Parameter-Varying Systems under threat of malicious actuator and sensor manipulation

Jonas Wagner

***Abstract*—In this project they dynamics of Discrete-Time Polytopic Linear Parameter-Varying (LPV) Systems will be examined. Specifically, various methods for the dual state and parameter estimation will be reproduced with the intent of analyzing effectiveness of these observers against various attacks. Each method performs optimization to minimize the estimation error in various ways while remaining stable and achieving certain performance criteria. Potentially the reachability of the system may be determined for various fault and attack scenarios through the minimization of an ellipsoidal bound.**

I. Polytopic Systems Background

Polytopic LPV system models are essentially a smooth interpolation of a set of LTI submodels constructed using a specified weighting function. This can be looked at as decomposing a system into multiple operating spaces that operate as linear submodels. It is possibile for a Polytopic model to take a complex nonlinear model and redefine it as a time-varying interpolation of multiple linear submodels.

Section references:[1] [1] [3] [4]

A. General Continuous Time Polytopic Model

The simple polyotopic LPV structure can be described by the following weighted linear combination of LTI submodels:

$$\begin{cases} \dot{x}(t) = \sum_{i=1}^{r} \mu_i(\xi(t))\{A_i x(t) + B_i u(t)\} \\ y(t) = \sum_{i=1}^{r} \mu_i(\xi(t)) C_i x(t) \end{cases} \tag{1}$$

with state variable $x \in \Re^n$ common to all r submodels, control input $u \in \Re^p$, output $y \in \Re^q$, weighting function $\mu_i(\cdot)$ and premise variable $\xi(t) \in \Re^w$.

Additionally, the weighting functions $\mu_i(\cdot)$ for each subsystem must satisfy the convex sum constraints:

$$0 \leq \mu_i(\xi),\ \forall i = 1, \dots, r \quad \text{and} \quad \sum_{i=1} \mu_i(\xi) = 1 \tag{2}$$

[1]Each subsection is mostly a summary of sections from these sources but with elaboration and consistent notation.

One notable downside, for our application, is the requirement for $\xi(t)$ to be explicitly known in real-time for the model to function. This requirement is the primary driving factor in investigating this system as when $\xi(t)$ is not explicitly known additional uncertainties now exist in a system that are open for exploitation by an attacker.

B. Discrete Time Polytopic Model

In the DT-Polytopic Model the CT-Polytopic Model, (1), is extended into the discrete time equivalence (either through sampling and zero-order holds or by definition) by the following parameter-varying system:

$$\begin{cases} x_{k+1} &= \sum_{i=1}^{m} \alpha^i (A_i x_k + B_i u_k) \\ y &= C x_k \end{cases} \tag{3}$$

with state variable $x \in \Re^n$, control input $u \in \Re^p$, and output $y \in \Re^q$ common to all of the m submodels. Each submodel is also associated with state matricies A_i and B_i while the output is calculated from the actual state by matrix C.

The scheduling parameter, $\alpha \in \mathcal{A}$ is unknown and time-varying, with $\mathbf{A}$ defined as:

$$\mathcal{A} = \{\alpha \in \Re^m \mid \sum_{i=1}^{m} \alpha^i = 1,\ \alpha^i \geq 0\ \forall\, i \in \{1, 2, \dots, m\}\} \tag{4}$$

In the discrete time case, the unknown scheduling parameter, α, is problematic for when developing a state-estimator, thus a Joint State-Parameter estimator must be used. The discrete nature of the measurements may also prove to be even more problematic if an attack is injected in any discrete measurement.

II. Joint State-Parameter Estimation Problem

The problem of developing joint state and parameter estimator is defined as finding recursive update rules to ensure that state and parameter estimates approach the actual states and parameters. This can be described as finding f_x and f_α such that

$$\begin{cases} \hat{x}_{k+1} &= f_x(\hat{x}_k, \hat{\alpha}_k, \{u_l, y_l\}_{l=\bar{k}_x}) \\ \hat{\alpha}_{k+1} &= f_\alpha(\hat{x}_k, \hat{\alpha}_k, \{u_l, y_l\}_{l=\bar{k}_\alpha}) \end{cases} \tag{5}$$

with $\bar{k}_x, \bar{k}_\alpha < k$ result in $\|x_k - \hat{x}_k\| \to 0$ and $\|\alpha - \hat{\alpha}_k\| \to 0$ as $k \to \infty$.

A. Problem Simplifications/Assumptions [1]

For simplicity, and probably for feasibility reasons, the following assumptions will also be made:

1) $\alpha \in \mathcal{A}$ is constant (or at least slowly time-varying)
2) The estimated system is observable $\forall\ \hat{\alpha} \in \mathcal{A}$, (i.e. $(\sum_{i=1}^{m} \hat{\alpha} A_i, C)$ is an observable pair)
3) A unique solution exists to the joint-estimation problem

$$C\big(qI - \textstyle\sum_{i=1}^{m} \alpha^i A_i x_k\big)^{-1} \sum_{i=1}^{m} \alpha^i B_i \\ = C\big(qI - \textstyle\sum_{i=1}^{m} \hat{\alpha}^i A_i x_k\big)^{-1} \sum_{i=1}^{m} \hat{\alpha}^i B_i \implies \alpha = \hat{\alpha}$$

B. Potential Methods

Three joint state and parameter estimator methods for LPV systems will be developed and tested for the ability of each to react to malicious input and measurement interference. The three estimation methods of interest include:

1) Dual-Estimation (DE) approach is a method that first solves a two step optimization problem for parameters-estimation and then uses a "traditional" robust polytopic observer design for state estimation. [1]
2) Extended Kalman Filter (EKF) using prediction and update steps for the system estimates, but this version does require the assumption of Gaussian noise. [2]
3) Interacting Multiple Model estimation (IMM) method which uses a different kalmen filter for multiple modes and the probability that the system will be a certain mode.[5]

III. Kalmen Filter Derived Methods

A. Extended Kalmen Filter (EKF) Method [2]

The Extended Kalmen Filter (EKF) method relies on a prediction and update steps for both state and parameter estimates by augmenting the state with the parameters creating a nonlinear system.

$$\begin{bmatrix} x_{k+1} \\ \alpha_{k+1} \end{bmatrix} = \begin{bmatrix} \sum_{i=1}^{m} \alpha^i (A_i x_k + B_i u_k) \\ \alpha_k \end{bmatrix} := f(x_k, \alpha_k, u_k) \tag{6}$$

This is then solved through a two-step prediction and estimation process. The prediction step is defined as

$$\hat{x}_{k|k-1} = \sum_{k=1}^{m} \hat{\alpha}^i_{k-1|k-1}\big(A_i \hat{x}_{k-1|k-1} + B_i u_{k-1}\big) \tag{7a}$$

$$\hat{\alpha}_{k|k-1} = \hat{\alpha}_{k|k} \tag{7b}$$

$$P_{k|k-1} = \hat{A}_{k-1} P_{k-1|k-1} \hat{A}^{\mathsf{T}}_{k-1} + Q \tag{7c}$$

with the update steps defined by

$$\begin{bmatrix} \hat{x}_{k|k} \\ \hat{\alpha}_{k|k} \end{bmatrix} = \begin{bmatrix} \hat{x}_{k|k-1} \\ \bar{\alpha}_{k|k-1} \end{bmatrix} + L_k\big(y_k - C\hat{x}_{k|k-1}\big) \tag{8a}$$

$$P_{k|k} = \Big(I - L_k \hat{C}\Big) P_{k|k-1} \tag{8b}$$

and then to restrict $\hat{\alpha}_{k|k} \in \mathcal{A}$, it is determined by

$$\hat{\alpha}_{k|k} = \arg\min_{\alpha \in \mathcal{A}} \big\|\alpha - \bar{\alpha}_{k|k}\big\| \tag{8c}$$

with the augmented estimated parameters defined by

$$L_k = P_{k|k-1} \hat{C}^{\mathsf{T}} \Big(R + \hat{C} P_{k|k-1} \hat{C}^{\mathsf{T}}\Big)^{-1} \tag{9a}$$

$$\hat{A}_k = \left. \frac{\partial f(x_k, \alpha_k, u_k))}{\partial \big[x \quad \alpha\big]^{\mathsf{T}}} \right|_{(\hat{x}_k, \hat{\alpha}_k, u_k)} \\ = \begin{bmatrix} \sum_{i=1}^{m} \hat{\alpha}^i_{k|k} A_i & [A_1 x_{k|k} + B_1 u_k \cdots A_m x_{k|k} + B_m u_k] \\ 0 & I \end{bmatrix} \tag{9b}$$

$$\hat{C} = \begin{bmatrix} C & 0 \end{bmatrix} \tag{9c}$$

In this estimator, Q and R are Positive Semi-definite covariance matrices, traditionally corresponding to process and measurement noise estimates, and can be used as tuning parameters. However, this does require the assumption of Gaussian noise which is not always valid within the actual system.

This method is widely used and will converge when the state and parameter estimates are close enough and any nonlinearities are continuous, but no specific guarantees can be made about the robustness.

IV. DE Method Optimization Problem Definition and Solution [1]

The primary optimization problems to be analyzed are the state and parameter estimation problems within the DE method. This method separates the joint-estimation problem into a state and then a parameter-estimation problem, each of which are determined using convex optimization techniques.

The parameter estimation is done independently using a recursive parameter-estimation method relying solely on the input-output data $\{u_l, y_l\}_{l=1}^{k}$ without any reliance on the state-estimates. This itself is an optimization problem that aims to minimize the prediction error given the parameter estimates.

A. State Estimation Problem and Solution

The system state is estimated using a modified Luenberger observer defined by

$$\hat{x}_{k+1} = \sum_{i=1}^{m} \hat{\alpha}_k^i (A_i \hat{x}_k + B_i u_k + L_i(C\hat{x}_k - y_k)) \quad (10)$$

with $\hat{\alpha}_k \in \mathcal{A}$ being the estimated parameters and L_i being selected to ensure the estimation error, $e_k = x_k - \hat{x}_k$, is stable.

1) State Estimation Error:: The estimation error is defined as

$$e_{k+1} = \sum_{i=1}^{m} \left(\hat{\alpha}_k^i (A_i + L_i C) e_k + \left(\alpha^i - \hat{\alpha}_k^i\right)(A_i x_k + B_i u_k)\right) \quad (11)$$

A solution that ensures e_k is stable is that when the disturbance caused by the deviation of $\hat{\alpha}$, v_k decays to zero (a requirement for the parameter estimation problem), e_k also decays to zero. This is equivelent to ensuring Input-to-State Stability (ISS) of the estimator.

2) Estimator ISS Lyapanov Function:: The estimator can be rewritten as

$$e_{k+1} = \sum_{i=1}^{m} \hat{\alpha}_k^i (A_i + L_i C) e_k + v_k \quad (12)$$

and ISS with respect to the disturbance term, v_k, defined by

$$v_k = \sum_{i=1}^{m} (\alpha^i - \hat{\alpha}_k^i)(A_i x_k + B_i u_k) \quad (13)$$

can be guaranteed by using an ISS Lyapnov function to solve for an appropriate Luenberger gain.
The ISS Lyapnov function is a function $V : \Re^{n+m} \to \Re$, such that

$$V(e, \hat{\alpha}) > 0 \forall e \neq 0, \quad V(0, \hat{\alpha}) = 0,$$

and

$$V(e_{k+1}, \hat{\alpha}_{k+1}) - V(e_k, \hat{\alpha}_k) < -\|e_k\|^2 + \zeta \|v_k\|^2 \quad (14)$$

for all $k \in \{0, \ldots, N\}$.

3) LMI Feasibility Problem for ISS: A theorem provided in [6] states that if there exists matrices $P_i,\ F_i,\ G_i,\ i \in \{1, \ldots, m\}$ and a scaler ζ such that

$$\begin{bmatrix} G_i + G_i^\intercal - P_j & 0 & G_i A_i + F_i C & G_i \\ 0 & I & I & 0 \\ A_i^\intercal G_i^\intercal + C^\intercal F_i^\intercal & I & P_i & 0 \\ G_i^\intercal & 0 & 0 & \zeta I \end{bmatrix} \succ 0 \quad (15)$$

for all $i, j \in \{1, \ldots, m\}$, then for (12) with the Luenberger gains calculated with

$$L_i = G_i^{-1} F_i \quad (16)$$

guarantees ISS with respect to v.[2] This essentially means that the $e_k \to 0$ whenever $v_k \to 0$, or identically, $\hat{x}_k \to x_k$ and $\hat{\alpha}_k \to \alpha_k$, over a finite number of steps.

B. Parameter Estimation Problem and Solution

The DE method estimates the parameters by using only the input-ouput data $\{u_l, y_l\}_{l=0}^{k}$ and completely ignores the estimated states. This is done through the minimization of the prediction error directly through an optimization based on a modified least-squares type algorithm.

1) Parameter Estimate Optimization Problem[1]: The parameter-estimation problem is solved by minimizing the prediction error defined as

$$\hat{\alpha}_k = \arg\min_{\alpha \in \mathcal{A}} \sum_{l=0}^{k} \gamma^{k-l} \|y_k - \varphi_k \nu(\alpha)\|^2 \quad (17a)$$

with a the forgetting factor $0 < \gamma \leq 1$ and

$$\varphi_k = \begin{bmatrix} y_{k-1} & \ldots & y_{k-n_x} & u_{k-1} & \ldots & u_{k-n_x} \end{bmatrix} \quad (17b)$$

$$\nu(\alpha) = \begin{bmatrix} a_1(\alpha) & \ldots & a_{n_x}(\alpha) & b_1(\alpha) & \ldots & b_{n_x}(\alpha) \end{bmatrix}^\intercal \quad (17c)$$

with variables a_i and b_i being the coefficients from the polynomials

$$\sum_{i=1}^{n_x} a_i(\alpha) q^i = \det\left(qI - \sum_{i=1}^{m} \alpha^i A_i\right) \quad (18a)$$

$$\sum_{i=1}^{n_x} b_i(\alpha) q^i = C\text{adj}\left(qI - \sum_{i=1}^{m} \alpha^i A_i\right) \sum_{i=1}^{m} \alpha^i B_i \quad (18b)$$

which are in fact the transfer function coefficients for the system itself, (3).
It is also noteworthy that the optimization problem (17) is a nonlinear (and possibly nonconvex) optimization problem, so instead of solving this directly, a two step approach can be taken to first solve for the coefficients in (18) and then determining the scheduling parameter corresponding to those transfer function coefficients.

2) Step 1: Determine Polynomial Coefficients: Determining the polynomial coefficients is a simple optimization problem consisting of an unconstrained linear least-squares optimization problem given by

$$\hat{\nu}_k = \arg\min_{\nu} \sum_{l=0}^{k} \gamma^{k-l} \|y_k - \varphi_k \nu\|^2 \quad (19)$$

[2]The derivation explained in [6] is interesting (but long and more Optimal Estimation focused in nature) and essentially proves that the ISS Lyapnov function always exists if the solution is feasible.

The solution to this optimization problem can be computed recursively with

$$P_k = \frac{1}{\gamma}P_{k-1} - \frac{1}{\gamma}\varphi_k^{\intercal}(\gamma I + \varphi P_{k-1}\varphi_k^{\intercal})^{-1}\varphi_k P_{k-1} \tag{20a}$$

$$\hat{\nu}_k = \hat{\nu}_{k-1} + P_k\varphi_k^{\intercal}(y_k - \varphi_k\hat{\nu}_{k-1}) \tag{20b}$$

with initial estimates of $\hat{\nu}_0$ and some matrix $P \succ 0$.

3) Step 2: Determine Scheduling Parameter: The second step is to calculate parameter estimates as the solution to the nonlinear optimization problem

$$\hat{\alpha}_k = \arg\min_{\alpha\in\mathcal{A}} (\hat{\nu}_k - \nu(\alpha))^{\intercal} P_k^{-1}(\hat{\nu}_k - \nu(\alpha)) \tag{21}$$

The optimal solution and resultant parameter estimates are equivalent to the optimal solution to the original optimization problem (17). The proof for this is shown in [1] and consists of taking the solution to (21) and going in reverse through the definition and solution of (19) to equate the optimal solution to that of the original optimization problem (17).

V. Numerical Experimentation

A. Simulation Implementation

A generalized Method in Simulink was attempted to be created that would be able to simulate an arbitrary Polytopic System, each of the estimation methods, and introduce attack vectors into the system. This is the future goal to be worked on this summer, but after spending a considerable amount of time learning how to do this, it was abandoned for the short term testing for this project and instead the "toy"/numerical DT model given in [1] was simulated alongside each estimator in a simple MATLAB script for the entirely DT model. These MATLAB scripts can be seen in Appendix A.

B. Example Polyotopic System

A simple DT Polyotopic Model taken from [1] is defined by (3) with the following state-matrices:

$$\begin{aligned} A_1 &= \begin{bmatrix} -0.80 & 0.25 \\ 0.25 & -0.30 \end{bmatrix}, B_1 = \begin{bmatrix} 1.9 \\ 0 \end{bmatrix} \\ A_2 &= \begin{bmatrix} 0.30 & 0.70 \\ 0.70 & 0 \end{bmatrix}, B_2 = \begin{bmatrix} -1 \\ 1.50 \end{bmatrix} \\ A_3 &= \begin{bmatrix} -0.30 & 0.65 \\ 0.55 & 0.10 \end{bmatrix}, B_3 = \begin{bmatrix} 0.30 \\ -2 \end{bmatrix} \\ A_4 &= \begin{bmatrix} 0.55 & = 0.20 \\ -0.40 & -0.30 \end{bmatrix}, B_4 = \begin{bmatrix} -0.60 \\ 0 \end{bmatrix} \end{aligned} \tag{22}$$

with the output matrix $C = \begin{bmatrix} 0 & 1 \end{bmatrix}$, making the dimensions

C. DE Implementation

The Dual Estimate method was implemented as explained in Section IV was then simulated. Untimely it was very difficult to find a solver that was consistent at solving and being robust for a wider array of starting positions for the second step. Eventually it was determined that using Yalmip directly to solve the parameter problem was more effective at converging. (This primarily occurred with the first step as the update for P could result in a negative semi-definite matrix) The 'forgetting factor' from the paper was used:

$$\gamma = 0.9$$

D. EKF Implementation

The Extended Kalmen Filter was implemented as well using the parameters from the paper:

$$R = 0.01 \text{ and } Q = \begin{bmatrix} 0_{n\times n} & \\ & 100 * I_{m\times m} \end{bmatrix}$$

E. Recreated Results

The results from the original paper were recreated in MATLAB (AppendixA). The input was selected as essentially a random PWM signal of strength $u_0 = 0.5$. Specifically, the actual system response is can be seen in Fig. 1. Additionally, the response for the DE and EKF methods can be seen in Fig. 2 and Fig. 3 respectively.

VI. Attack Implementation

The attacks for this system were essentially Gaussian Noise. In the future a more general model (preferably in Simulink) will be developed to test this on more complicated systems, but for now a simple Gaussian noise signal was generated and applied to the measurement each time-step.

VII. Results and Discussion

The results from the original paper were recreated in MATLAB (AppendixA) and a complete collection of the plots can be seen attached in AppendixB. The input was selected as essentially a random PWM signal of strength $u_0 = 0.5$ and multiple power levels of gaussian (attack) noise were tested.

A few important examples to look at are (as expected) the noise increases, the error increases for both of the methods of estimation. One big takaway can be seen in the ability for the EKF method to work better for really small deviations, but untimely, the state estimates are not stable and the error goes to infinity, while the DE method actually guarantees feasibility and convergence.

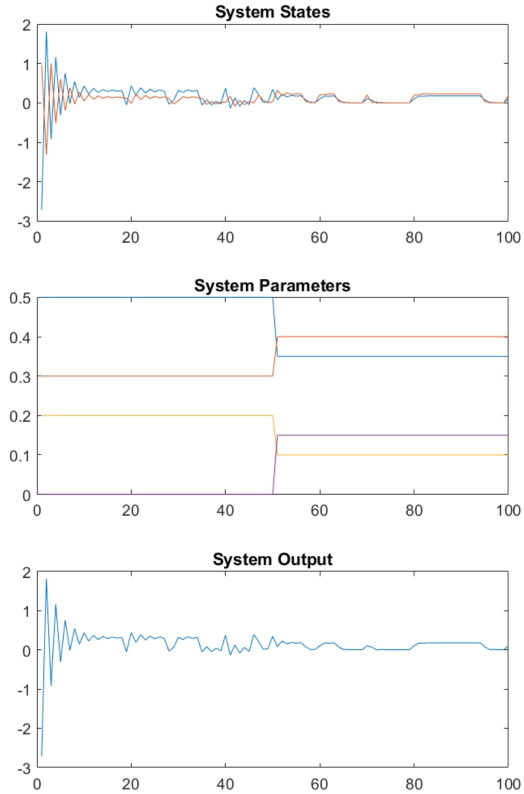

Fig. 1. Recreated System Results

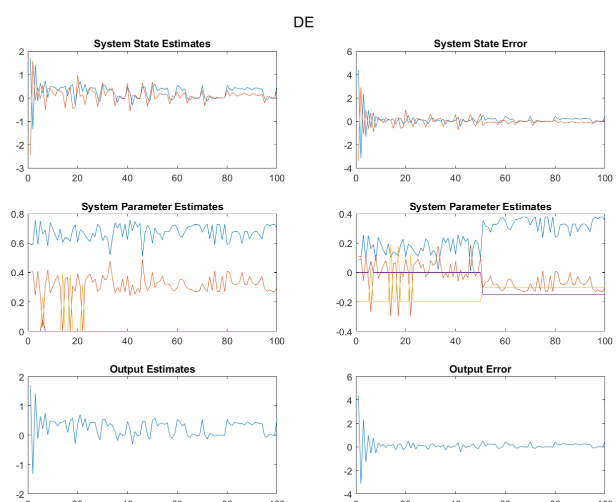

Fig. 2. Recreated DE Simulated Results

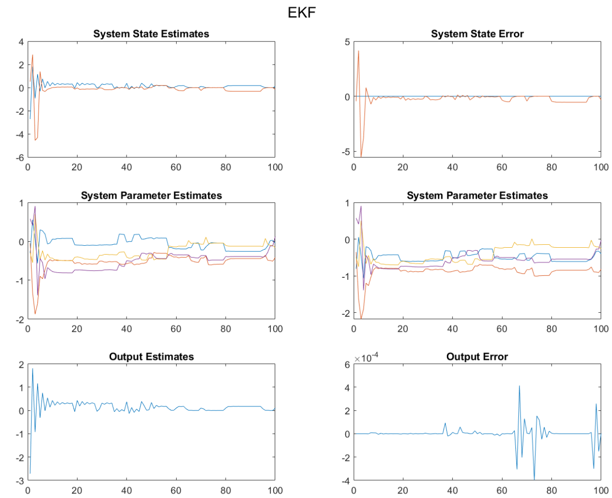

Fig. 3. Recreated EKF Simulated Results

References

[1] H. Beelen and M. Donkers, "Joint state and parameter estimation for discrete-time polytopic linear parameter-varying systems," *IFAC-PapersOnLine*, vol. 50, no. 1, pp. 9778–9783, 2017.

[2] L. Ljung, "Asymptotic behavior of the extended kalman filter as a parameter estimator for linear systems," *IEEE Transactions on Automatic Control*, vol. 24, no. 1, pp. 36–50, 1979.

[3] R. Orjuela, D. Ichalal, B. Marx, D. Maquin, and J. Ragot, "Chapter 9 - polytopic models for observer and fault-tolerant control designs," in *New Trends in Observer-Based Control* (O. Boubaker, Q. Zhu, M. S. Mahmoud, J. Ragot, H. R. Karimi, and J. Dávila, eds.), Emerging Methodologies and Applications in Modelling, pp. 295–335, Academic Press, 2019.

[4] R. Orjuela, B. Marx, J. Ragot, and D. Maquin, "Nonlinear system identification using heterogeneous multiple models," *International Journal of Applied Mathematics and Computer Science*, vol. 23, no. 1, pp. 103–115, 2013.

[5] Y. Bar-Shalom, X. R. Li, and T. Kirubarajan, *Estimation with applications to tracking and navigation: theory algorithms and software*. John Wiley & Sons, 2004.

[6] W. P. M. H. Heemels, J. Daafouz, and G. Millerioux, "Observer-based control of discrete-time lpv systems with uncertain parameters ," *IEEE Transactions on Automatic Control*, vol. 55, no. 9, pp. 2130–2135, 2010.

Appendix

A. MATLAB

All code I wrote for this project can be found on my GitHub repository:
https://github.com/jonaswagner2826/DT_LPV_attack_analysis

Script 1. DT_LPV_sim_script

```
% simulink Generation Scipt
% Jonas Wagner
% 2021-04-08
%
% Important References:
% ----------------------
% https://www.mathworks.com/help/simulink/slref/add_block.html
% https://www.mathworks.com/help/simulink/programmatic-modeling.html
% https://www.mathworks.com/help/simulink/referencelist.html
% https://www.mathworks.com/help/simulink/slref/common-block-parameters.html
% https://www.mathworks.com/help/simulink/slref/block-specific-parameters.html

clear
close all

%% Settings
generateModel = false;
openModel = true;
simulateModel = false;
plotResults = false;

% Name of the simulink model
[cfolder,~,~] = fileparts(mfilename('fullpath'));
subfolder = ''; %include / at end of subfolder
fname = 'DT_LPV_sim';


%% System Definitions

%-------------------------------------------------------------------
% DT_LPV Model Definition
% (https://www.sciencedirect.com/science/article/pii/S2405896317313459)

% DT Settings
k_max = 100;
K = 1:k_max; % Simulation Time

% System Size
n = 2; % States
m = 4; % Subsystems
p = 1; % Inputs
q = 1; % Outputs

% Polyotopic Matrix definitions
A(:,:,1) = [-0.80, 0.25; 0.25,-0.30]; B(:,:,1) = [ 1.90; 0.00];
A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00]; B(:,:,2) = [-1.00; 1.50];
A(:,:,3) = [-0.30, 0.65; 0.55, 0.10]; B(:,:,3) = [ 0.30;-2.00];
```

```
A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30]; B(:,:,4) = [-0.60; 0.00];

A0 = [0,0;0,0]; B0 = [0;0];
A1 = [-0.80, 0.25; 0.25,-0.30]; B1 = [ 1.90; 0.00];
A2 = [ 0.30, 0.70; 0.70, 0.00]; B2 = [-1.00; 1.50];
A3 = [-0.30, 0.65; 0.55, 0.10]; B3 = [ 0.30;-2.00];
A4 = [ 0.55,-0.20;-0.40,-0.30]; B4 = [-0.60; 0.00];

% Output Eq
C = [1, 0]; D = 0;

% Input
u0 = 0.5;
T = 10;
U = zeros(1,k_max);
i = 1;
duty = 100;
for k = 1:k_max
   if i == T
      duty = 100*rand();
      i = 1;
   end
   U(:,k) = square((2*pi*k)/T, duty);
   i = i+1;
end
U = u0 * (U+1)/2;

% Attack (essentially just noise for this small system)
v0 = 0.5;
V = v0 * randn(q,k_max);



% Initial Conditions
x_0 = [0.25; -6.4];
x_hat_0 = [-1.25; 3.4];
alpha_hat_0 = [0.25; 0.25; 0.25; 0.25];



%% Rough Simulation of the System
% Sim Data
X = zeros(n,k_max);
Y = zeros(q, k_max);
Alpha = zeros(m,k_max);

% DE Data
X_hat_DE = zeros(n,k_max);
Alpha_hat_DE = zeros(m,k_max);
P_data_DE = zeros(2*n,2*n,k_max);
Phi = zeros(1,2*n,k_max);
Nu_hat = zeros(2*n,1,k_max);

% EKF Data
```

```
X_hat_EKF = zeros(n,k_max);
Alpha_hat_EKF = zeros(m,k_max);
P_data_EKF = zeros(n+m,n+m,k_max);


% Simulation Initialization
x = x_0;

% DE Initialization
x_hat_DE = x_hat_0;
alpha_hat_DE = alpha_hat_0;
P_DE = randn(2*n); P_DE = P_DE*P_DE.'; %random ??? or this rand one:
P_DE = [1.7552 1.7244 0.7521 0.4716;
      1.7244 4.3679 -0.4500 -0.7000;
      0.7521 -0.4500 3.5146 2.5992;
      0.4716 -0.7000 2.5992 1.9602];
% nu_hat = randn(2*n,1); %random ??? or this rand one:
nu_hat = [0.3; -0.2; 0.1; -0.4];
% phi_hat = randn(2*n,1); %random ??? or this rand one: %[1.2424; -1.0667;
   0.9337; 0.3503];
phi = zeros(1,m);

gamma = 0.9;

% EKF Initialization
x_hat_EKF = x_hat_0;
alpha_hat_EKF = alpha_hat_0;
P_EKF = 1e5 * eye(n+m);% random large P...
Q_EKF = diag([zeros(1,n), 100*ones(1,m)]);
R_EKF = 0.01;

for k = K(1:100)
   % Plant Simulation
   alpha = alpha_traj(k);
   u = U(k);
   x_old = x;
   x = 0;
   for i = 1:m
      x = x + alpha(i) * (A(:,:,i) * x_old + B(:,i) * u);
   end
   y = C*x + D*u + V(:,k);


   % Sim Data
   X(:,k) = x;
   Y(:,k) = y;
   Alpha(:,k) = alpha;

   % DE Method
   if k >= 3
   disp(['DE Method: k = ', num2str(k)])
   P_DE_old = P_DE;
   P_DE = (1/gamma)*P_data_DE(:,:,k-1) - (1/gamma)*(phi'...
      * (gamma + phi * P_data_DE(:,:,k-1) * phi')^(-1)*(phi * P_data_DE(:,:,k
```

```
        -1)));
    if any(eig(inv(P_DE)) < 0)
       warning('INV(P_DE) not PSD')

    end

    end
% phi = [fliplr(Y(:,(k-2):(k-1))), fliplr(U((k-2):(k-1)))];
    phi_old = phi;
    phi(2:n) = phi_old(1:n-1);
    phi(1) = y;
    phi(n+2:2*n) = phi_old(n+1:2*n-1);
    phi(n+1) = u;
    Phi(:,:,k) = phi;


    [x_hat_DE] = est_DE(x_hat_DE, alpha_hat_DE, y, u,...
                                 P_DE, phi, nu_hat,...
                                 gamma, A, B, C);


    % Param. Opt. Problem
    disp('DE Parameter Estimation Started')
    % attempting yalmip
    yalmip('clear')
    alpha_yalmip = sdpvar(m,1);
    nu_alpha = sdpvar(2*n,1);
    param_opt_sum = sdpvar(k,1);

    Constraints = [sum(alpha_yalmip) == 1];
    Constraints = [Constraints, nu_alpha == nuVector(alpha_yalmip)];

    Constraints = [Constraints, nu_alpha == [
    1;
    -((3*alpha_yalmip(2))/10 - (11*alpha_yalmip(1))/10 - alpha_yalmip(3)/5 +
       alpha_yalmip(4)/4 + 1);
    0;
    (alpha_yalmip(2) - (19*alpha_yalmip(1))/10 - (3*alpha_yalmip(3))/10 + (3*
       alpha_yalmip(4))/5 + 1)
    ]];


    for i = 1:m
       Constraints = [Constraints, alpha_yalmip(i) >= 0];
    end
    Constraints = [Constraints, param_opt_sum(1) == norm(Y(:,1) - Phi(1) *
       nu_alpha)^2];

    for i = 2:k
       Constraints = [Constraints, param_opt_sum(i) ...
          == gamma * param_opt_sum(k-1) + norm(Y(:,k) - Phi(k) * nu_alpha)^2];
    end

    Objective = sum(param_opt_sum);
```

```

    options = sdpsettings();

    sol = optimize(Constraints,Objective,options);

    if sol.problem == 0
        % Extract and display value
        alpha_hat_DE = value(alpha_yalmip);
        nu_hat = value(nu_alpha);
    else
        disp('Hmm, something went wrong!');
        sol.info
        yalmiperror(sol.problem)
        error('issue here.....')
    end

    % DE Data
    X_hat_DE(:,k) = x_hat_DE;
    Alpha_hat_DE(:,k) = alpha_hat_DE;
    P_data_DE(:,:,k) = P_DE;
    Nu_hat(:,:,k) = nu_hat;

    % EKF Method
    disp(['EKF Method: k = ', num2str(k)])
    [x_hat_EKF, alpha_hat_EKF, P_EKF] = est_EKF(x_hat_EKF,...
                                    alpha_hat_EKF,...
                                    P_EKF, y, u,...
                                    A, B, C, Q_EKF, R_EKF);
    % EKF Data
    X_hat_EKF(:,k) = x_hat_EKF;
    Alpha_hat_EKF(:,k) = alpha_hat_EKF;
    P_data_EKF(:,:,k) = P_EKF;

end


X = X(:,1:k);
Alpha = Alpha(:,1:k);
Y = Y(:,1:k);
X_hat_DE = X_hat_DE(:,1:k);
Alpha_hat_DE = Alpha_hat_DE(:,1:k);
X_hat_EKF = X_hat_EKF(:,1:k);
Alpha_hat_EKF = Alpha_hat_EKF(:,1:k);

figure('position', [0,0,500,750])
sgtitle(['Attack Power: ',num2str(v0)])
subplot(3,1,1)
plot(X')
title('System States')
subplot(3,1,2)
plot(Alpha')
title('System Parameters')
subplot(3,1,3)
plot(Y')
```

```
title('System Output')


figure('position', [0,0,1000,750])
sgtitle(['DE: Attack Power = ',num2str(v0)])
ax1 = subplot(3,2,1);
plot(X_hat_DE')
title('System State Estimates')
ax2 = subplot(3,2,3);
plot(Alpha_hat_DE')
title('System Parameter Estimates')
ax3 = subplot(3,2,5);
plot((C*X_hat_DE)')
title('Output Estimates')
ax4 = subplot(3,2,2);
plot(X_hat_DE'-X')
title('System State Error')
ax5 = subplot(3,2,4);
plot(Alpha_hat_DE'- Alpha')
title('System Parameter Estimates')
ax6 = subplot(3,2,6);
plot((C*X_hat_DE)'-Y')
title('Output Error')

figure('position', [0,0,1000,750])
sgtitle(['EKF: ','Attack Power = ',num2str(v0)])
ax1 = subplot(3,2,1);
plot(X_hat_EKF')
title('System State Estimates')
ax2 = subplot(3,2,3);
plot(Alpha_hat_EKF')
title('System Parameter Estimates')
ax3 = subplot(3,2,5);
plot((C*X_hat_EKF)')
title('Output Estimates')
ax4 = subplot(3,2,2);
plot(X_hat_EKF'-X')
title('System State Error')
ax5 = subplot(3,2,4);
plot(Alpha_hat_EKF'- Alpha')
title('System Parameter Estimates')
ax6 = subplot(3,2,6);
plot((C*X_hat_EKF)'-Y')
title('Output Error')
```

Script 2. alpha_traj

```
function [p] = alpha_traj(k)
   %alpha_traj Function returns the parameter as a function of k
   k_limit = 50;
   p1 = [0.50; 0.30; 0.20; 0.00];
   p2 = [0.35; 0.40; 0.10; 0.15];
   if (k > k_limit)
      p = p2;
   else
      p = p1;
   end
end
```

Script 3. est_DE

```
function [x_hat, alpha_hat, nu_hat] ...
                              = est_DE(x_hat, alpha_hat, y, u,...
                                P, phi, nu_hat,...
                                gamma, A, B, C)
   %est_JSPE Function performs a single iteration of the joint state and
   %parameter estimator

   arguments
      x_hat %x_hat_{k-1} (n,1)
      alpha_hat %\alpha_hat_{k-1} (m,1)
      y %y_k (1,1)
      u %u_k (1,1)
      P %P_k (2n,2n)
      phi %phi_k (2n,1)
      nu_hat %nu_hat_k (1,2n)
      gamma = 0.9
      A = string(-1)
      B = string(-1)
      C = [1, 0]
% nu_alpha = -1
   end

   %% System Definition
   if string(A) == string(-1)
      clear A
      A(:,:,1) = [-0.80, 0.25; 0.25,-0.30];
      A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00];
      A(:,:,3) = [-0.30, 0.65; 0.55, 0.10];
      A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30];
   end
   if string(B) == string(-1)
      clear B

      B(:,:,1) = [ 1.90; 0.00];
      B(:,:,2) = [-1.00; 1.50];
      B(:,:,3) = [ 0.30;-2.00];
      B(:,:,4) = [-0.60; 0.00];
   end

   % System Dimenstions
   n = size(A,1);
   m = size(A,3);
   p = size(B,2);
   q = size(C,1);

   %% State Estimation
   % CVX Feasability Problem on LMI
   disp('DE State Estimation Started')
   tol = 1e-6;
   cvx_begin sdp quiet
      variable P_cvx(n,n,m) symmetric
      variable F(n,q,m)
      variable G(n,n,m)
```

```
        variable zeta
        subject to
            for i = 1:m
                for j = 1:m
                    [G(:,:,i) + G(:,:,i)' - P_cvx(:,:,j), zeros(n), G(:,:,i)*A(:,:,
                        i) + F(:,:,i)*C, G(:,:,i);
                     zeros(n), eye(n), eye(n), zeros(n);
                     A(:,:,i)'*G(:,:,i)' + C'*F(:,:,i)', eye(n), P_cvx(:,:,i),
                         zeros(n);
                     G(:,:,i)', zeros(n), zeros(n), zeta*eye(n)] >= tol*eye(4*n);
                end
            end
            zeta >= 1;
    cvx_end

    % Gain Result Calc
    for i = 1:m
        L(:,:,i) = inv(G(:,:,i))*F(:,:,i);
    end

    % State Estimate Update
    x_hat_old = x_hat; % x_hat_{k-1}
    x_hat = 0;
    for i = 1:m
        x_hat = x_hat + alpha_hat(i)*(A(:,:,i)*x_hat_old + B(:,i)*u)...
            + L(:,:,i)*(C*x_hat_old - y);
    end
    %Note: x_hat = x_hat k

    %% Parameter Estimation

% % Phi Matrix Definitions
% if string(nu_alpha) == string(-1)
% Alpha_sym = sym('alpha',[m,1]);
% A_bar = zeros(n);
% B_bar = zeros(n,p);
% for i = 1:m
% A_bar = A_bar + Alpha_sym(i) * A(:,:,i);
% B_bar = B_bar + Alpha_sym(i) * B(:,:,i);
% end
% a_coeff = charpoly(A_bar);
% temp = sym('temp');
% b_coeff = charpoly(C*adjoint(temp*eye(n) - A_bar)*B_bar);
% if size(a_coeff,2) <= n
% a_coeff = [zeros(1,n+1-size(a_coeff,2)),a_coeff];
% end
% if size(b_coeff,2) <= n
% b_coeff = [zeros(1,n+1-size(b_coeff,2)),b_coeff];
% end
% nu_alpha = matlabFunction([fliplr(a_coeff(1:n)), fliplr(b_coeff(1:n))]');
% end


% % Est. Opt. Problem
```

```
% disp('DE Parameter Estimation Started')
% % attempting yalmip
% yalmip('clear')
% Alpha = sdpvar(m,1);
% % nu_alpha = sdpvar(2*n,1);
%
% Constraints = [sum(Alpha) == 1];
% % Constraints = [Constraints, nu_alpha == nuVector(Alpha)];
% for i = 1:m
% Constraints = [Constraints, Alpha(i) >= 0];
% end
%
% Objective = (nu_hat - nu_alpha)' * inv(P) * (nu_hat - nu_alpha);
%
% options = sdpsettings();%'debug',1,'verbose',1)%,'solver','quadprog','quadprog.maxiter',100);
%
% sol = optimize(Constraints,Objective,options)
%
%
% % cvx_begin quiet
% % variable alpha_cvx(m,1)
% % nu_alpha = nu_alpha(alpha_cvx);
% %
% % % Minimize This
% % minimize((nu_hat - nu_alpha)' * inv(P) * (nu_hat - nu_alpha))
% % subject to
% % sum = 0;
% % for i = 1:m
% % alpha_cvx(i) >= 0;
% % sum = sum + alpha_cvx(i);
% % end
% % sum == 1;
% % cvx_end
% if sol.problem == 0
% % Extract and display value
% alpha_hat = value(Alpha) %alpha_cvx;
% nu_hat = value(nu_alpha)
% else
% disp('Hmm, something went wrong!');
% sol.info
% yalmiperror(sol.problem)
% error('issue here.....')
% end



% % k step
% P_old = P;
% phi_old = phi;
% nu_hat_old = nu_hat;
%
%
% % % k+1 step
```

```
% % phi(2:n) = phi_old(1:n-1);
% % phi(1) = y;
% % phi(n+2:2*n) = phi_old(n+1:2*n-1);
% % phi(n+1) = u;
%
% % Recursive Update
% P = (1/gamma)*P_old - (1/gamma)*(phi'...
% * inv(gamma + phi * P_old * phi') * phi * P_old);
% nu_hat = nu_hat_old + P * phi' * (y - phi*nu_hat_old);





end
```

Script 4. est_EKF

```
function [x_hat, alpha_hat, P] = est_EKF(x_hat, alpha_hat, P, y, u,...
                                A, B, C, Q, R)
    %est_EKF Function performs a single iteration of the parameter and
    %state estimation using an EKF

    arguments
        x_hat
        alpha_hat
        P
        y
        u
        A = string(-1)
        B = string(-1)
        C = [1, 0]
        Q = string(-1)
        R = string(-1)
    end

    % System Matrices
    if string(A) == string(-1)
        clear A
        A(:,:,1) = [-0.80, 0.25; 0.25,-0.30];
        A(:,:,2) = [ 0.30, 0.70; 0.70, 0.00];
        A(:,:,3) = [-0.30, 0.65; 0.55, 0.10];
        A(:,:,4) = [ 0.55,-0.20;-0.40,-0.30];
    end
    if string(B) == string(-1)
        clear B
        B(:,:,1) = [ 1.90; 0.00];
        B(:,:,2) = [-1.00; 1.50];
        B(:,:,3) = [ 0.30;-2.00];
        B(:,:,4) = [-0.60; 0.00];
    end

    % System Dimenstion Matrices
    n = size(A,1);
    m = size(A,3);
    p = size(B,1);
    q = size(C,1);


    % EKF Covariance Matrices
    if string(Q) == string(-1)
        clear Q
        Q = diag([zeros(1,n), 100*ones(1,m)]);
    end
    if string(R) == string(-1)
        clear R
        R = 0.01;
    end

    % A_hat (k-1) calc
    A_hat = diag([zeros(1,n), ones(1,m)]);
```

```
    for i = 1:m
        A_hat(1:n,1:n) = A_hat(1:n,1:n) + alpha_hat(i) * A(:,:,i);
        A_hat(1:n,n+i) = A(:,:,i) * x_hat + B(:,:,i) * u;
    end


    % Preditiction Step
    x_hat_pre = 0;
    for i = 1:m
        x_hat_pre = x_hat_pre + alpha_hat(i) * (A(:,:,i) * x_hat + B(:,:,i) * u);
    end
    alpha_hat_pre = alpha_hat;
    P_pre = A_hat * P * A_hat' + Q;

    % L (k) calc
    C_hat = [C, zeros(1,m)];
    L = P_pre * C_hat' * inv(R + C_hat * P_pre * C_hat');

    % Update Step
    x_alpha_pre = [x_hat_pre; alpha_hat_pre];
    x_alpha_post = x_alpha_pre + L * (y - C * x_hat_pre);
    P_post = (eye(n+m) - L * C_hat) * P_pre;

    % Estimates
    x_hat = x_alpha_post(1:n);
    alpha_hat = x_alpha_post(n+1:n+m);
    P = P_post;
end
```

B. Complete Set of Results

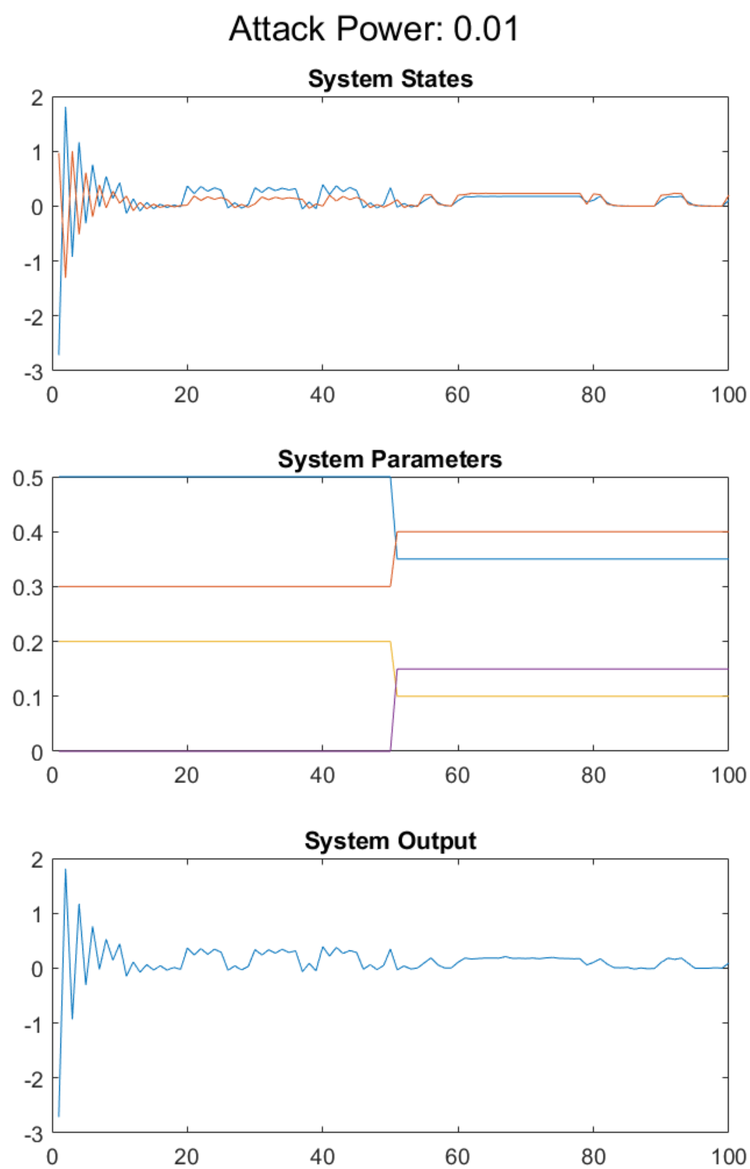

Fig. 4. $v_0 = 0.01$ System Results

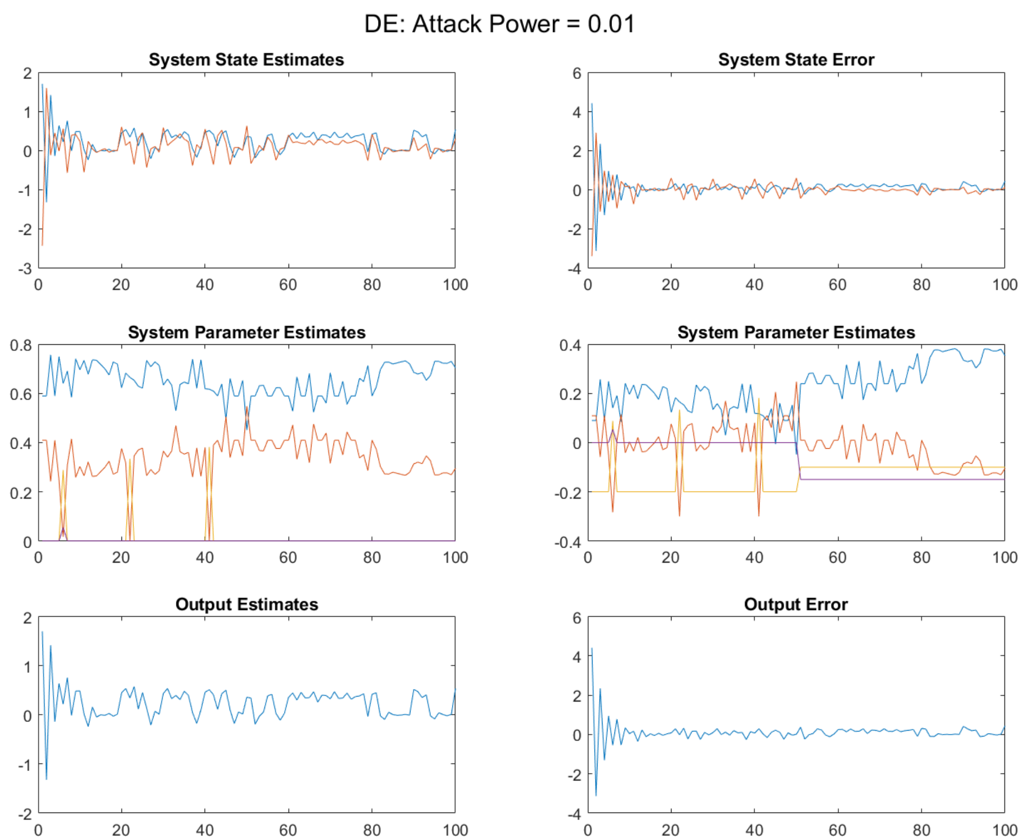

Fig. 5. DE with $v_0 = 0.01$ Simulated Results

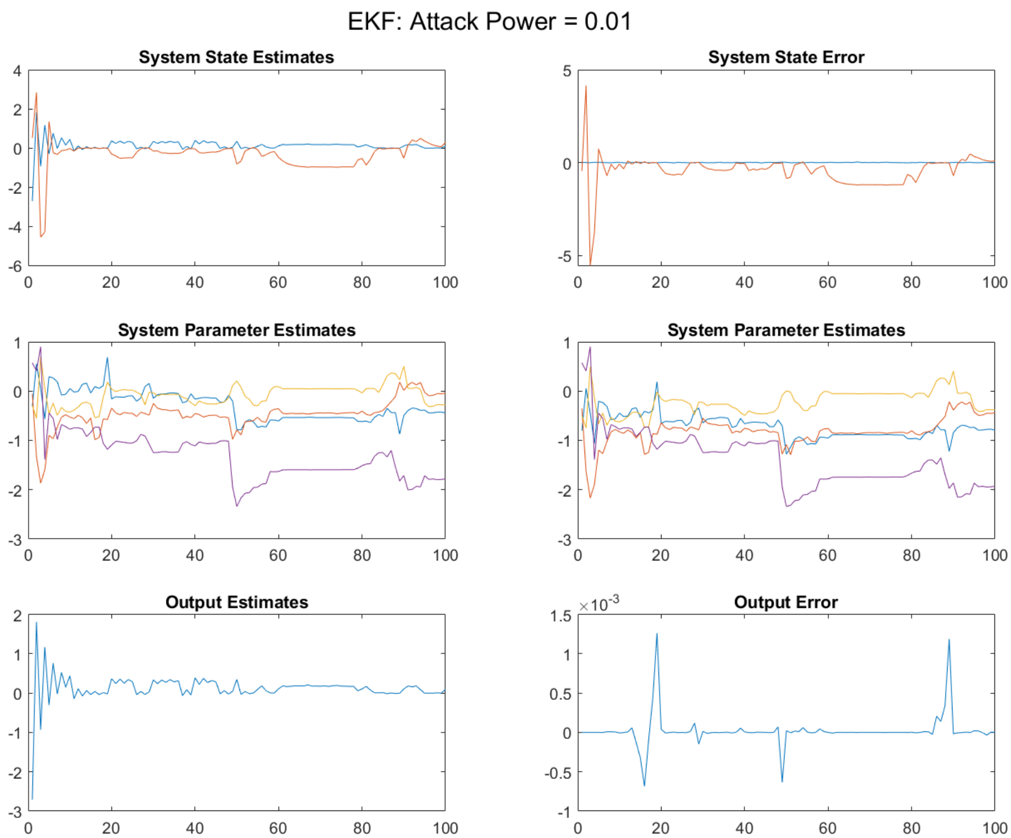

Fig. 6. EKF with $v_0 = 0.01$ Simulated Results

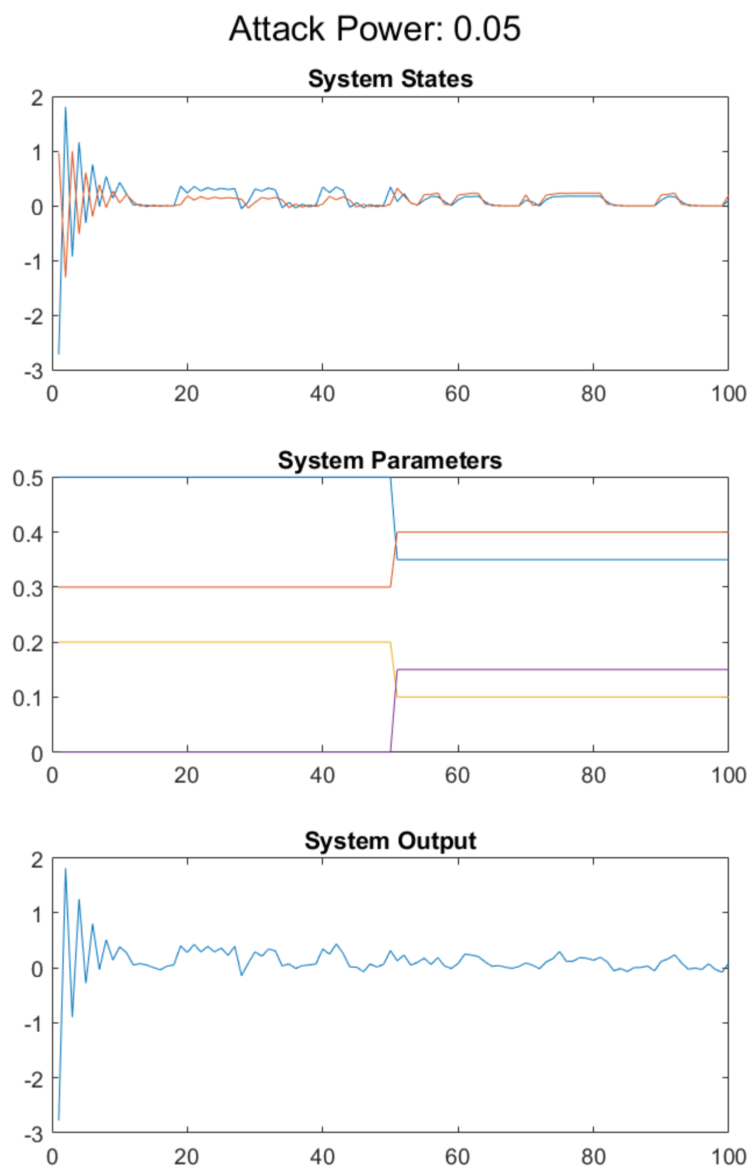

Fig. 7. $v_0 = 0.05$ System Results

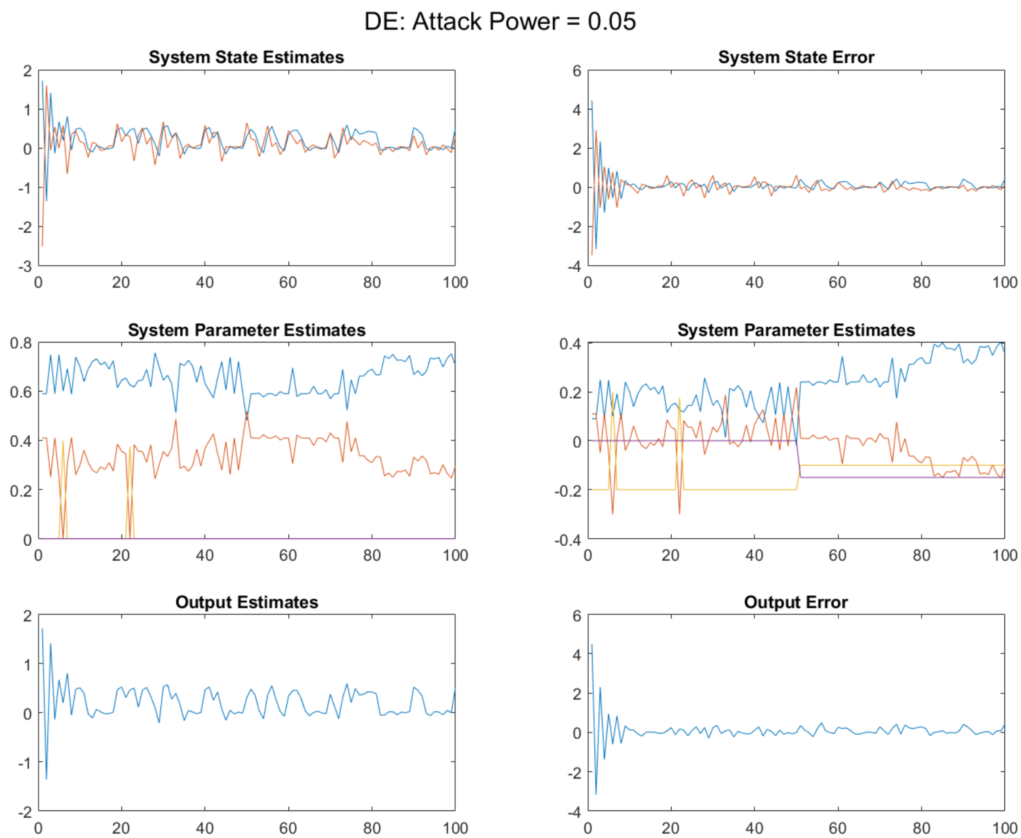

Fig. 8. DE with $v_0 = 0.05$ Simulated Results

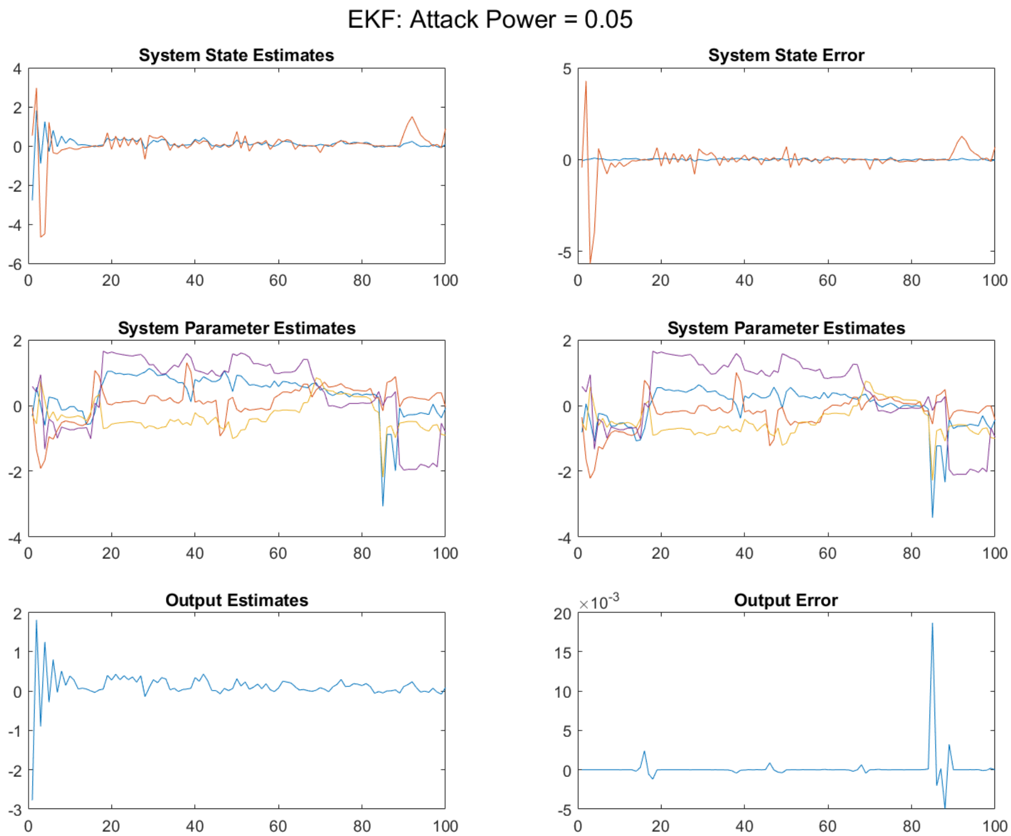

Fig. 9. EKF with $v_0 = 0.05$ Simulated Results

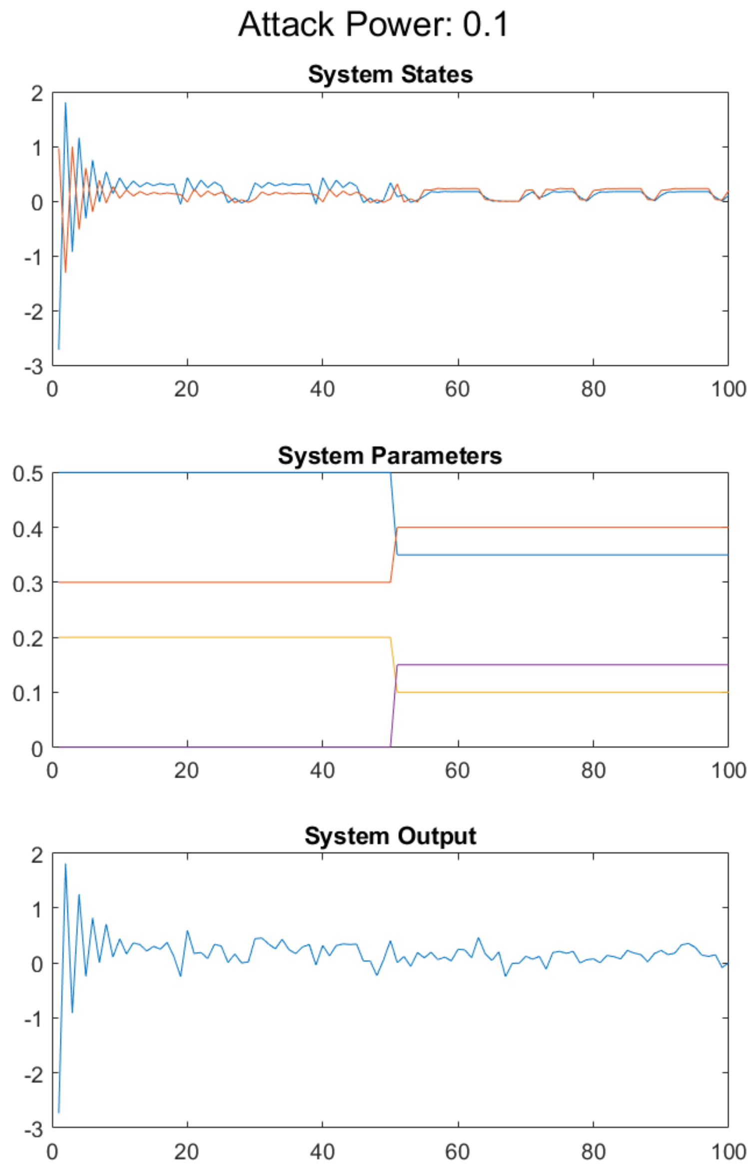

Fig. 10. $v_0 = 0.1$ System Results

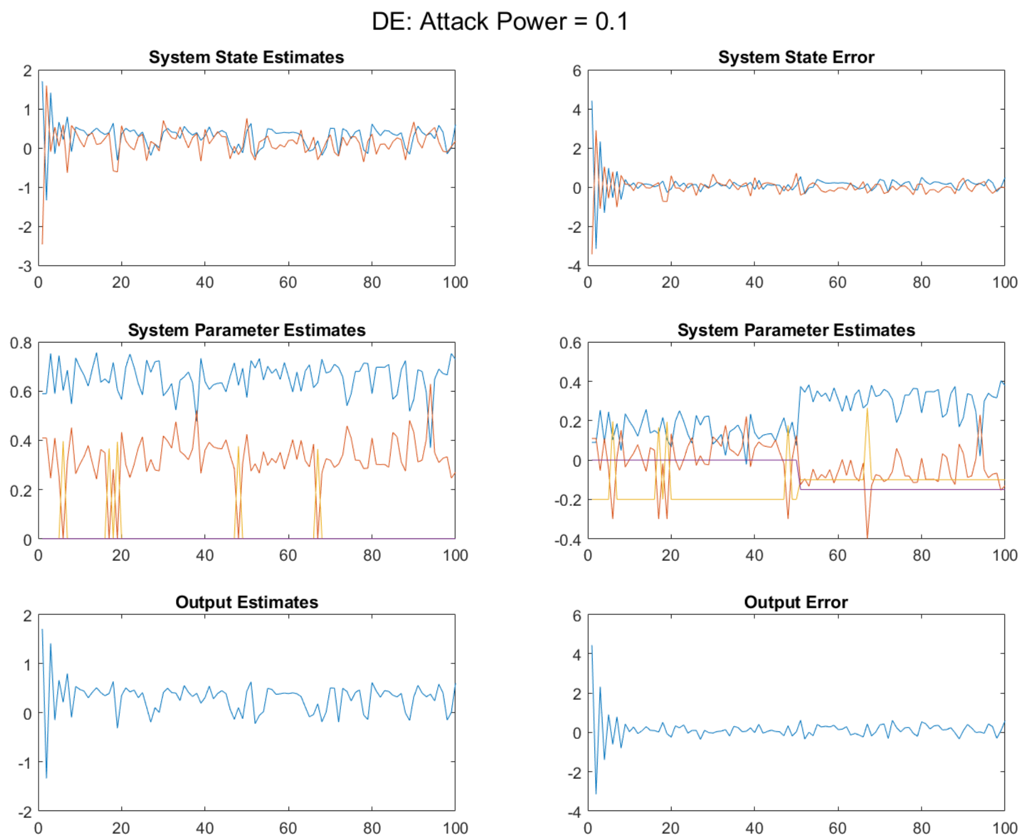

Fig. 11. DE with $v_0 = 0.1$ Simulated Results

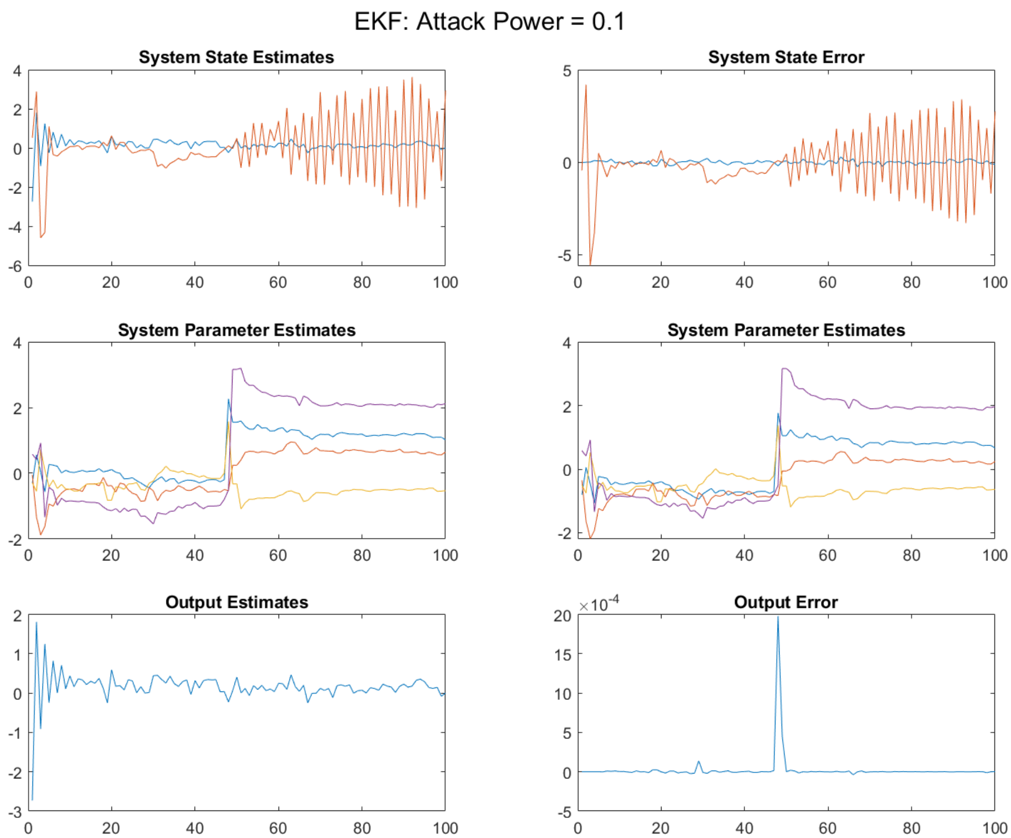

Fig. 12. EKF with $v_0 = 0.1$ Simulated Results

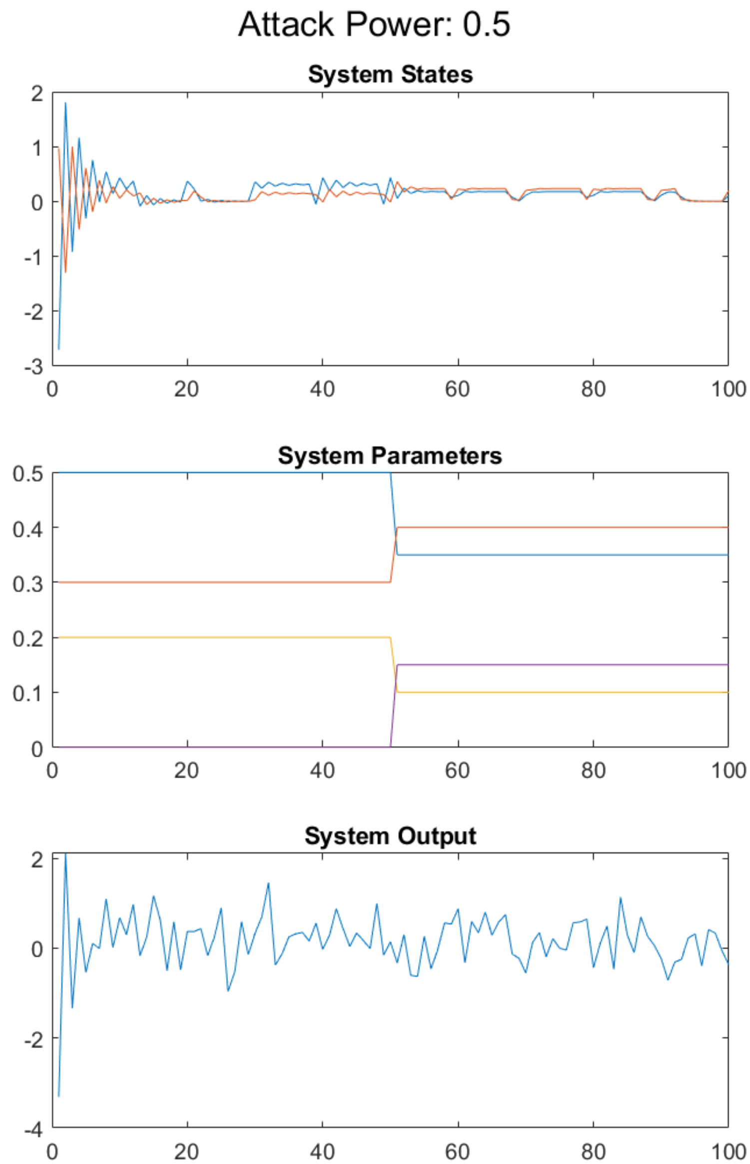

Fig. 13. $v_0 = 0.5$ System Results

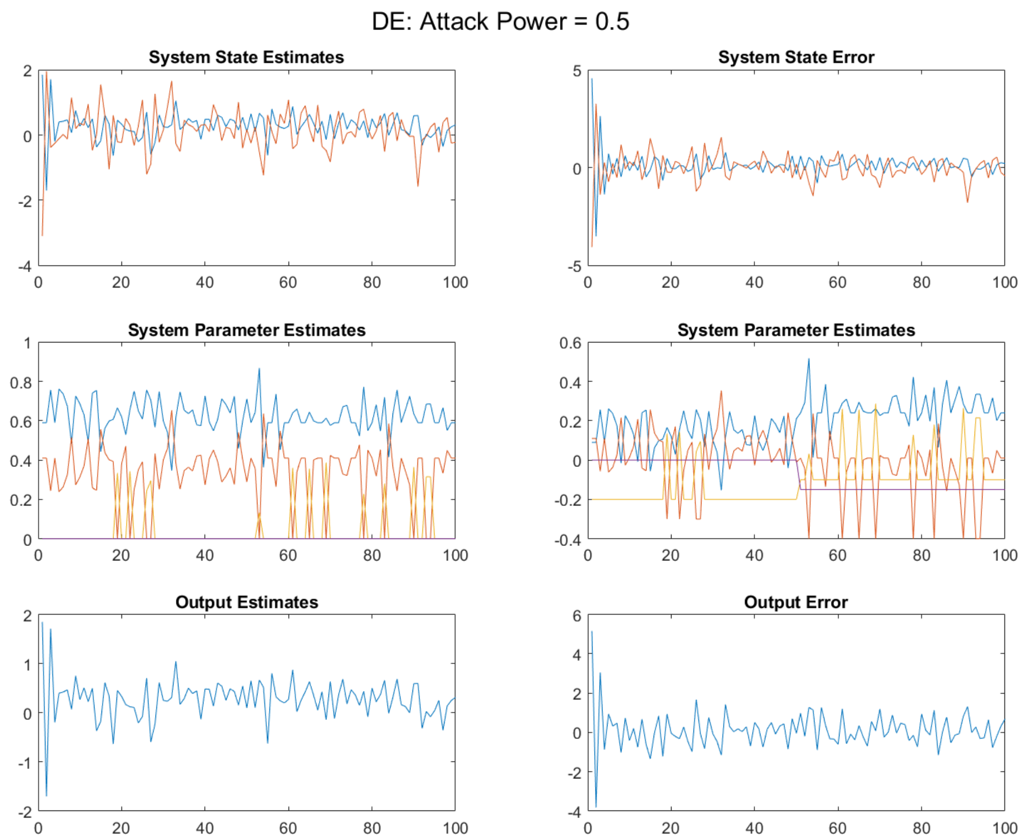

Fig. 14. DE with $v_0 = 0.5$ Simulated Results

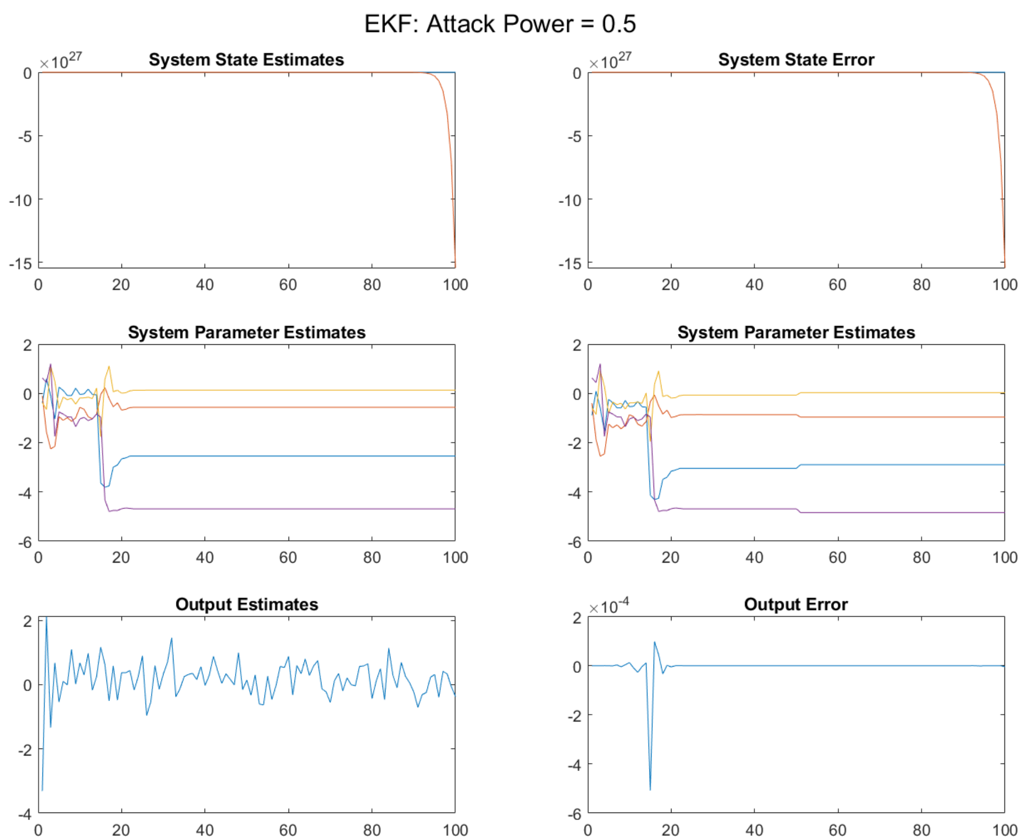

Fig. 15. EKF with $v_0 = 0.5$ Simulated Results