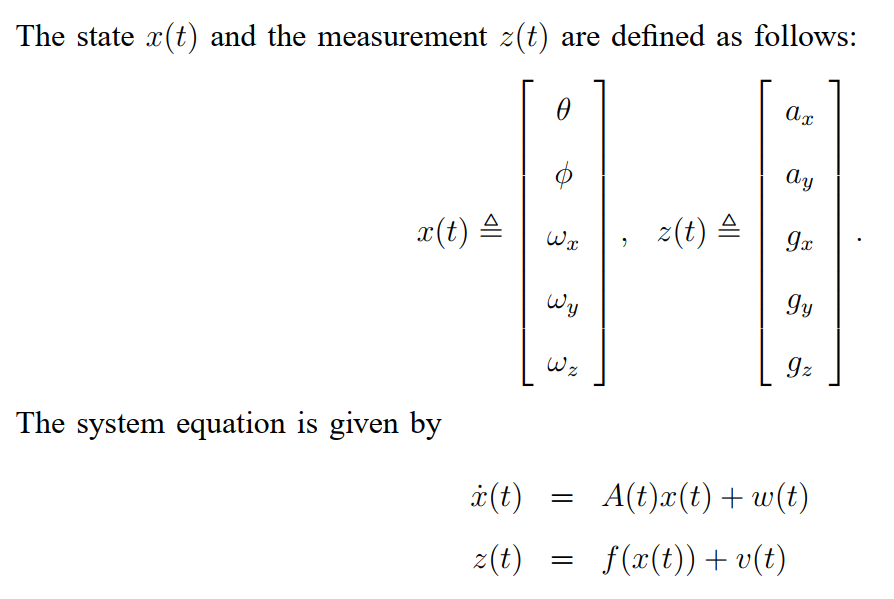
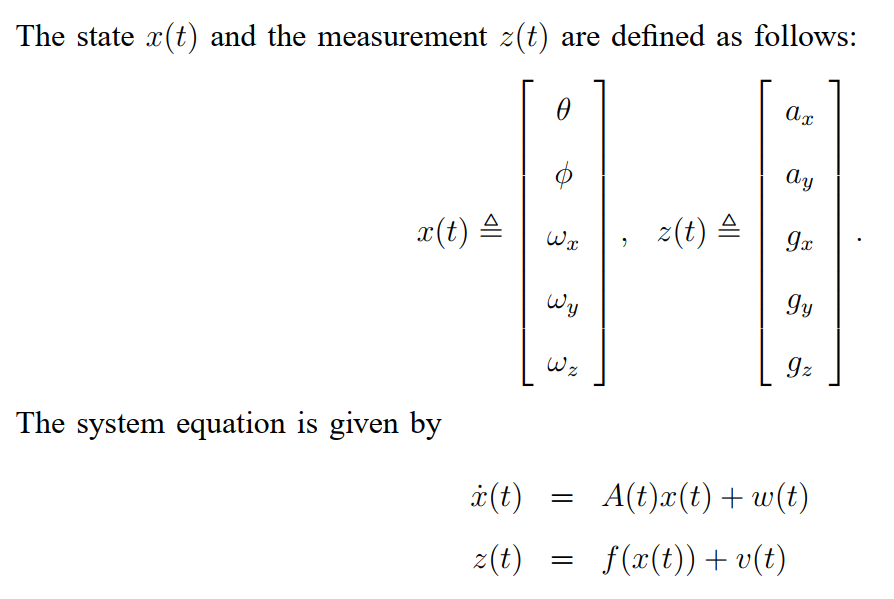
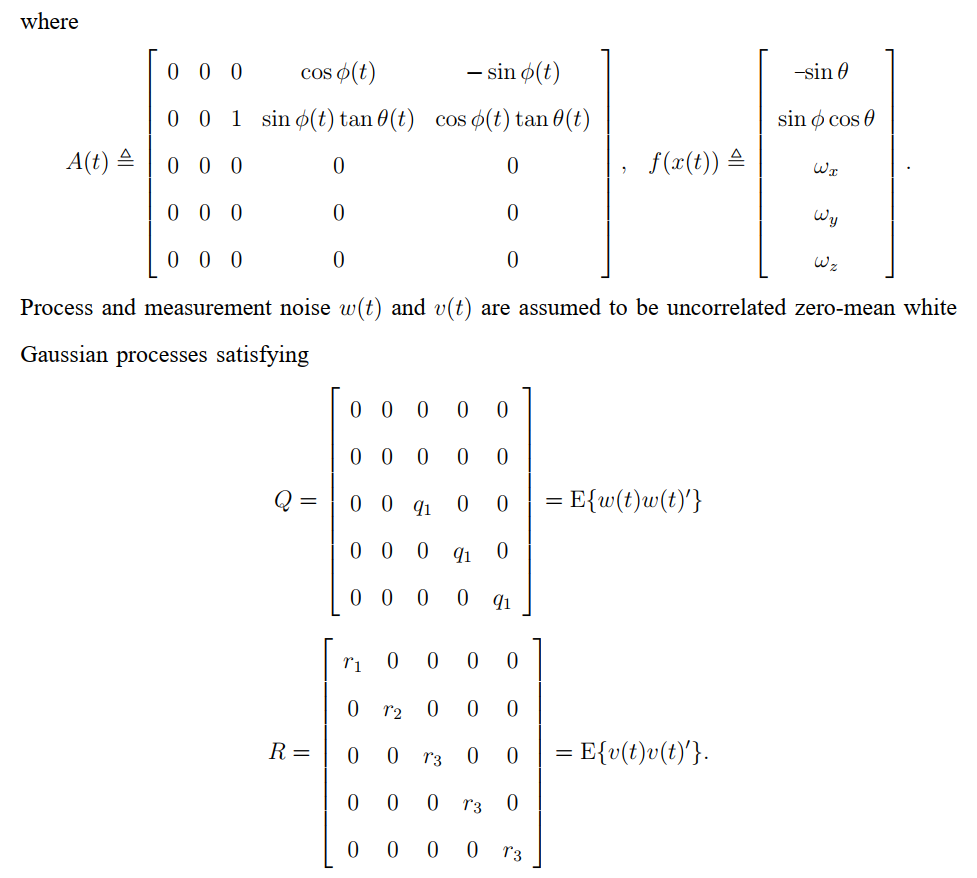
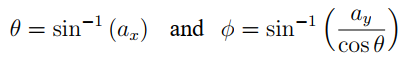
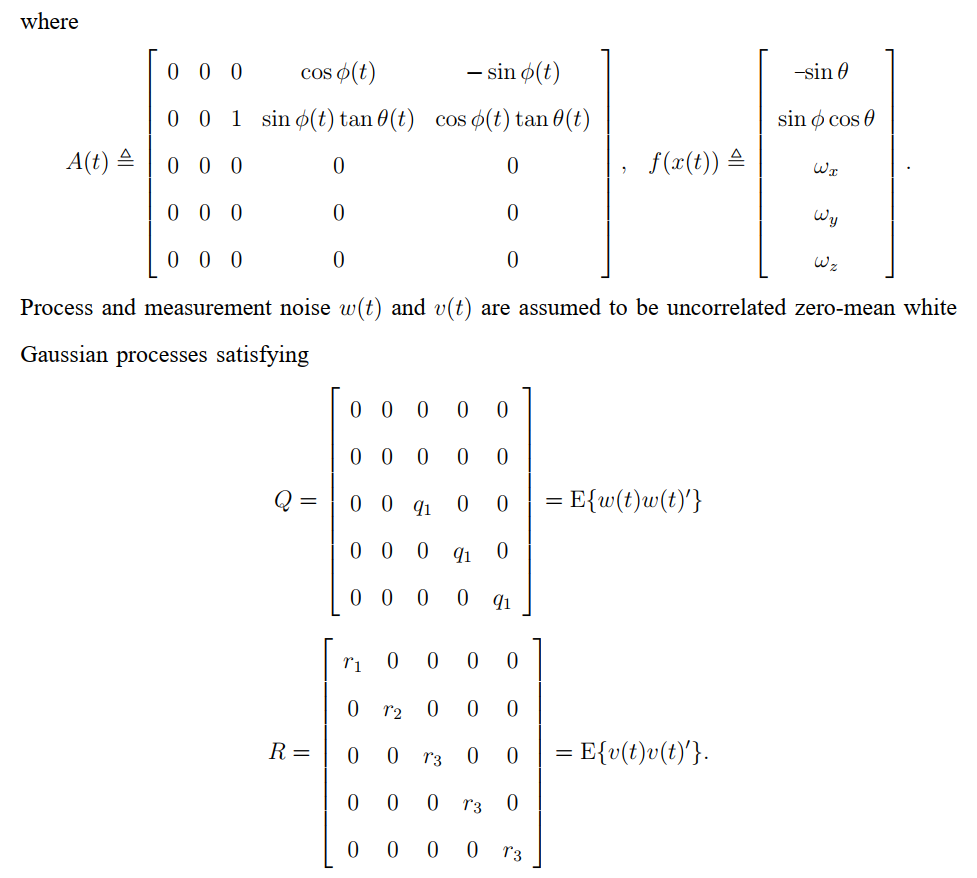
### **Thông tin về hệ thống**











### **Thực hiện rời rạc hóa các biến trạng thái (không xét nhiễu)**



**Ma trận Jacobian:**



### **Thực hiện rời rạc hóa các tín hiệu quan sát (không xét nhiễu)**

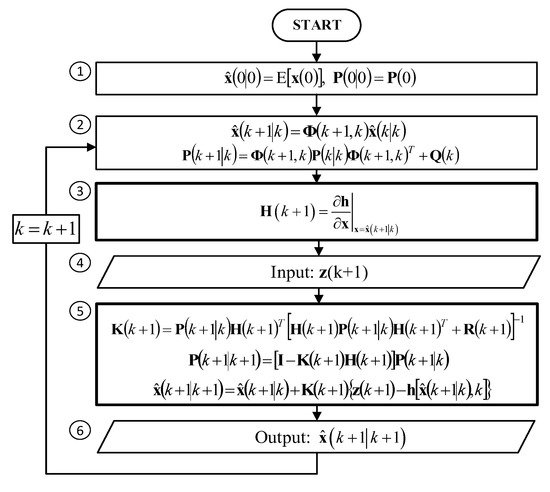




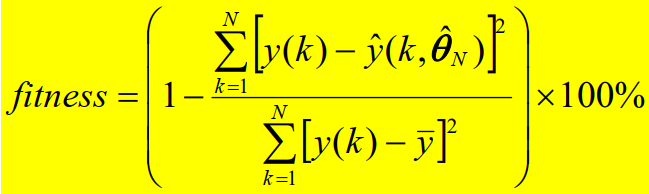
Ma trận Jacobian:



### **Vòng lặp bộ lọc Extended Kalman**



### **Đánh giá kết quả bằng độ phù hợp (fitness) và sai số toàn phương trung bình (RMSE)**

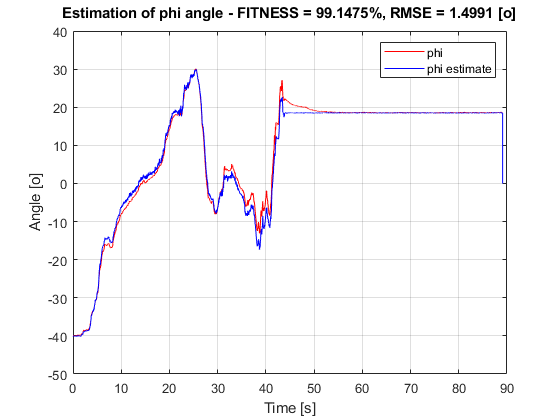
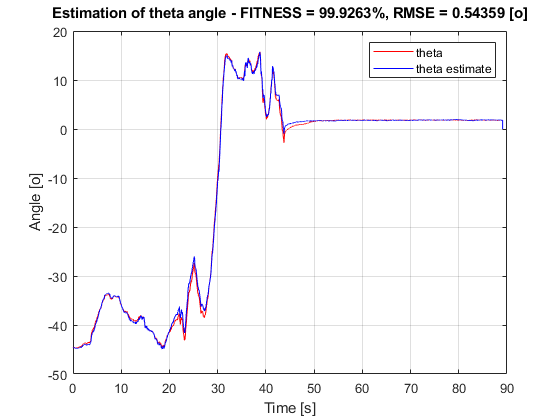
 ; 

## **CHƯƠNG TRÌNH MATLAB THỰC HIỆN BÀI TOÁN**

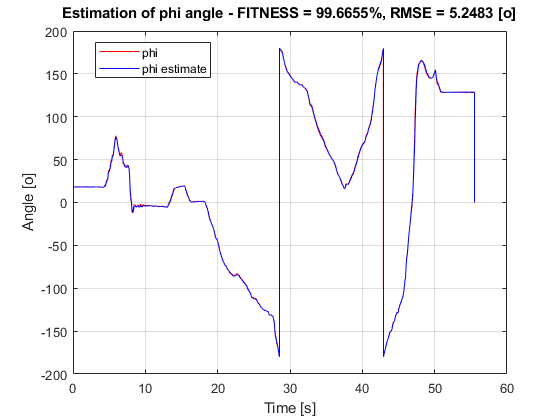
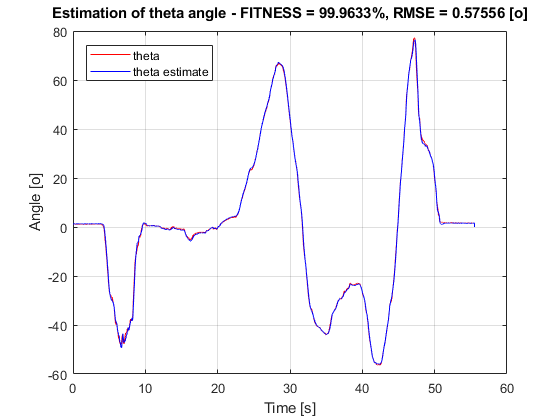
|  |
| --- |
| clear all;  clc;  %% Load data from XSENS imu  %% Select dataset “I” from provided datasets in pattern1\...:    %----- i=0:4 -----%  I = 0; % Select i  raw\_data = load(strcat(‘pattern1\MT\_cal\_00300827\_00’,num2str(i),’.log’));  euler\_data = load(strcat(‘pattern1\MT\_euler\_00300827\_00’,num2str(i),’.log’));    %% Basic information:  N = max(size(euler\_data)); % total dynamic steps  n = 5; % number of state  Ts = 0.01; % sample rate  tt = euler\_data(:,1); % time stamp    %% Data pre-processing:  g\_const = 9.81;  acc = raw\_data(:,2:3)/g\_const; % [m/s^2]  gyro = raw\_data(:,5:7); % [rad/s]  mag = raw\_data(:,8:10); % [mgauge]  mag = mag/norm(mag)\*10^(-3); % [gauge]    actual\_value = zeros(N,2); % Actual value  actual\_value(:,1) = euler\_data(:,3); % theta [o]  actual\_value(:,2) = euler\_data(:,2); % phi [o]    %% Std of process:  % Choose q1 for the best estimation  switch i  case 0  q1 = 5;  case 1  q1 = 0.5;  case 2  q1 = 5;  case 3  q1 = 30;  case 4  q1 = 1;  % General case:  otherwise  q1 = 10000;  end  Q = diag([0,0,q1,q1,q1]);  L = diag([0,0,1,1,1])\*Ts;    %% Std of measurement:  % Choose r1, r2, r3 for the best estimation  switch i  case 0  r1 = 0.1;  r2 = 0.0001;  r3 = 0.01;  case 1  r1 = 0.0006;  r2 = 0.01;  r3 = 0.02;  case 2  r1 = 0.001;  r2 = 0.002;  r3 = 0.01;  case 3  r1 = 0.01;  r2 = 0.01;  r3 = 0.02;  case 4  r1 = 0.00005;  r2 = 0.0007;  r3 = 0.01;  % General case:  otherwise  r1 = 0.001;  r2 = 0.001;  r3 = 0.001;  end  R = diag([r1,r2,r3,r3,r3]);  M = eye(5);    %% Allocate memory for estimation values:  estimate\_value = zeros(N,n);    %% Initialize values:  P\_minus = zeros(n,n); % Initialize P\_minus  P = zeros(n,n); % Initialize P  % Initialize x\_hat\_minus and x\_hat for the best estimation  switch i  case 0  x\_hat\_minus = [-44.62\*pi/180; -39.94\*pi/180; -0.014954; 0.000145; 0.016868];  x\_hat = x\_hat\_minus;  case 1  x\_hat\_minus = [1.21\*pi/180; 18.19\*pi/180; -0.003714; -0.000136; -0.018198];  x\_hat = x\_hat\_minus;  case 2  x\_hat\_minus = [1.64\*pi/180; -1.22\*pi/1808; -0.022062; -0.001835; 0.016377];  x\_hat = x\_hat\_minus;  case 3  x\_hat\_minus = [-10.81\*pi/180; 4.46\*pi/180; -0.004984; 0.060824; 0.000847];  x\_hat = x\_hat\_minus;  case 4  x\_hat\_minus = [-16.21\*pi/180; 27.80\*pi/180; -0.004858; -0.009552; -0.003057];  x\_hat = x\_hat\_minus;  % General case:  otherwise  x\_hat\_minus = zeros(n,n);  x\_hat = zeros(n,n);  end    %% Kalman filter loop:  for k=1:N-1  %% (value substition for the next step)  theta = x\_hat\_minus(1);  phi = x\_hat\_minus(2);  omega\_x = x\_hat\_minus(3);  omega\_y = x\_hat\_minus(4);  omega\_z = x\_hat\_minus(5);  %% Compute Jacobian matrix of g(u\_t,x\_t\_1):  Jg12 = - Ts\*omega\_y\*sin(phi) – Ts\*omega\_z\*cos(phi);  Jg14 = Ts\*cos(phi);  Jg15 = -Ts\*sin(phi);  Jg21 = Ts\*omega\_z\*cos(phi)\*(tan(theta)^2 + 1) + Ts\*omega\_y\*sin(phi)\*(tan(theta)^2 + 1);  Jg22 = Ts\*omega\_y\*cos(phi)\*tan(theta) – Ts\*omega\_z\*sin(phi)\*tan(theta) + 1;  Jg24 = Ts\*sin(phi)\*tan(theta);  Jg25 = Ts\*cos(phi)\*tan(theta);    Jg = [1, Jg12 , 0 , Jg14 , Jg15;...  Jg21 , Jg22, Ts , Jg24 , Jg25;...  0 , 0 , 1 , 0 , 0;...  0 , 0 , 0 , 1 , 0;...  0 , 0 , 0 , 0 , 1];    %% (value substition for the next step)  theta = x\_hat(1);  phi = x\_hat(2);  omega\_x = x\_hat(3);  omega\_y = x\_hat(4);  omega\_z = x\_hat(5);    g11 = theta – Ts\*omega\_z\*sin(phi) + Ts\*omega\_y\*cos(phi);  g21 = phi + Ts\*omega\_x + Ts\*omega\_z\*cos(phi)\*tan(theta) + Ts\*omega\_y\*sin(phi)\*tan(theta);  g31 = omega\_x;  g41 = omega\_y;  g51 = omega\_z;    g=[g11; g21; g31; g41; g51];    %% Project the error covariance ahead:  x\_hat\_minus = g;  P\_minus = Jg\*P\*Jg’ +L\*Q\*L’;    %% (value substition for the next step)  theta = x\_hat\_minus(1);  phi = x\_hat\_minus(2);  omega\_x = x\_hat\_minus(3);  omega\_y = x\_hat\_minus(4);  omega\_z = x\_hat\_minus(5);    %% Compute Jacobian matrix of h(x\_t):  Jh11 = -cos(theta);  Jh21 = -sin(phi)\*sin(theta);  Jh22 = cos(phi)\*cos(theta);    Jh=[Jh11 , 0 , 0 , 0 , 0;  Jh21 , Jh22 , 0 , 0 , 0;  0 , 0 , 1 , 0 , 0;  0 , 0 , 0 , 1 , 0;  0 , 0 , 0 , 0 , 1];    %% Compute Kalman gain:  S = Jh\*P\_minus\*Jh’+M\*R\*M’;  K = P\_minus\*Jh’\*inv(S);    %% (value substition for the next step)  h11 = -sin(theta);  h21 = sin(phi)\*cos(theta);  h31 = omega\_x;  h41 = omega\_y;  h51 = omega\_z;    hx = [h11; h21; h31; h41; h51];    %% Update estimate with measurement:  z = [acc(k,1); acc(k,2); gyro(k,1); gyro(k,2); gyro(k,3)];  x\_hat = x\_hat\_minus + K\*(z – hx);  estimate\_value(k,😊 = x\_hat; % Store estimate values    %% Compute error covariance for updated estimate:  P = (eye(n)-(K\*Jh))\*P\_minus;    %% Estimate value data post-processing:  while(estimate\_value(k,1)<-pi)  estimate\_value(k,1)=estimate\_value(k,1)+2\*pi;  end  while(estimate\_value(k,2)<-pi)  estimate\_value(k,2)=estimate\_value(k,2)+2\*pi;  end    while(estimate\_value(k,1)>pi)  estimate\_value(k,1)=estimate\_value(k,1)-2\*pi;  end  while(estimate\_value(k,2)>pi)  estimate\_value(k,2)=estimate\_value(k,2)-2\*pi;  end    end    %% Visualize the results  title\_labels = {‘Estimation of theta angle’, ‘Estimation of phi angle’};  legend\_labels = {{‘theta’, ‘theta estimate’}, {‘phi’, ‘phi estimate’}};  for I = 1:2  figure(i);  plot(tt, actual\_value(:,i),'r');  hold on;  plot(tt, estimate\_value(:,i)\*180/pi,'b');  hold off;  legend(legend\_labels{i});  xlabel('Time [s]');  ylabel('Angle [o]');  grid on;    %% Fitness evaluation  fitness = fitnessCalculator(estimate\_value(:,i)\*180/pi, actual\_value(:,i), N);  RMSE = errorCalculator(estimate\_value(:,i)\*180/pi, actual\_value(:,i), N);  txt = append('\_ - FITNESS = ',num2str(fitness),'%',', RMSE = ',num2str(RMSE),' [o]');  title(append(title\_labels{i}, txt));  end    function [fitness] = fitnessCalculator(estimate, true, N)  mean\_value = mean(true(:));  a=0;  b=0;  for k=1:N  a=a+(true(k)-estimate(k))^2;  b=b+(true(k)-mean\_value)^2;  end  fitness = (1-a/b)\*100;  end  function [RMSE] = errorCalculator(estimate, true, N)  RMSE = 0;  for k=1:N  RMSE = RMSE + (true(k)-estimate(k))^2;  end  RMSE = sqrt(RMSE/(N));  end |

## **KẾT QUẢ NHẬN DẠNG**

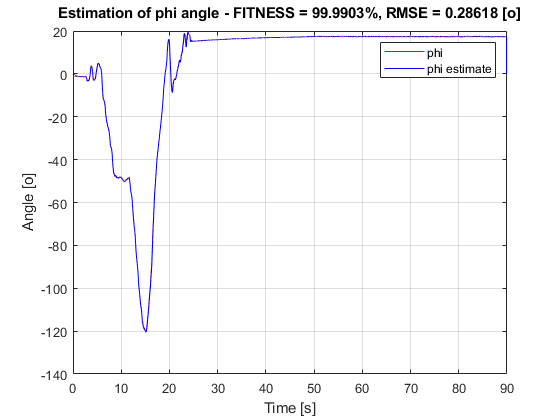
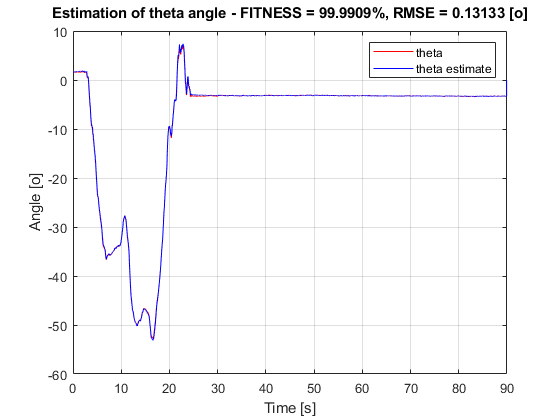
* **Kết quả nhận dạng trên tập dataset** MT\_cal\_00300827\_000.log



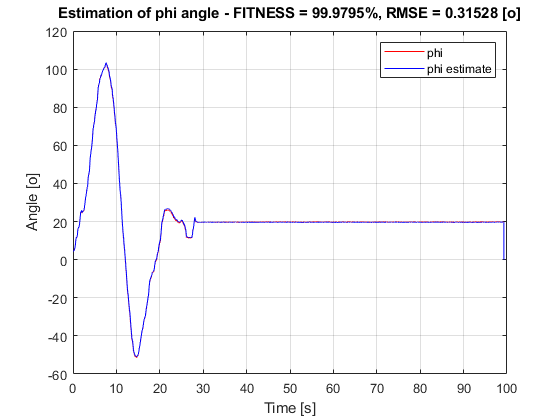
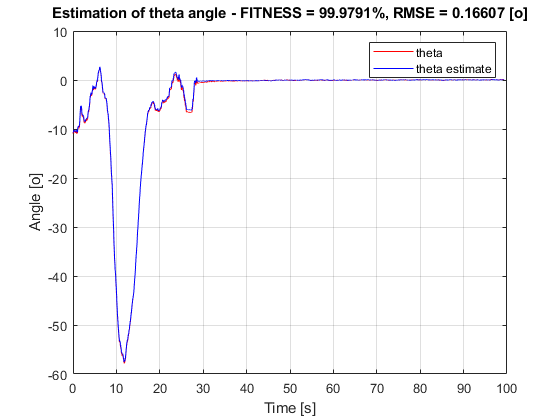
* **Kết quả nhận dạng trên tập dataset** MT\_cal\_00300827\_001.log



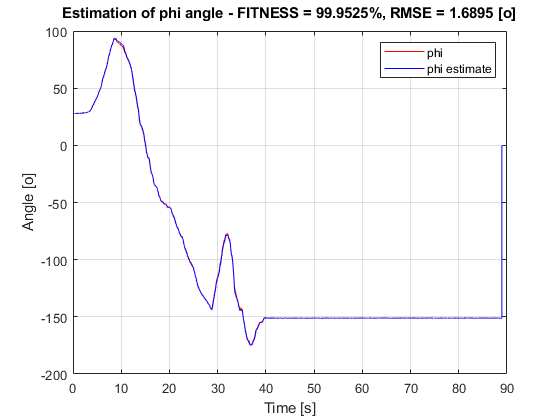
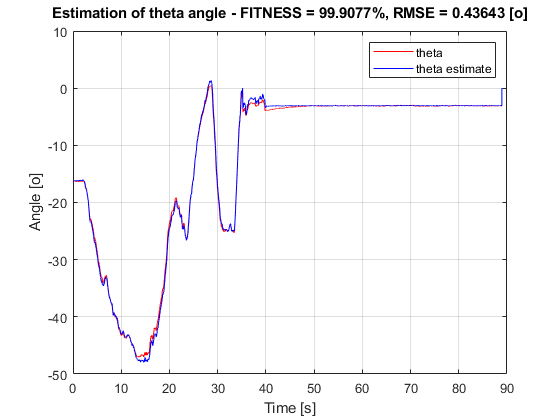
* **Kết quả nhận dạng trên tập dataset** MT\_cal\_00300827\_002.log



* **Kết quả nhận dạng trên tập dataset** MT\_cal\_00300827\_003.log



* **Kết quả nhận dạng trên tập dataset** MT\_cal\_00300827\_004.log



### **Nhận xét kết quả**

Độ phù hợp của ước lượng trong bài làm đạt kết quả gần như chính xác, chỉ tiêu fitness đạt được ở các ước lượng đều trên 99% và cũng như sai số toàn phương trung bình ước lượng nhỏ.