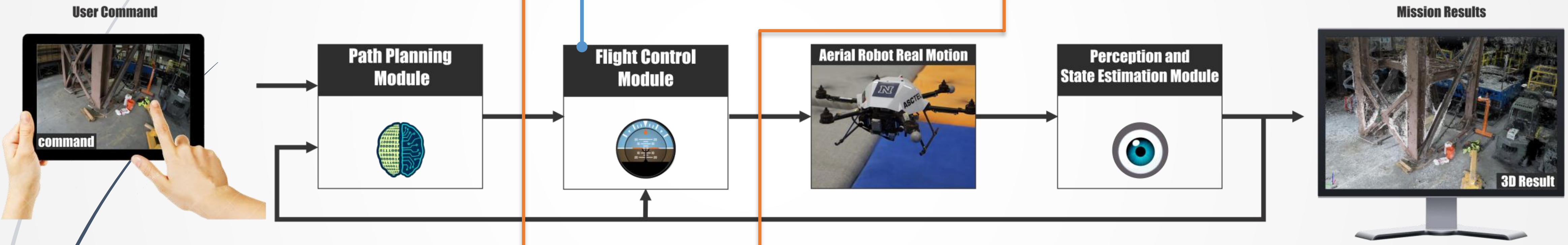
CS491/691: Introduction to Aerial Robotics

Topic: LQR Flight Control

Dr. Kostas Alexis (CSE)

The Aerial Robot Loop

- How to design the flight control of an aerial robot and program its autopilot.



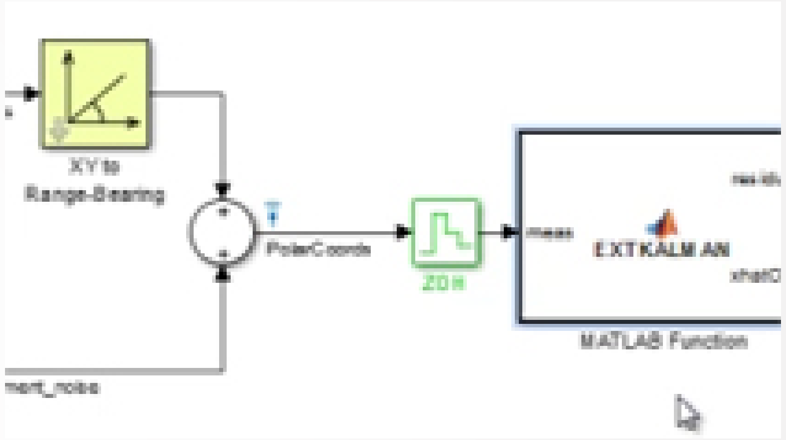
Section 3 of our course

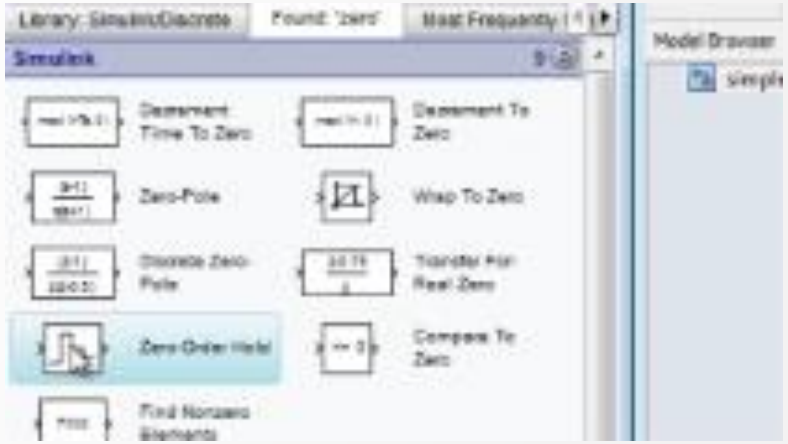
MATLAB Simulink

- Simulink® is a block diagram environment for multidomain simulation and [Model-Based Design](). It supports simulation, automatic code generation, and continuous test and verification of embedded systems.
- Simulink provides a graphical editor, customizable block libraries, and solvers for modeling and simulating dynamic systems. It is integrated with MATLAB®, enabling you to incorporate MATLAB algorithms into models and export simulation results to MATLAB for further analysis.

MATLAB Simulink

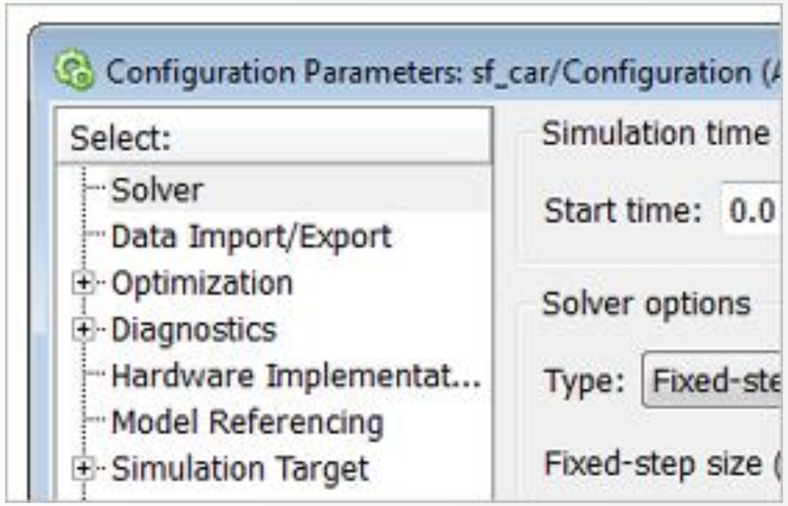
- Building the Model
  - Simulink® provides a set of predefined blocks that you can combine to create a detailed [block diagram]() of your system. Tools for hierarchical modeling, data management, and subsystem customization enable you to represent even the most complex system concisely and accurately.
- Selecting Blocks
  - Continuous and discrete dynamics blocks, such as Integration and Unit Delay
  - Algorithmic blocks, such as Sum, Product, and Lookup Table
  - Structural blocks, such as Mux, Switch, and Bus Selector
  - Customized blocks
- Building and Editing the Model
  - You build a model by dragging blocks from the Simulink Library Browser into the Simulink Editor. You then connect these blocks with signal lines to establish mathematical relationships between system components.
  - The Simulink Editor gives you complete control over what you see and use within the model.

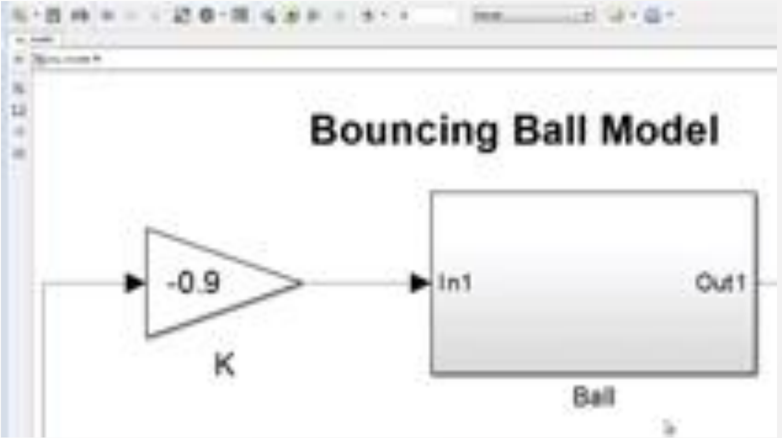




MATLAB Simulink

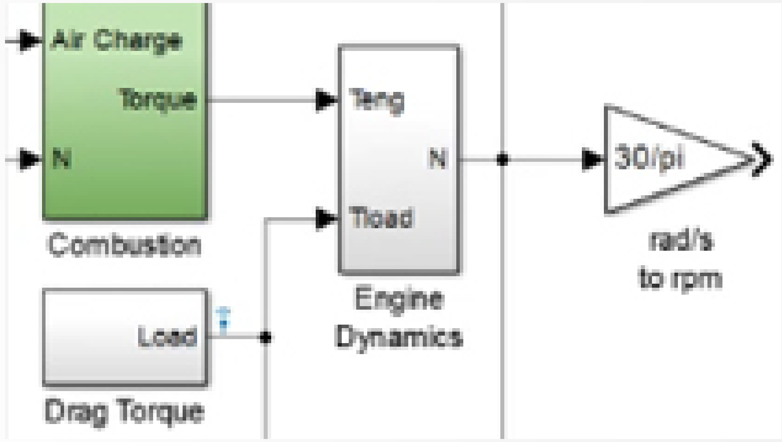
- Simulating the Model
  - You can simulate the dynamic behavior of your system and view the results as the simulation runs. To ensure simulation speed and accuracy, Simulink provides fixed-step and variable-step ODE solvers, a graphical debugger, and a model profiler.Selecting Blocks
- Choosing a Solver
  - Solvers are numerical integration algorithms that compute the system dynamics over time using information contained in the model. Simulink provides solvers to support the simulation of a broad range of systems, including continuous-time (analog), discrete-time (digital), hybrid (mixed-signal), and multirate systems of any size.
- Running the Simulation
  - Normal (the default), which interpretively simulates your model
  - Accelerator, which increases simulation performance by creating and executing compiled target code but still provides the flexibility to change model parameters during simulation
  - Rapid Accelerator, which can simulate models faster than Accelerator mode by creating an executable that can run outside Simulink on a second processing core





MATLAB Simulink

- Analyzing Simulation Results
  - After running a simulation, you can analyze the simulation results in MATLAB and Simulink. Simulink includes debugging tools to help you understand the simulation behavior.
- Viewing Simulation Results
  - You can visualize the simulation behavior by viewing signals with the displays and scopes provided in Simulink. You can also view simulation data within the Simulation Data Inspector, where you can compare multiple signals from different simulation runs.
  - Alternatively, you can build custom HMI displays using MATLAB, or log signals to the MATLAB workspace to view and analyze the data using MATLAB algorithms and visualization tools.
- Debugging the Simulation
  - Simulink supports debugging with the Simulation Stepper, which lets you step back and forth through your simulation viewing data on scopes or inspecting how and when the system changes states.
  - With the Simulink debugger you can step through a simulation one method at a time and examine the results of executing that method. As the model simulates, you can display information on block states, block inputs and outputs, and block method execution within the Simulink Editor.



Linear Quadratic Regulator

- **Theorem:** Consider the system: $\dot{x} = Ax + Bu$ and the performance index:

$$J = \int_0^\infty \left[x^T(t)Qx(t) + u^T(t)Ru(t)\right] dt$$

- where $Q = M^T M$, R is symmetric and positive definite, (A,B) is stabilizable, and (A,M) is detectable. The optimal control is:

$$u(t) = -R^{-1}B^T Px$$

- where P is the symmetric positive semidefinite solution of the Algebraic Riccati Equation (ARE):

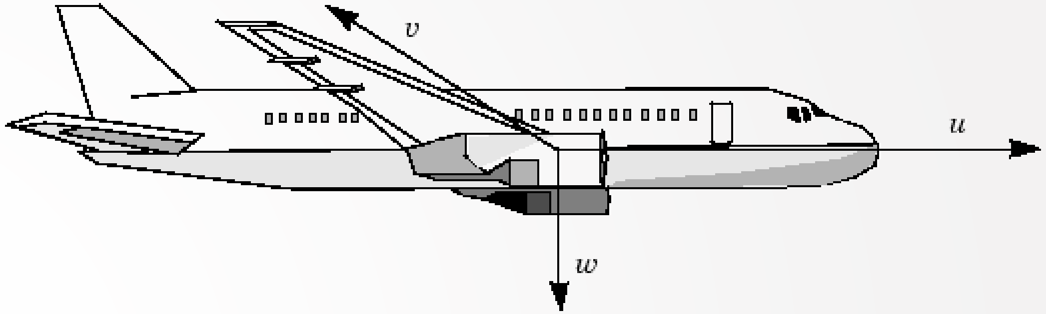
$$0 = PA + A^T P + Q - PBR^{-1}B^T P$$

Design LQR Servo Control

- **State-Space Equations for an Airframe**

$$\dot{x} = Ax + Bu$$

$$x = [u, v, w, p, q, r, \theta, \phi]^T$$



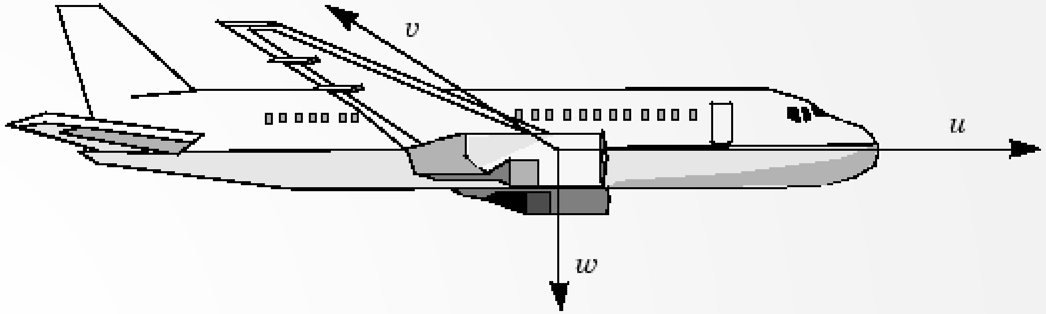
- **Nonlinear Component of the State-Space Equation**

$$\dot{x} = Ax + Bu + \begin{bmatrix} -g \sin\theta \\ g \cos\theta \sin\phi \\ g \cos\theta \cos\phi \\ 0 \\ 0 \\ 0 \\ q \cos\phi - r \sin\phi \\ (q \sin\phi + r \cos\phi) \cdot \tan\theta \end{bmatrix}$$

Based on:
http://www.mathworks.com/help/control/getstart/design-an-lqr-servo-controller-in-simulink.html

Design LQR Servo Control

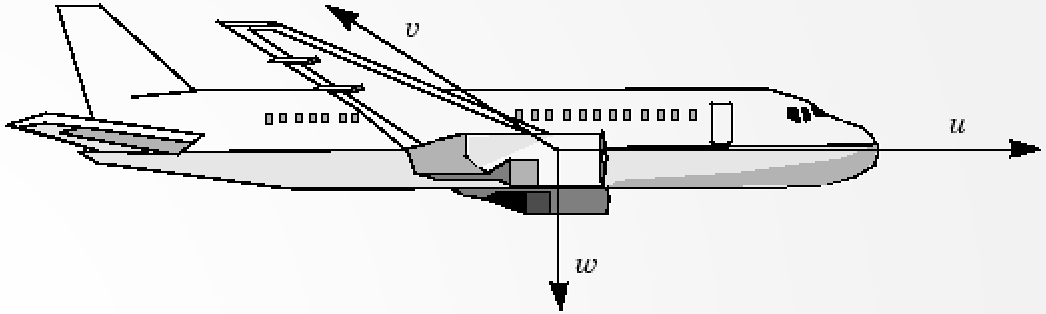
- **Trimming**
  - For LQG design purposes, the nonlinear dynamics are trimmed at φ=15 and p, q, r, and θ set to zero. Since u, v, and w do not enter into the nonlinear term in the preceding figure, this amounts to linearizing around (θ,φ)=(0,15) with all remaining states set to zero.




```
>> A15

A15 =

   -0.0404    0.0618    0.0501         0   -0.0005         0  -32.2000         0
   -0.1686   -1.1889    7.6870         0    0.0041         0         0   31.1028
    0.1633   -2.6139   -3.8519         0    0.0489         0         0   -8.3340
         0         0         0   -0.3386   -0.0474   -6.5405         0         0
         0         0         0   -1.1288   -0.9149   -0.3679         0         0
         0         0         0    0.9931   -0.1763   -1.2047         0         0
         0         0    0.9056         0    0.9659   -0.2588         0         0
         0         0         0         0    0.9467   -0.0046         0         0
```

Design LQR Servo Control

- **Trimming**
  - For LQG design purposes, the nonlinear dynamics are trimmed at φ=15 and p, q, r, and θ set to zero. Since u, v, and w do not enter into the nonlinear term in the preceding figure, this amounts to linearizing around (θ,φ)=(0,15) with all remaining states set to zero.

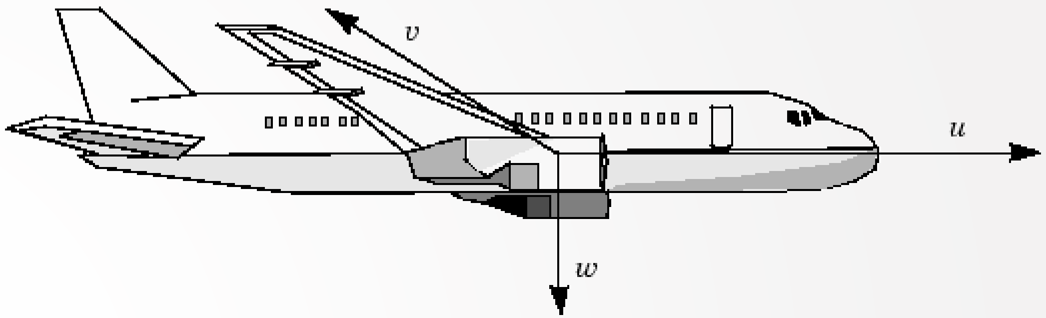



```
>> A15

A15 =

   -0.0404    0.0618    0.0501         0   -0.0005         0  -32.2000         0
   -0.1686   -1.1889    7.6870         0    0.0041         0         0   31.1028
    0.1633   -2.6139   -3.8519         0    0.0489         0         0   -8.3340
         0         0         0   -0.3386   -0.0474   -6.5405         0         0
         0         0         0   -1.1288   -0.9149   -0.3679         0         0
         0         0         0    0.9931   -0.1763   -1.2047         0         0
         0         0    0.9056         0    0.9659   -0.2588         0         0
         0         0         0         0    0.9467   -0.0046         0         0
```

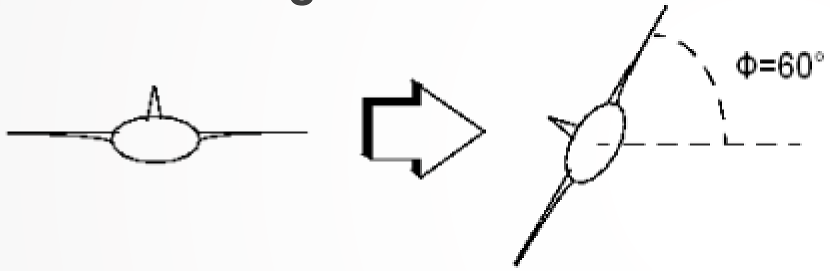
```
>> B

B =

   20.3929   -0.4694   -0.2392   -0.7126
    0.1269   -2.6932    0.0013    0.0033
  -64.6939  -75.6295    0.6007    3.2358
         0         0    0.1865    3.6625
         0         0   23.6053    5.6270
   -0.0001         0    3.9462  -41.4112
         0         0         0         0
         0         0         0         0
```

Design LQR Servo Control

- **Problem Definition**
  - The goal to perform a steady coordinated turn, as shown in this figure.



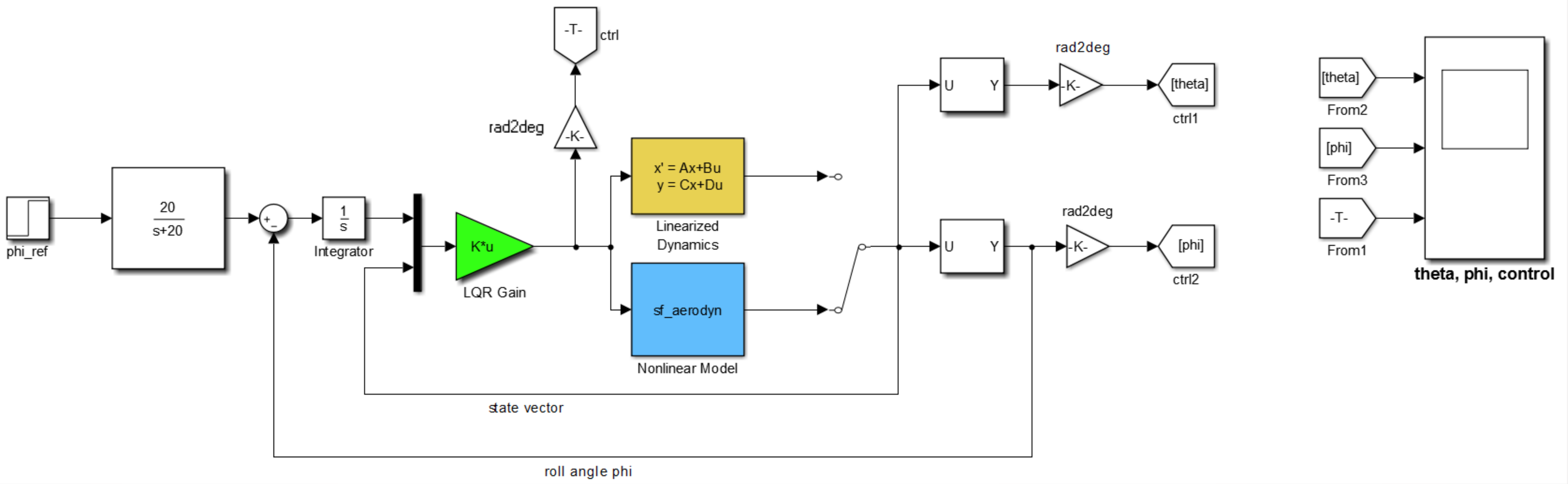
  - **Aircraft Making a 60° Turn**



    - To achieve this goal, you must design a controller that commands a steady turn by going through a 60° roll. In addition, assume that θ, the pitch angle, is required to stay as close to zero as possible.

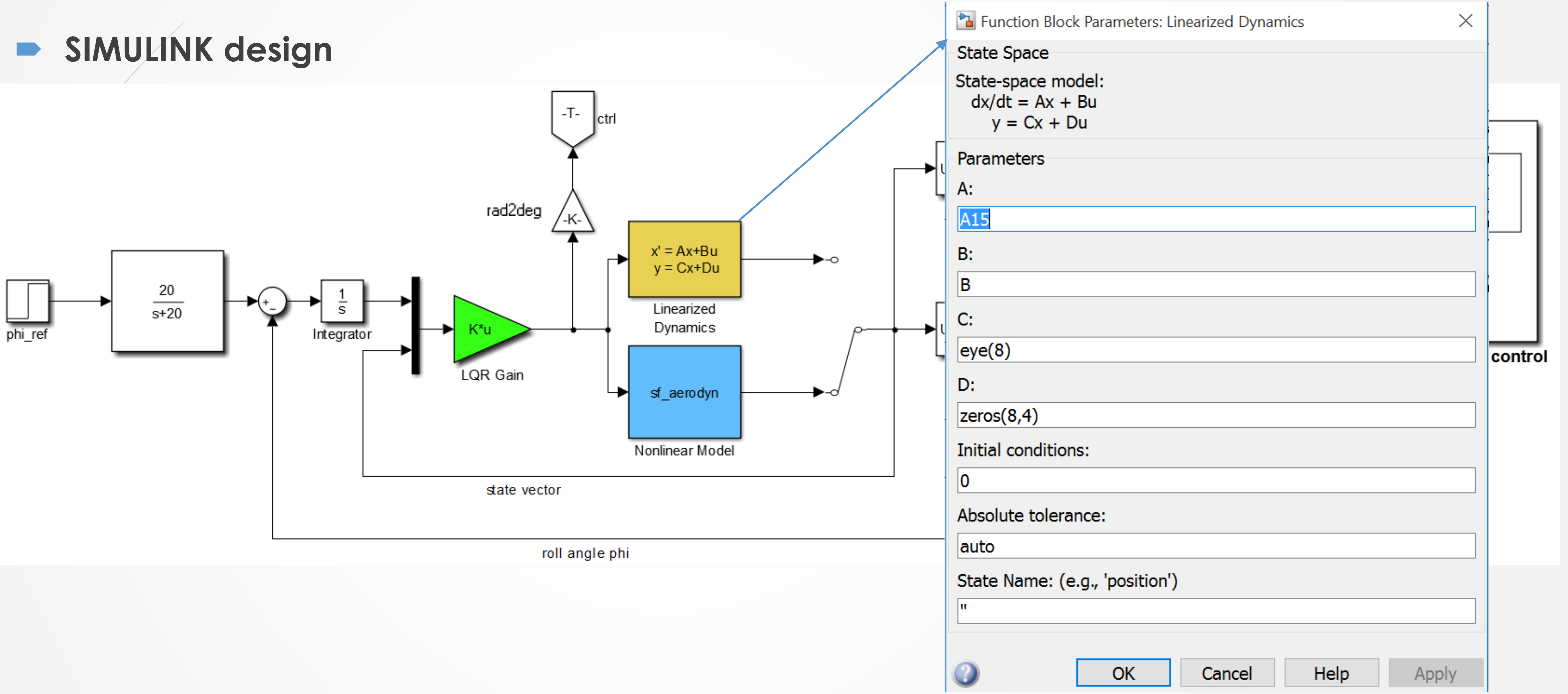
Design LQR Servo Control

- **SIMULINK design**



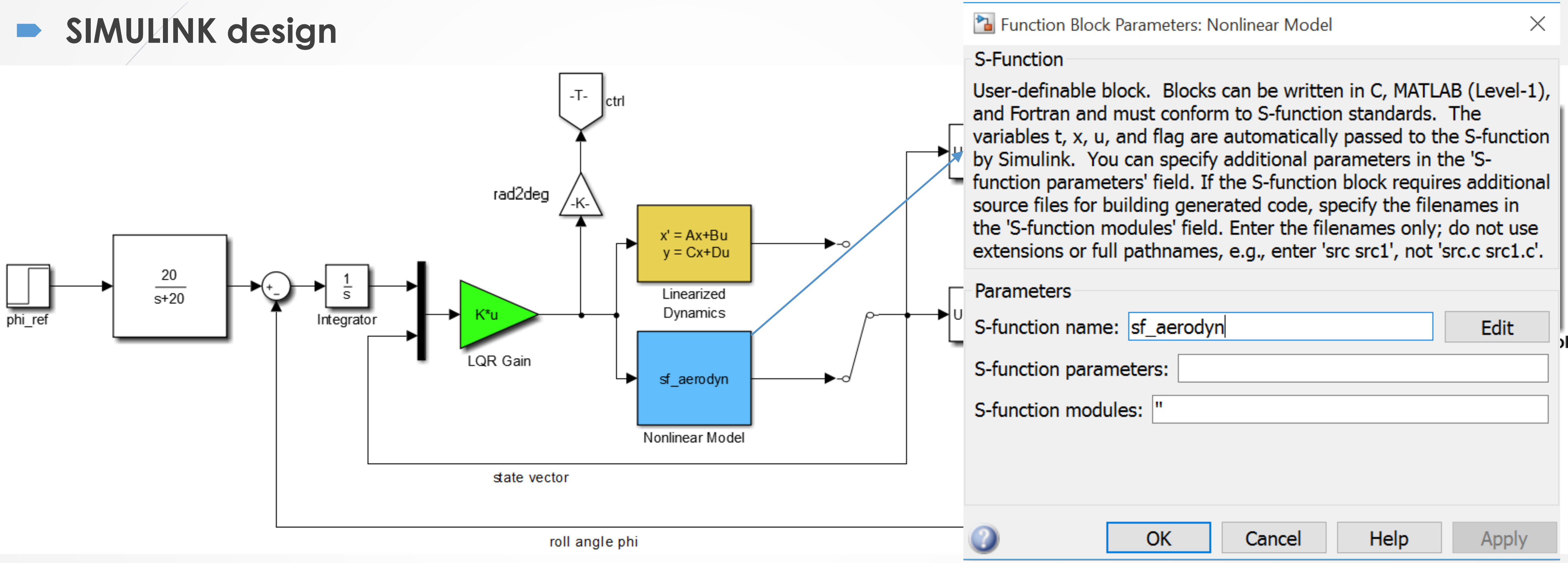
Design LQR Servo Control

- **SIMULINK design**



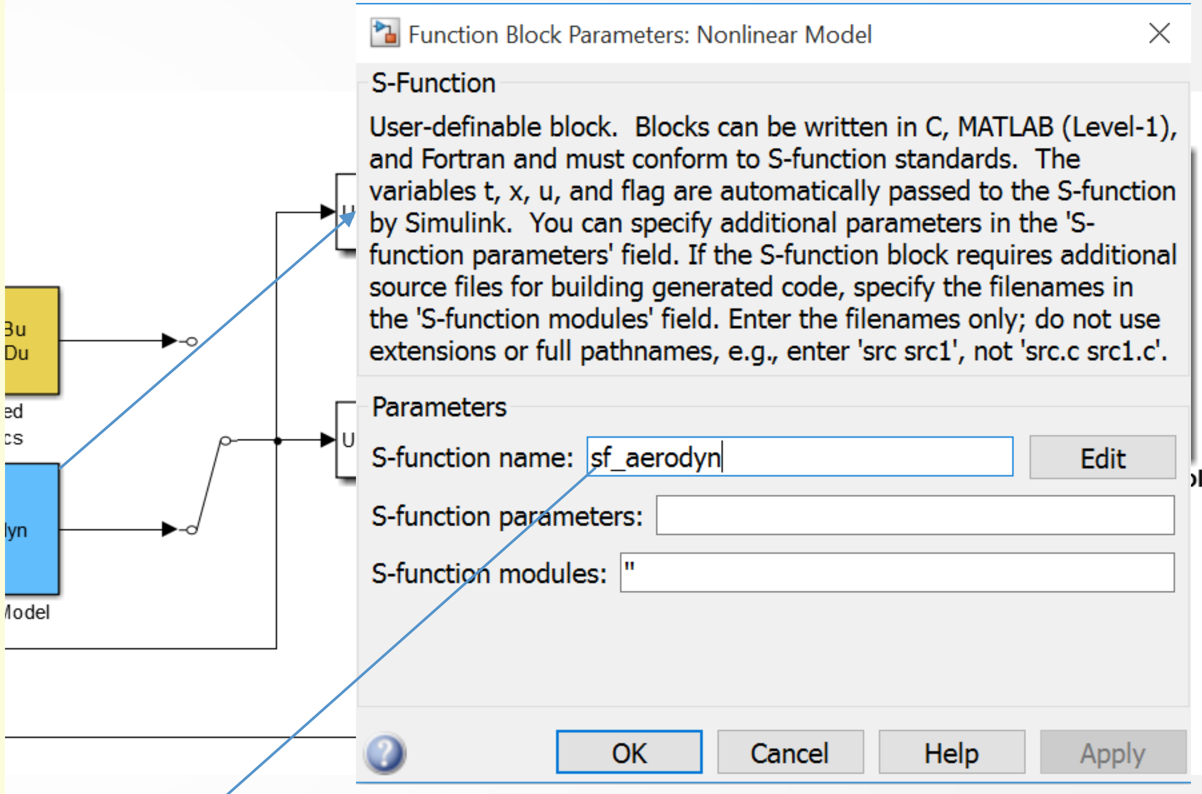
Design LQR Servo Control

- **SIMULINK design**



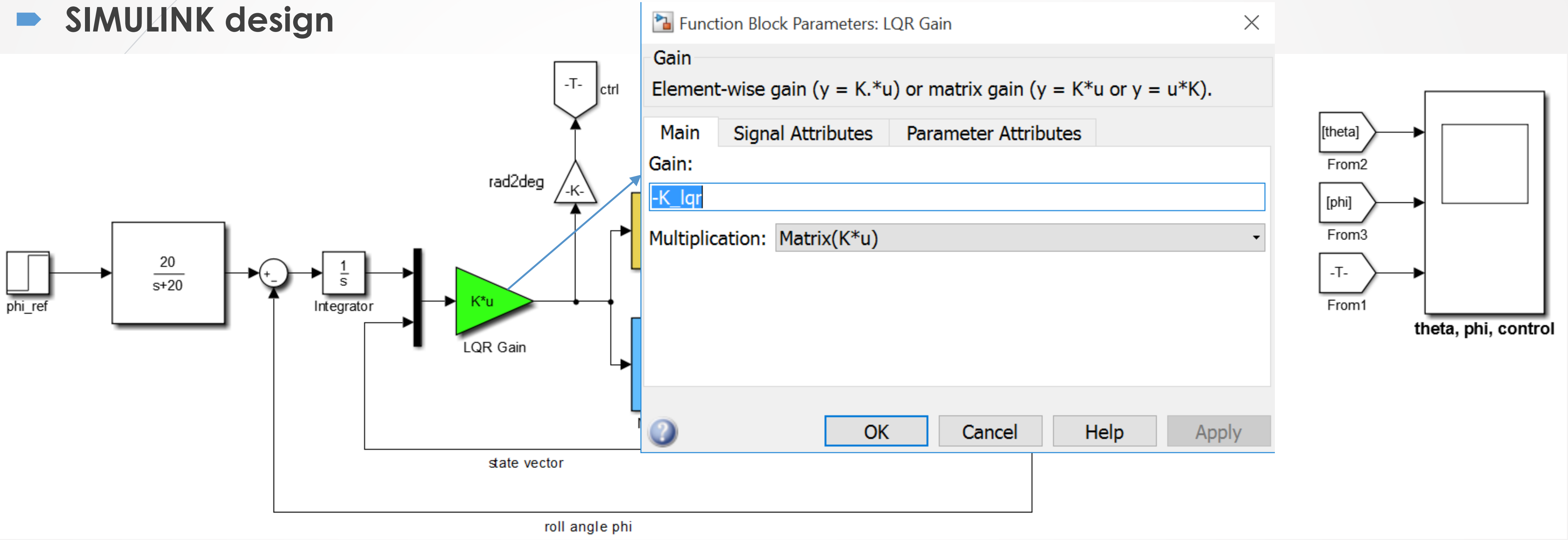
o Control

```
function [sys,x0,str,ts] = sf_aerodyn(t,x,u,flag)
% S-function sf_aerodyn.M
% This S-function represents the nonlinear aircraft dynamics

% Copyright 1986-2007 The MathWorks, Inc.


switch flag,

  %%%%%%%%%%%%%%%%%%
  % Initialization %
  %%%%%%%%%%%%%%%%%%
  case 0,
    [sys,x0,str,ts]=mdlInitializeSizes;

  %%%%%%%%%%%%%%%
  % Derivatives %
  %%%%%%%%%%%%%%%
  case 1,
    sys=mdlDerivatives(t,x,u);

  %%%%%%%%%%
  % Update %
  %%%%%%%%%%
  case 2,
    sys=mdlUpdate(t,x,u);

  %%%%%%%%%%%
  % Outputs %
  %%%%%%%%%%%
  case 3,
    sys=mdlOutputs(t,x,u);

  %%%%%%%%%%%%%%%%%%%%%%%
  % GetTimeOfNextVarHit %
  %%%%%%%%%%%%%%%%%%%%%%%
  case 4,
    sys=mdlGetTimeOfNextVarHit(t,x,u);

  %%%%%%%%%%%%%
  % Terminate %
  %%%%%%%%%%%%%
```



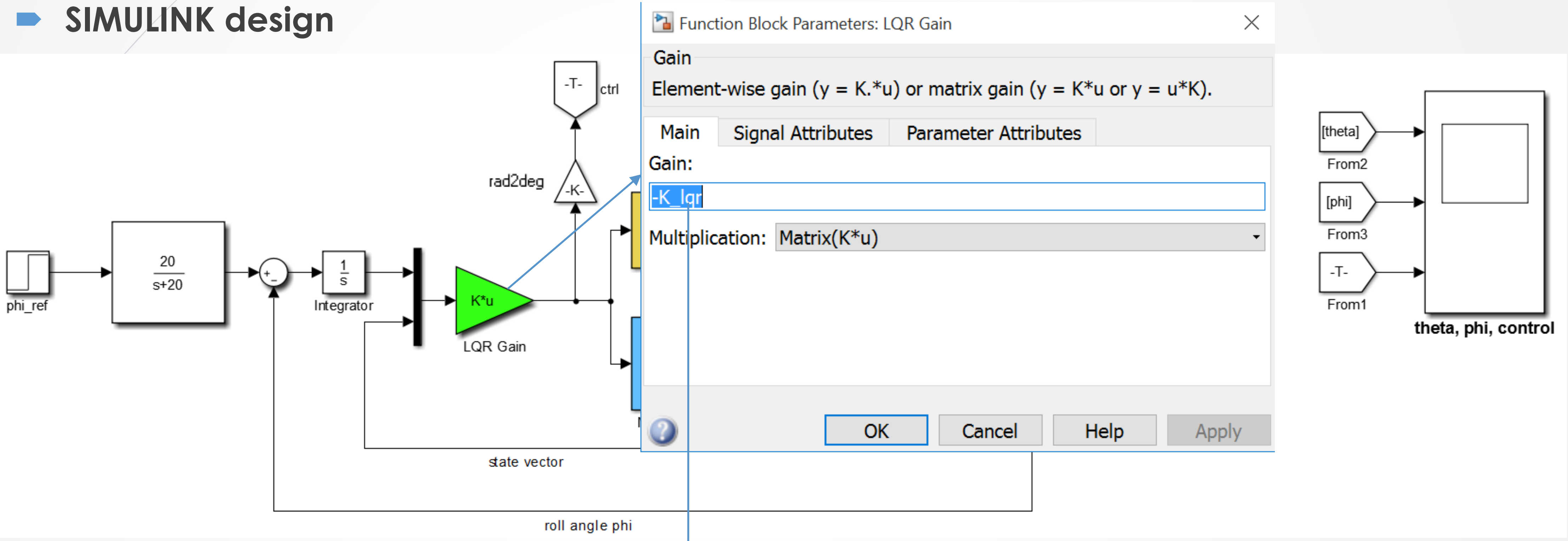
Design LQR Servo Control

- **SIMULINK design**



Design LQR Servo Control

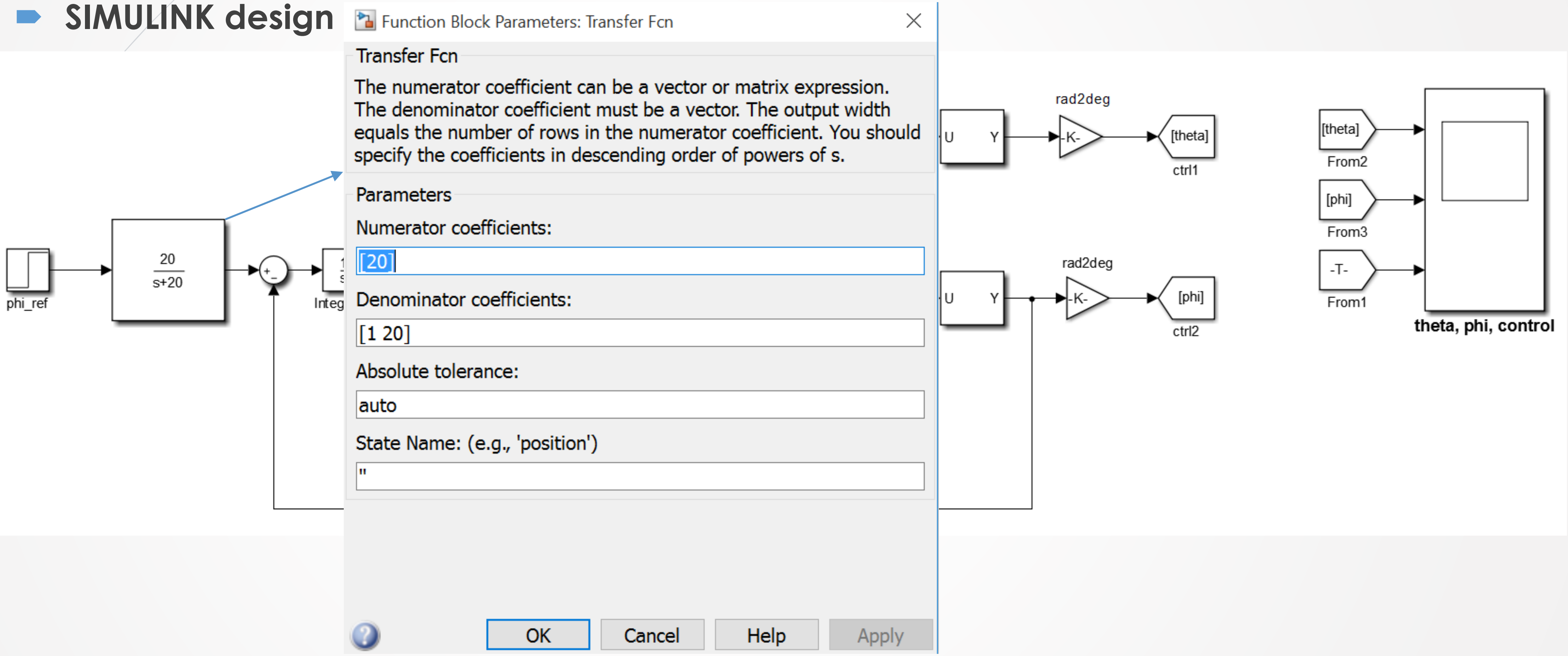
- **SIMULINK design**




```
>> K_lqr

K_lqr =

   -0.2285    0.0489    0.0322   -0.2470    0.0369   -0.1511    0.0329   -5.8225    0.9581
    0.0044   -0.0353    0.0166   -0.0685    0.0125   -0.0330    0.0082   -1.4366    0.3932
   -0.6644   -0.1470    0.1535    0.5090   -0.0737    1.1450   -0.0541   17.2943    4.8754
    0.1878   -0.0072   -0.0514    0.4479    0.2316    0.4978   -0.4808   15.6688   -0.4545
```

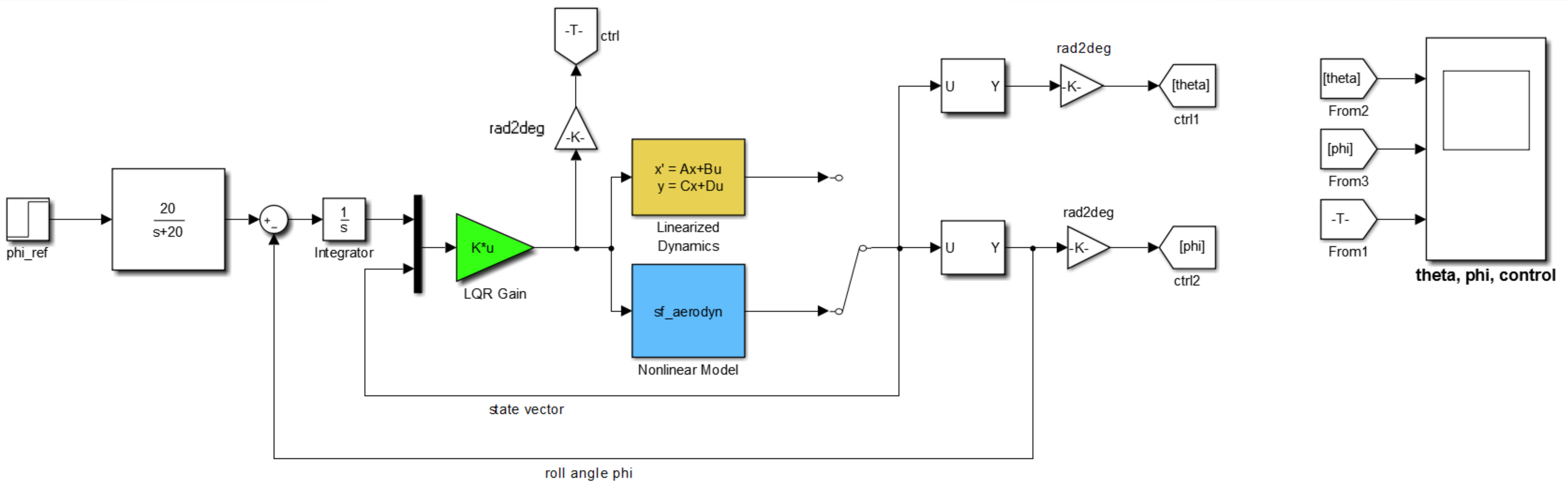
Design LQR Servo Control

- **SIMULINK design**



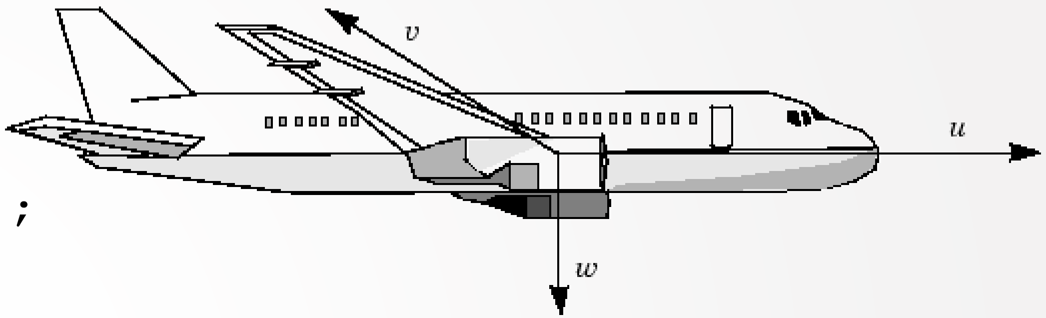
Design LQR Servo Control

- **SIMULINK design**



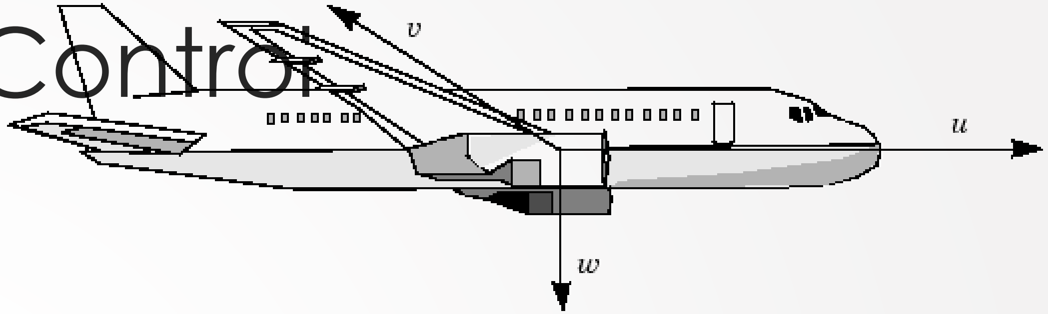
Design LQR Servo Control

- **LQR Servo Control**




```
% Add integrator state dz/dt = -phi
A_aug = [zeros(1,8) -1; zeros(8,1) A15];
B_aug = [zeros(1,4) ; B];

% LQR gain synthesis
Q = blkdiag(1,0.1*eye(6),1000,1);
R = diag([10,50,1,1]);
K_lqr = lqr(A_aug,B_aug,Q,R);
```

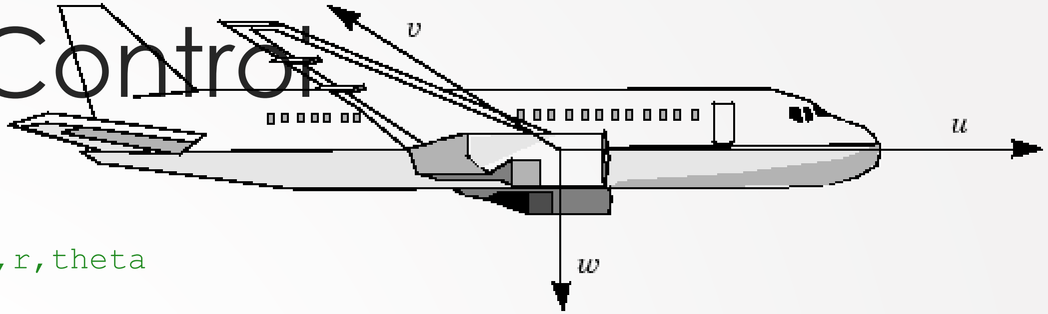
Design LQR Servo Control



- **Modeling**

```
% State vector = [u,v,w,p,q,r,theta,phi]
%       u,v,w: linear velocities
%       p,q,r: roll, pitch, yaw rates
%       theta: pitch angle
%       phi: bank angle

% Linear dynamics
A = [-0.0404    0.0618    0.0501   -0.0000   -0.0005    0.0000
   -0.1686   -1.1889    7.6870         0    0.0041         0
    0.1633   -2.6139   -3.8519    0.0000    0.0489   -0.0000
   -0.0000   -0.0000   -0.0000   -0.3386   -0.0474   -6.5405
   -0.0000    0.0000   -0.0000   -1.1288   -0.9149   -0.3679
   -0.0000   -0.0000   -0.0000    0.9931   -0.1763   -1.2047
         0         0    0.9056         0         0   -0.0000
         0         0   -0.0000         0    0.9467   -0.0046];
A = [A, zeros(8,2)];


B =[ 20.3929   -0.4694   -0.2392   -0.7126
    0.1269   -2.6932    0.0013    0.0033
  -64.6939  -75.6295    0.6007    3.2358
   -0.0000         0    0.1865    3.6625
   -0.0000         0   23.6053    5.6270
   -0.0001         0    3.9462  -41.4112
         0         0         0         0
         0         0         0         0];
```

Design LQR Servo Control

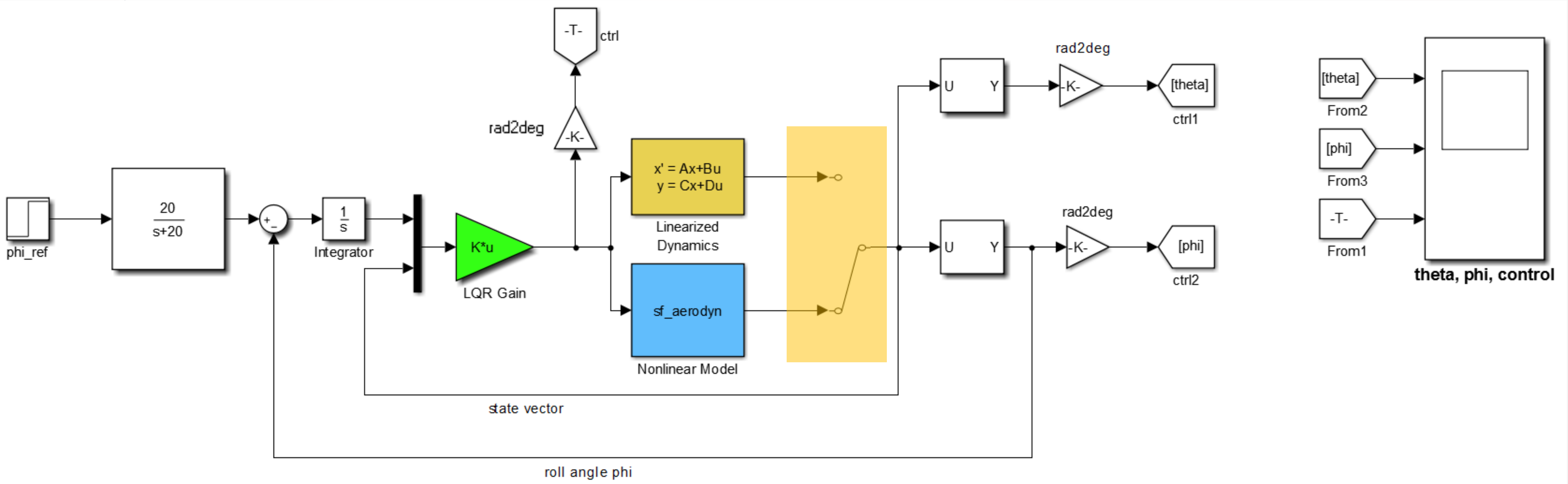


Modeling

```
% Trim nonlinear dynamics about phi=15 degrees with p,q,r,theta
small.
% Note: Since
%           * u,v,w don't enter the nonlinear terms
%           * theta=0 during the maneuver (cancels the trim
values of p,q,r)
%        this is equivalent to linearizing about
x=[0,0,0,0,0,0,0,phi]
g = 32.2;
th0 = 0;
ph0 = 15*pi/180;  % 15 degrees
q0 = 0;
r0 = 0;
A_nl = [...
        0  0 -g*cos(th0) 0; ...
        0  0 -g*sin(th0)*sin(ph0) g*cos(th0)*cos(ph0); ...
        0  0 -g*sin(th0)*cos(ph0) -g*cos(th0)*sin(ph0); ...
        0  0  0  0; ...
        0  0  0  0; ...
        0  0  0  0; ...
        cos(ph0)  -sin(ph0) 0  -q0*sin(ph0)-r0*cos(ph0); ...
        sin(ph0)*tan(th0)  cos(ph0)*tan(th0) ...
           (q0*sin(ph0)+r0*cos(ph0))*(1+tan(th0)^2)
(q0*cos(ph0)-r0*sin(ph0))*tan(th0)];
A15 = A + [zeros(8,4) A_nl];
```

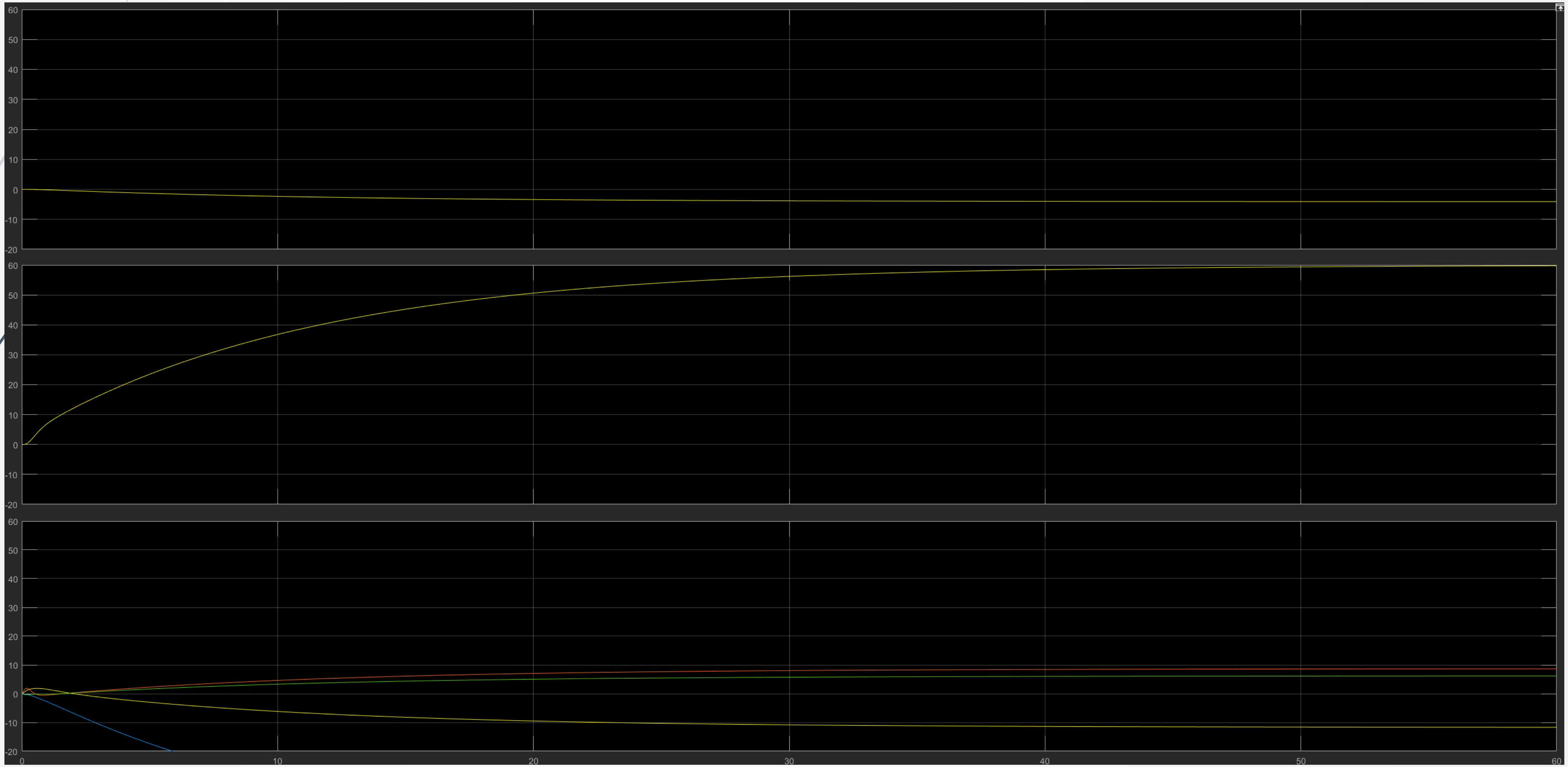
Design LQR Servo Control

- **SIMULINK design**



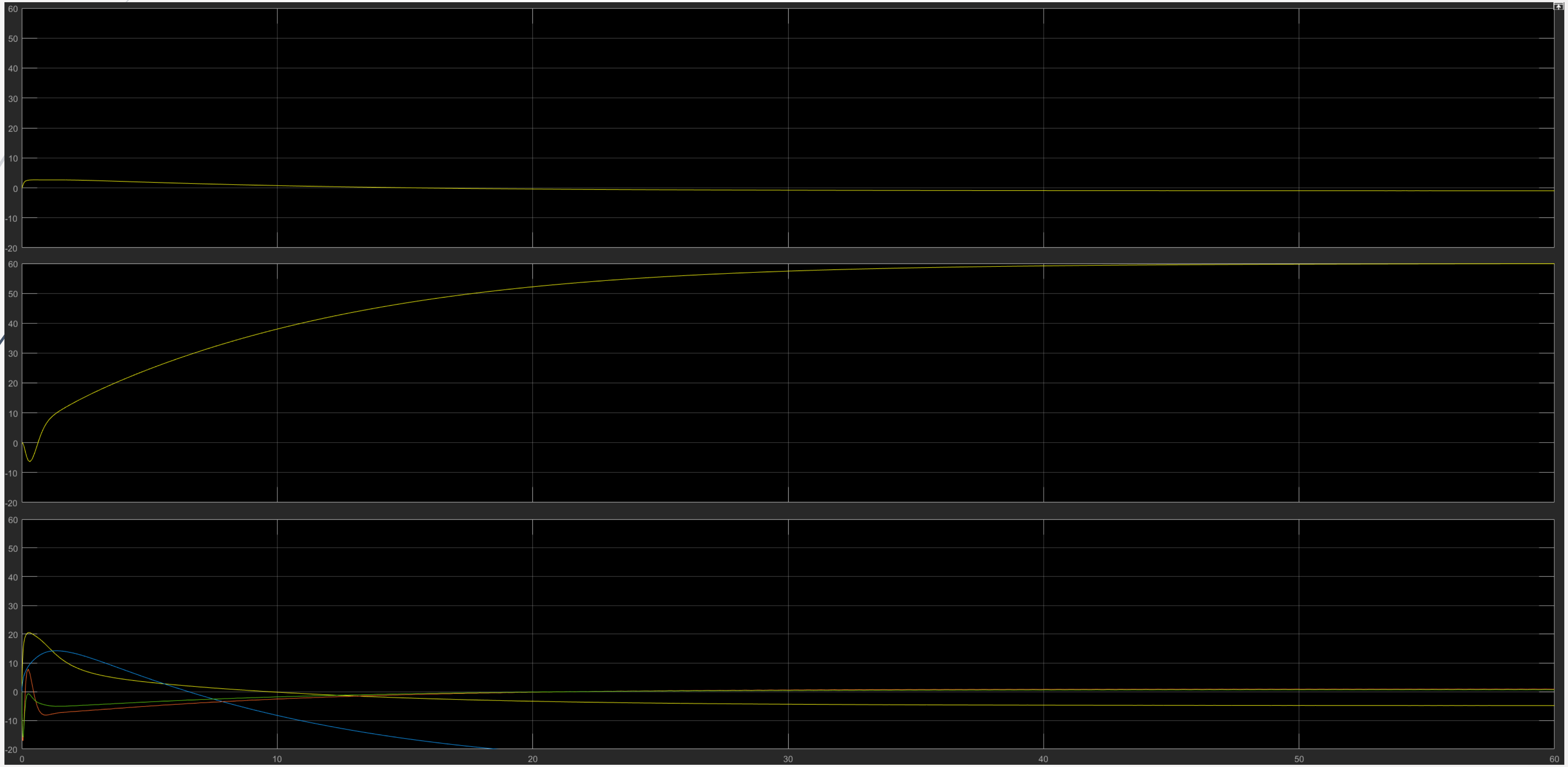
Design LQR Servo Control

- **Linear Response**



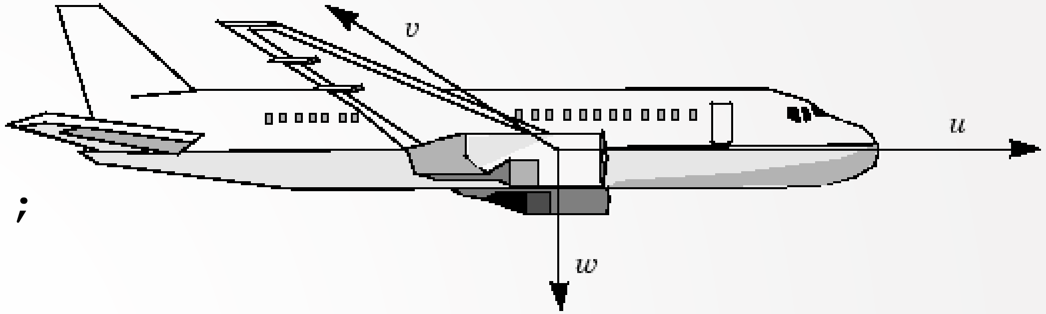
Design LQR Servo Control

- **Nonlinear Response**



Design LQR Servo Control

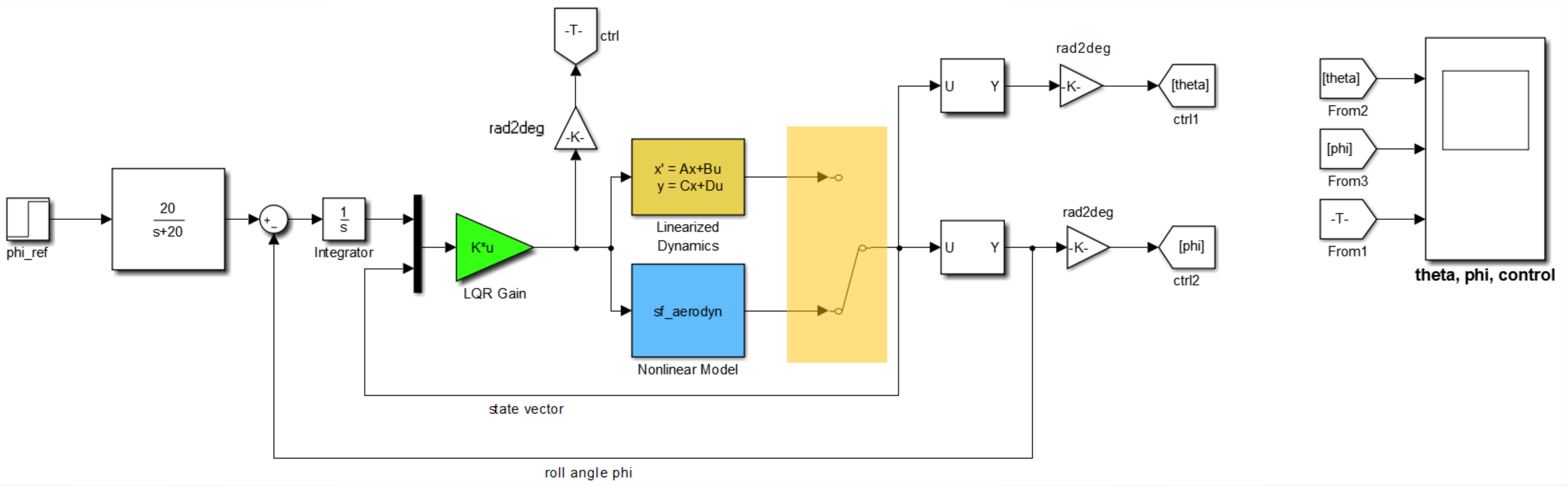
- **LQR Servo Control**




```
% Add integrator state dz/dt = -phi
A_aug = [zeros(1,8) -1; zeros(8,1) A15];
B_aug = [zeros(1,4) ; B];

% LQR gain synthesis
Q = 10*blkdiag(100,1*eye(6),1000,1);
R = diag([10,50,1,1]);
K_lqr = lqr(A_aug,B_aug,Q,R);
```

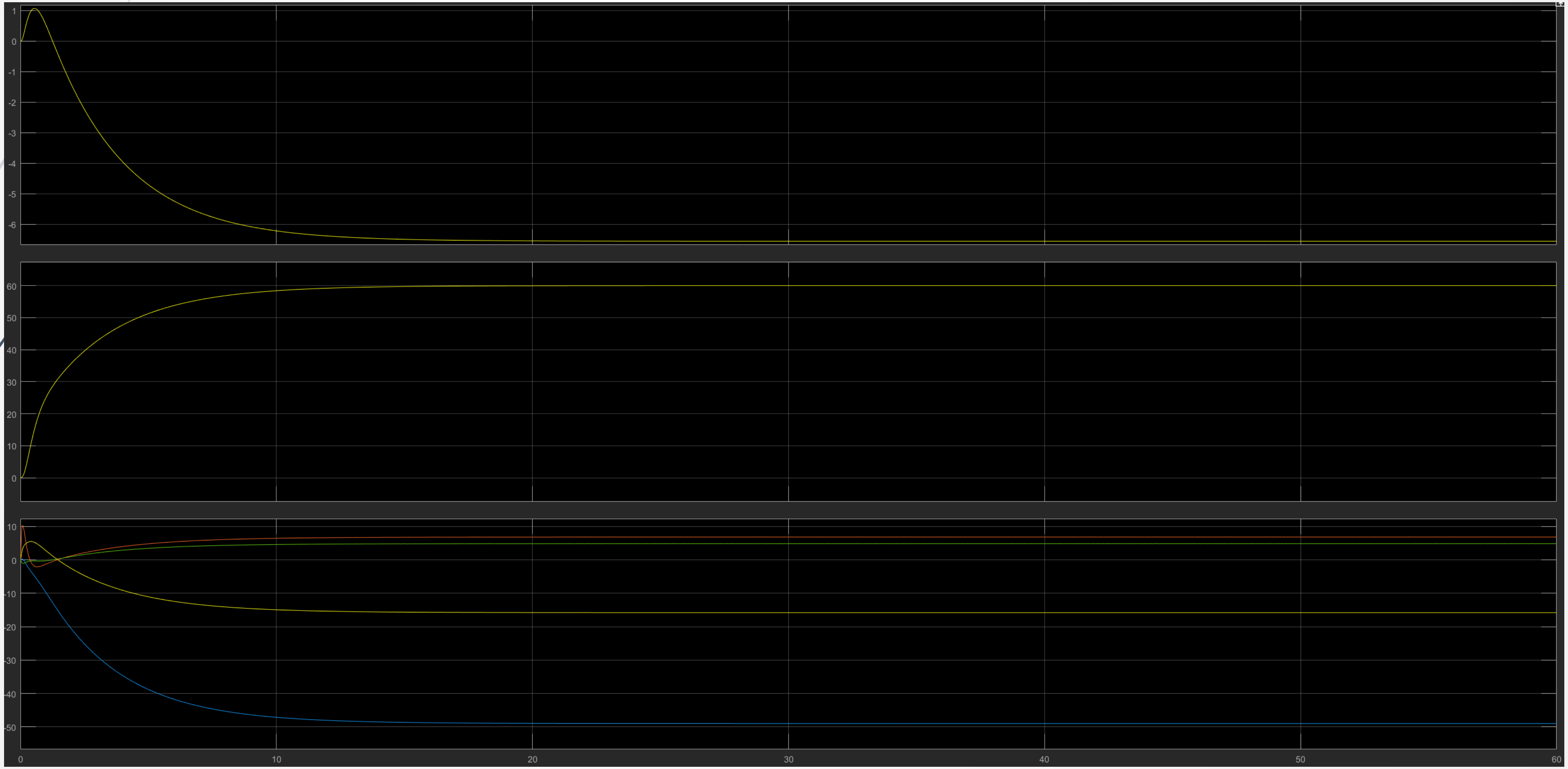
Design LQR Servo Control

- **SIMULINK design**



Design LQR Servo Control

- **Linear Response**



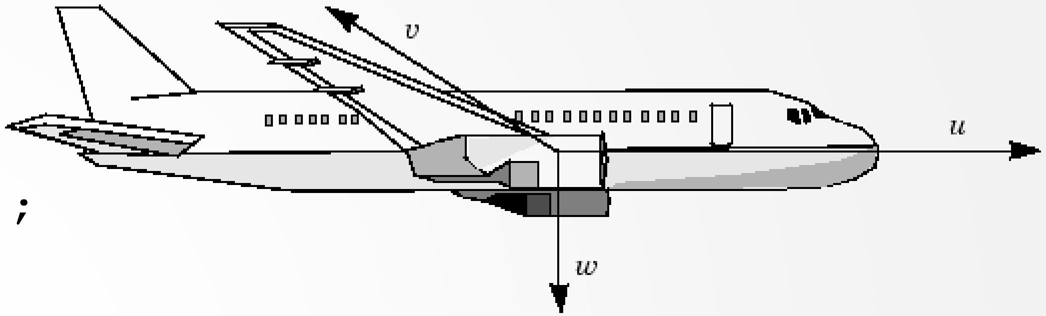
Design LQR Servo Control

- **Nonlinear Response**

Design LQR Servo Control

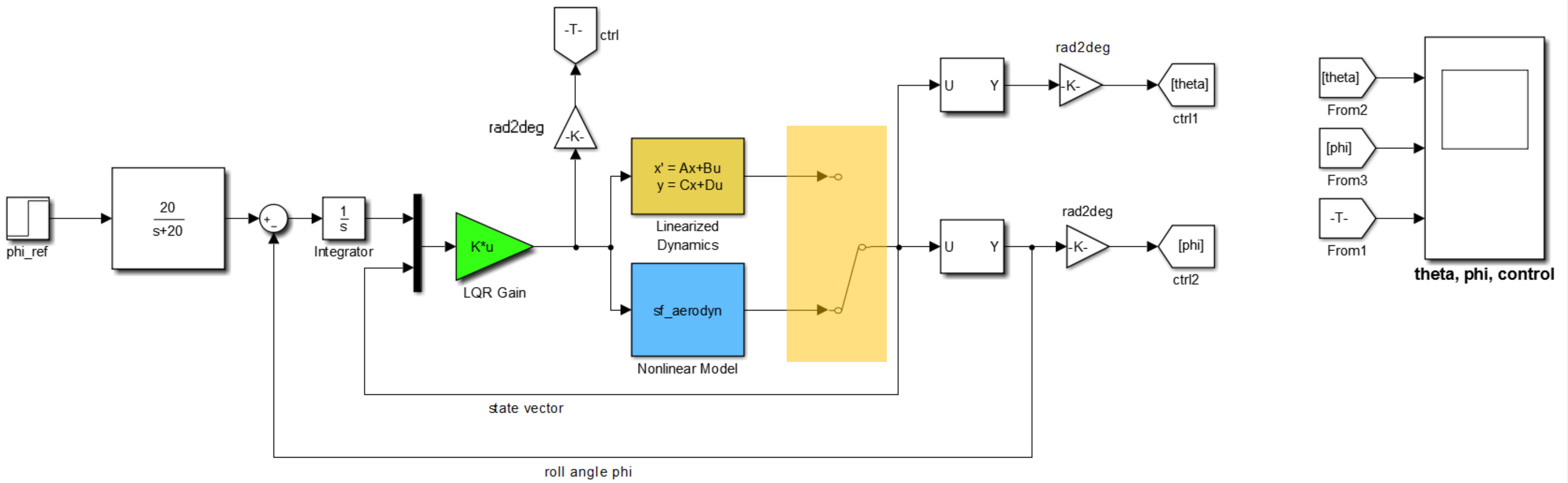
- **LQR Servo Control**

```
% Add integrator state dz/dt = -phi
A_aug = [zeros(1,8) -1; zeros(8,1) A15];
B_aug = [zeros(1,4) ; B];

% LQR gain synthesis
Q = blkdiag(1,0.1*eye(6),1000,1);
R = 100*diag([10,50,1,1]);
K_lqr = lqr(A_aug,B_aug,Q,R);
```



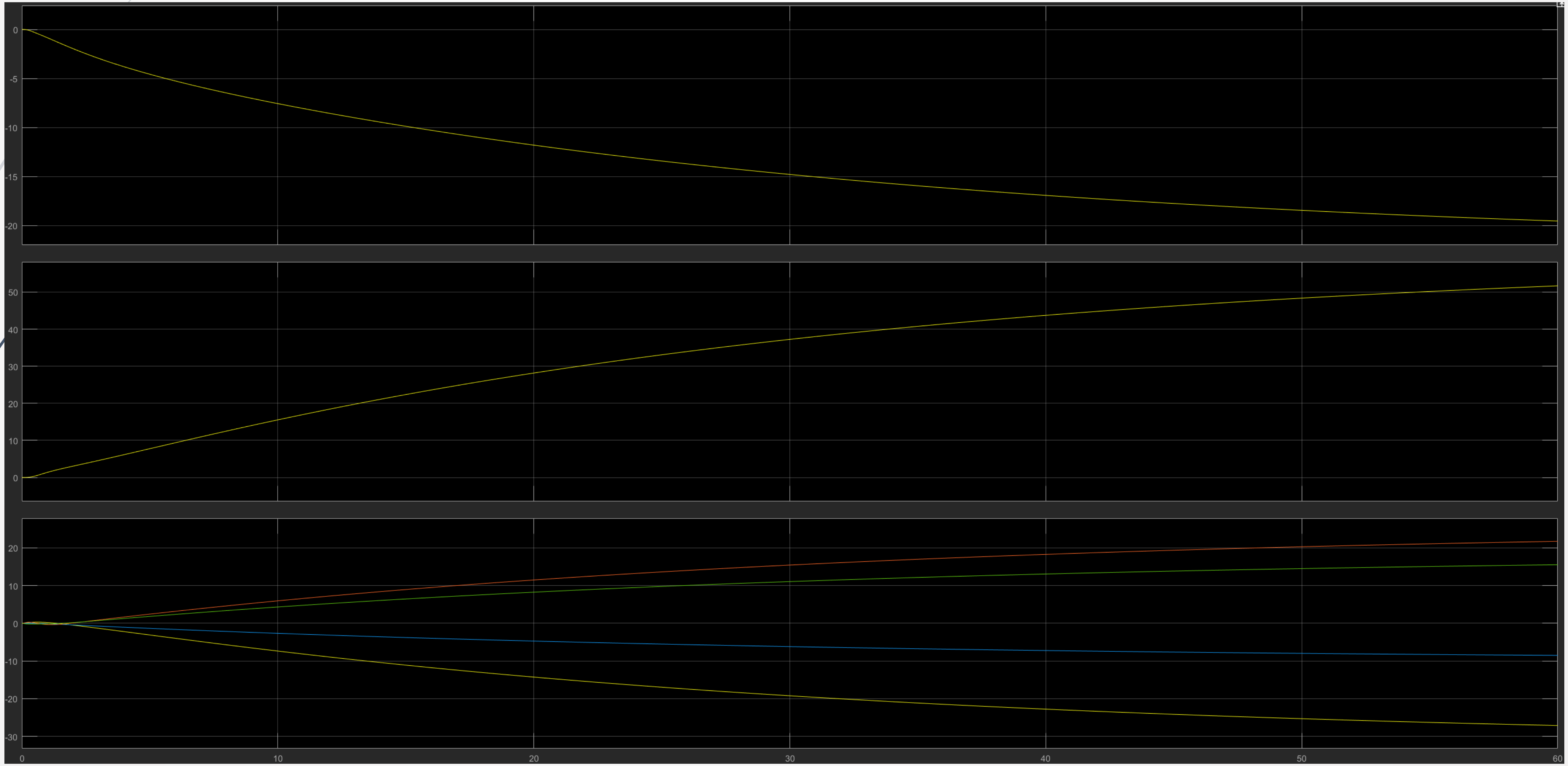
Design LQR Servo Control

- **SIMULINK design**



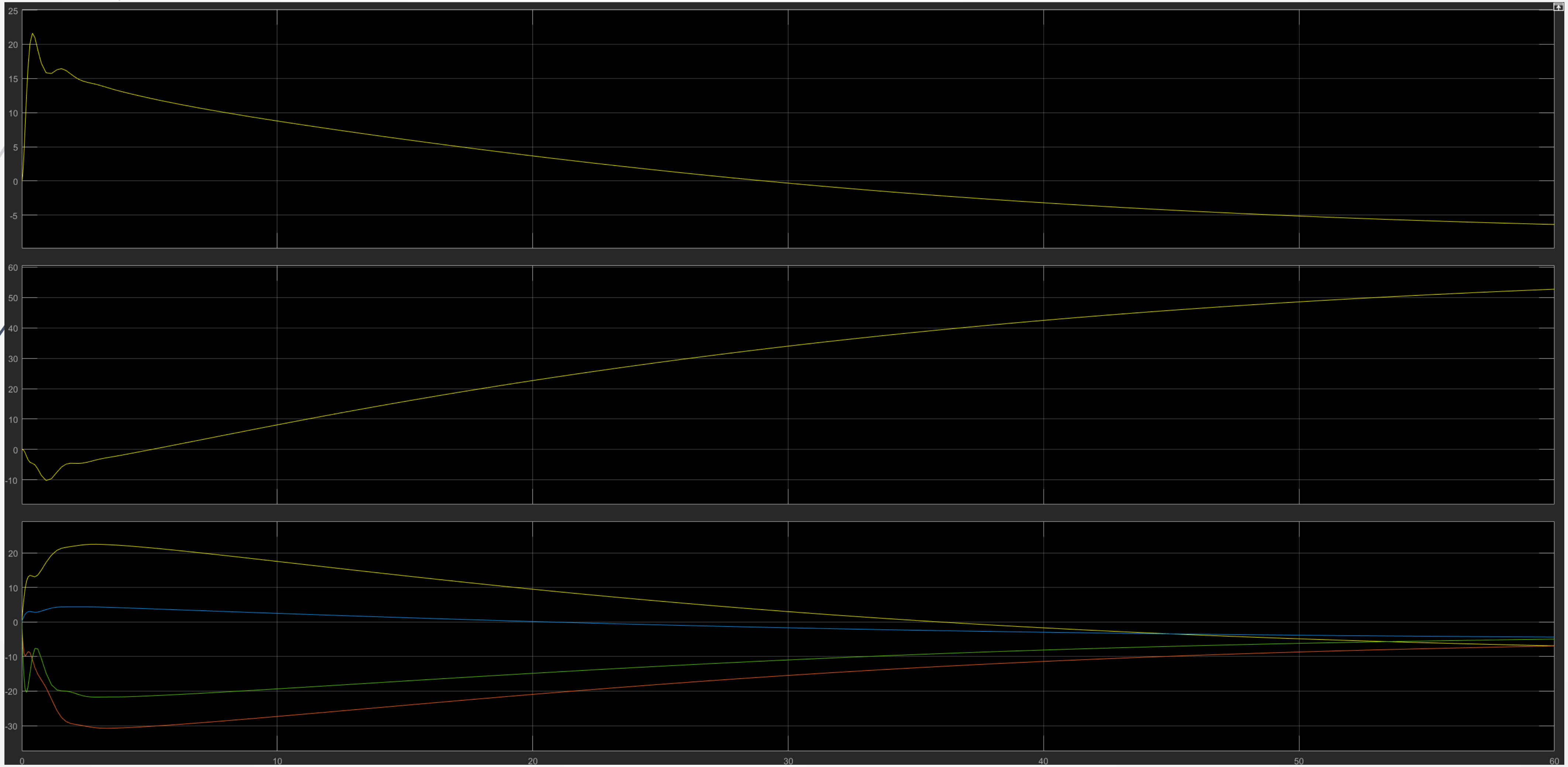
Design LQR Servo Control

- **Linear Response**



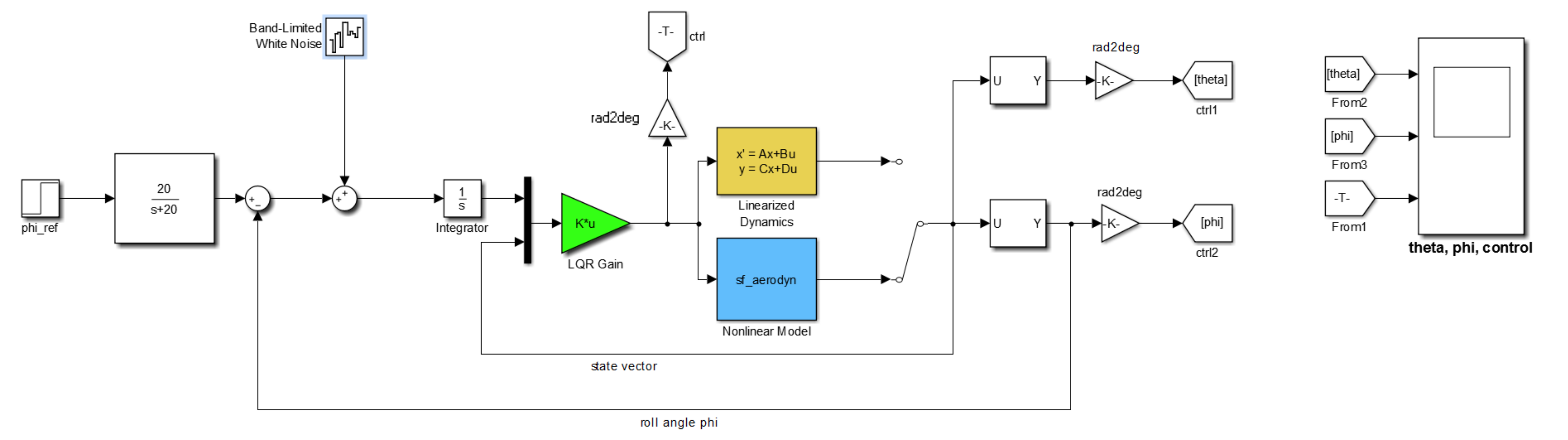
Design LQR Servo Control

- **Nonlinear Response**



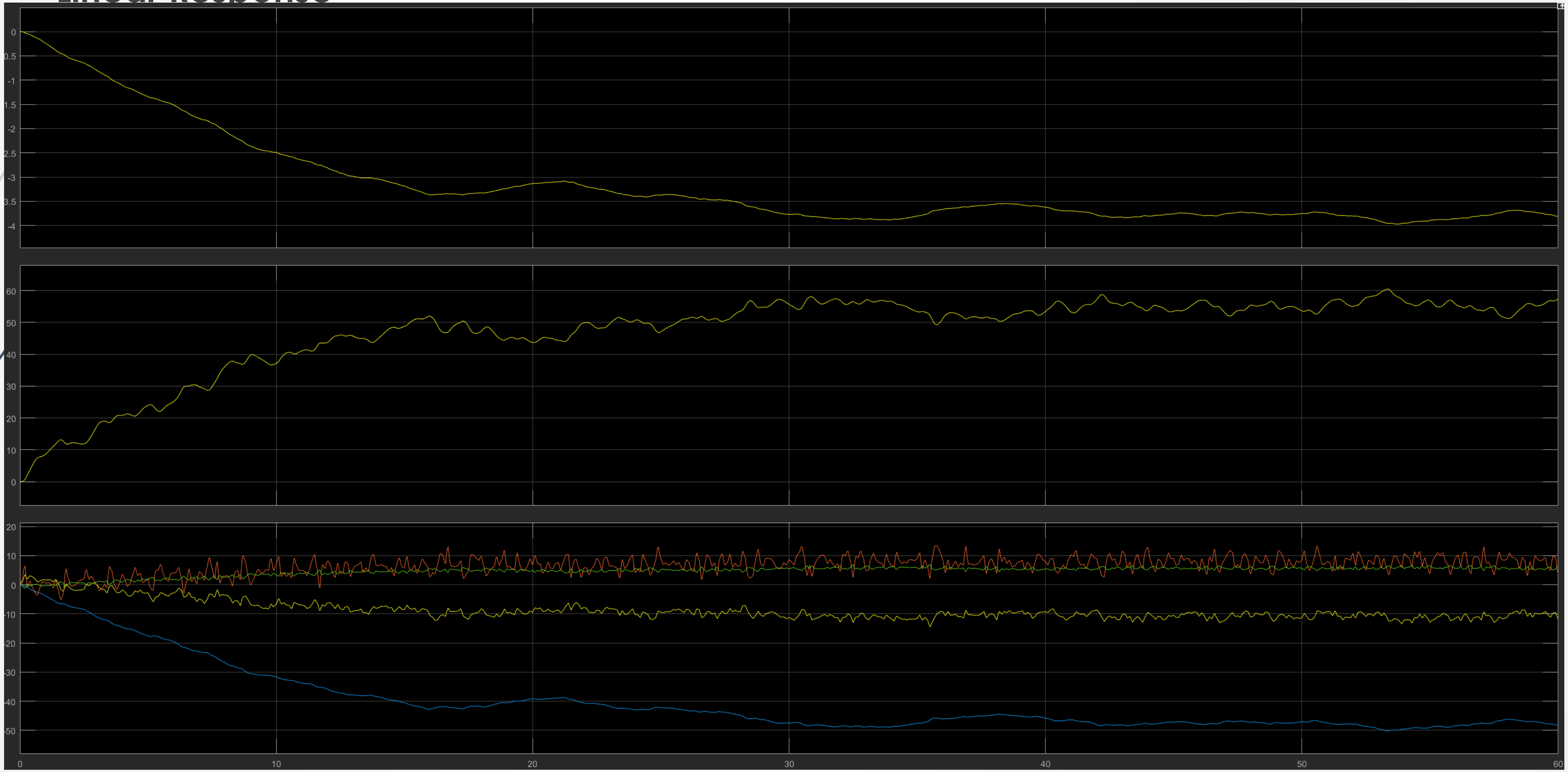
Design LQR Servo Control

- **SIMULINK design**



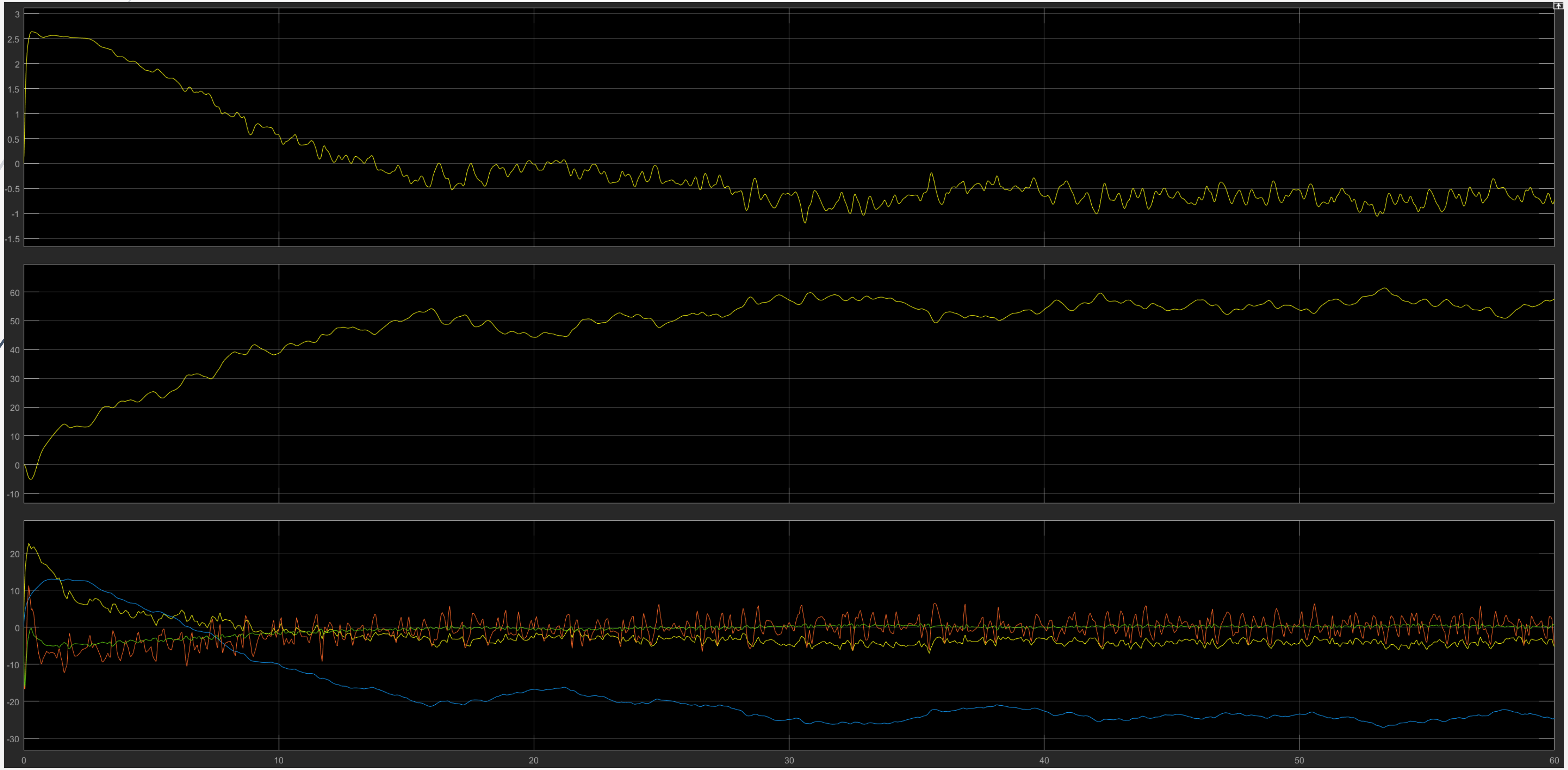
Design LQR Servo Control

- **Linear Response**



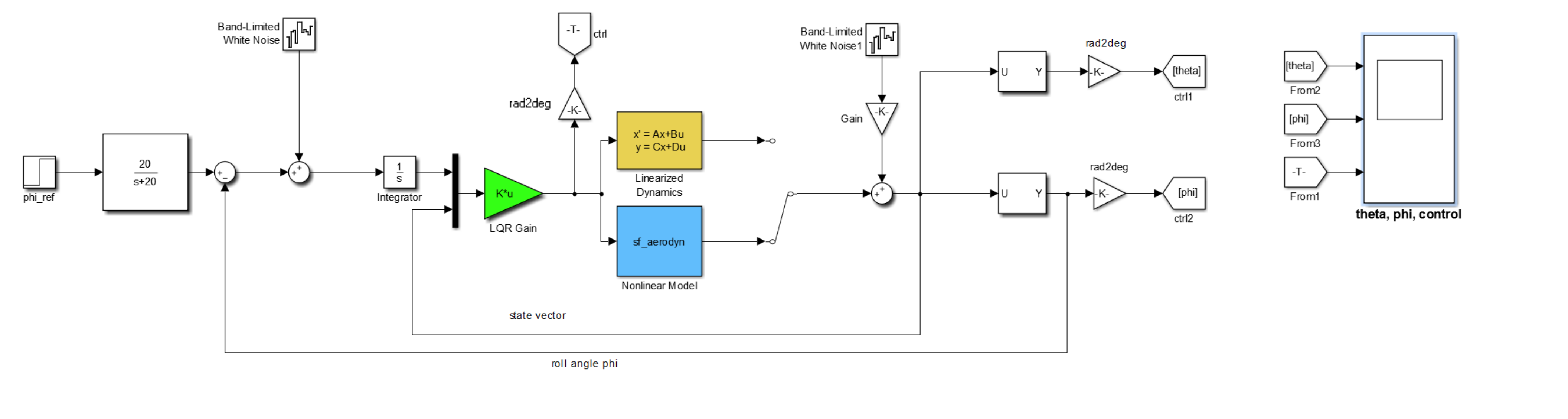
Design LQR Servo Control

- **Nonlinear Response**



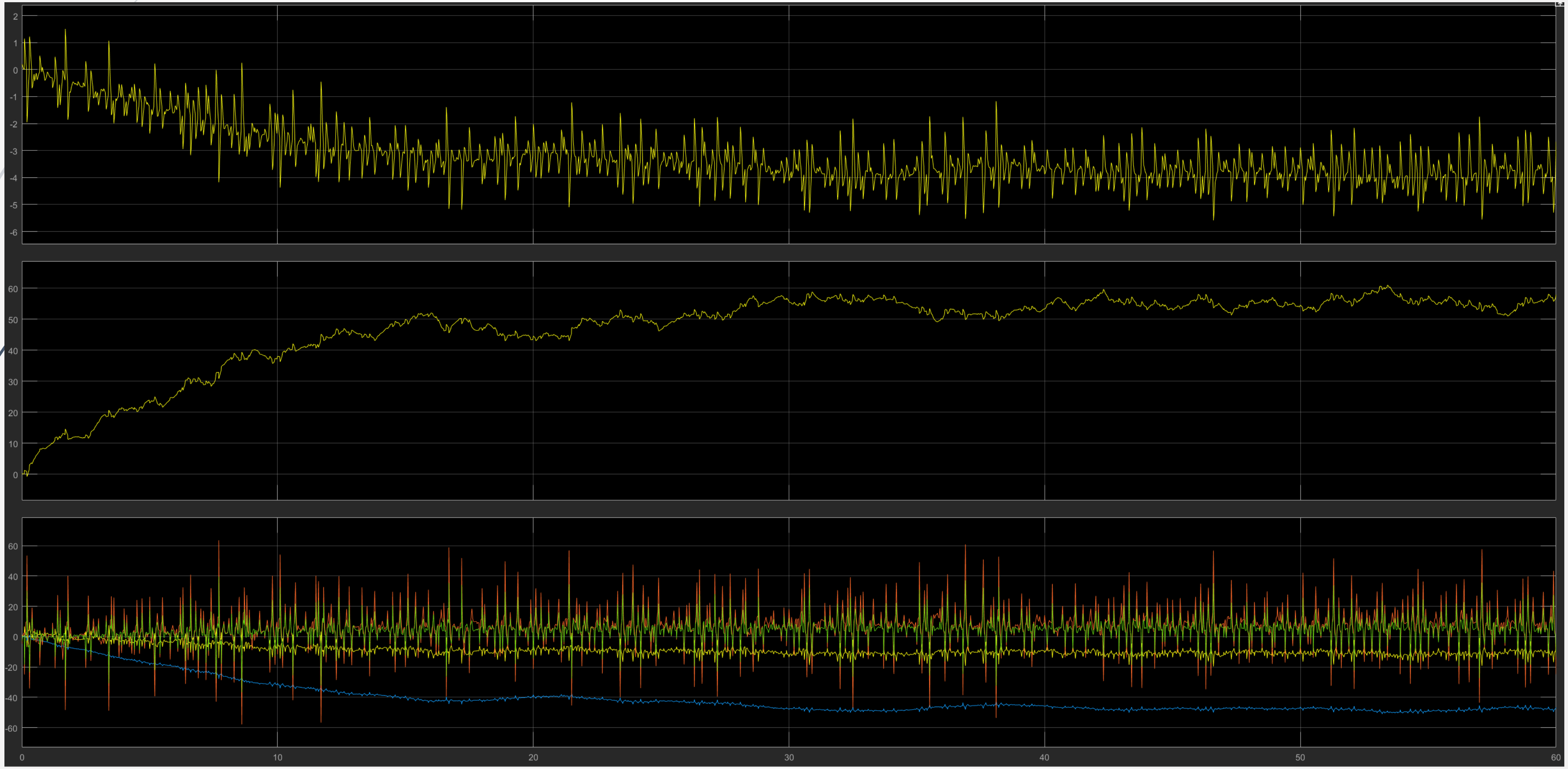
Design LQR Servo Control

- **SIMULINK design**



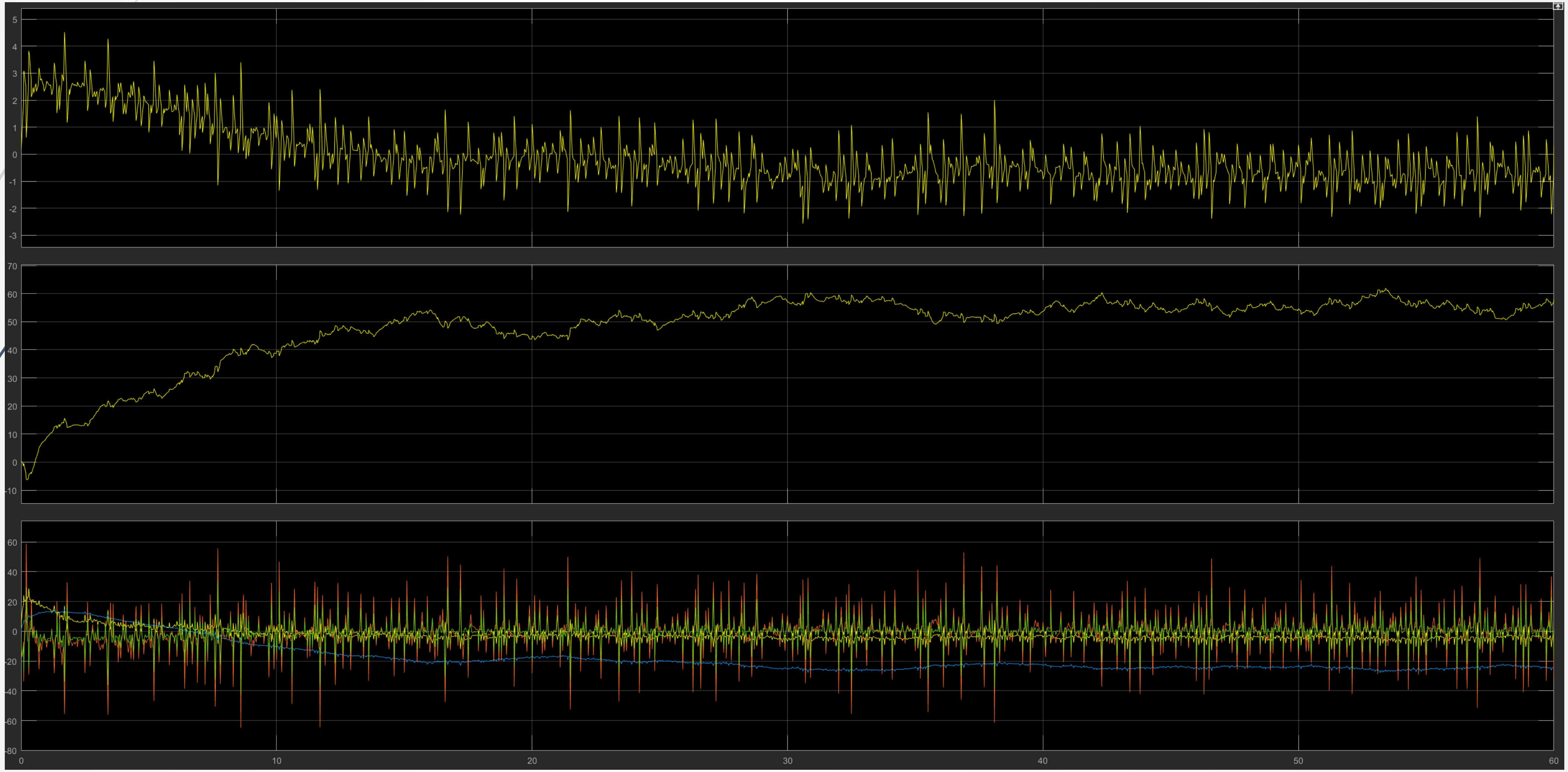
Design LQR Servo Control

- **Linear Response**



Design LQR Servo Control

- **Nonlinear Response**

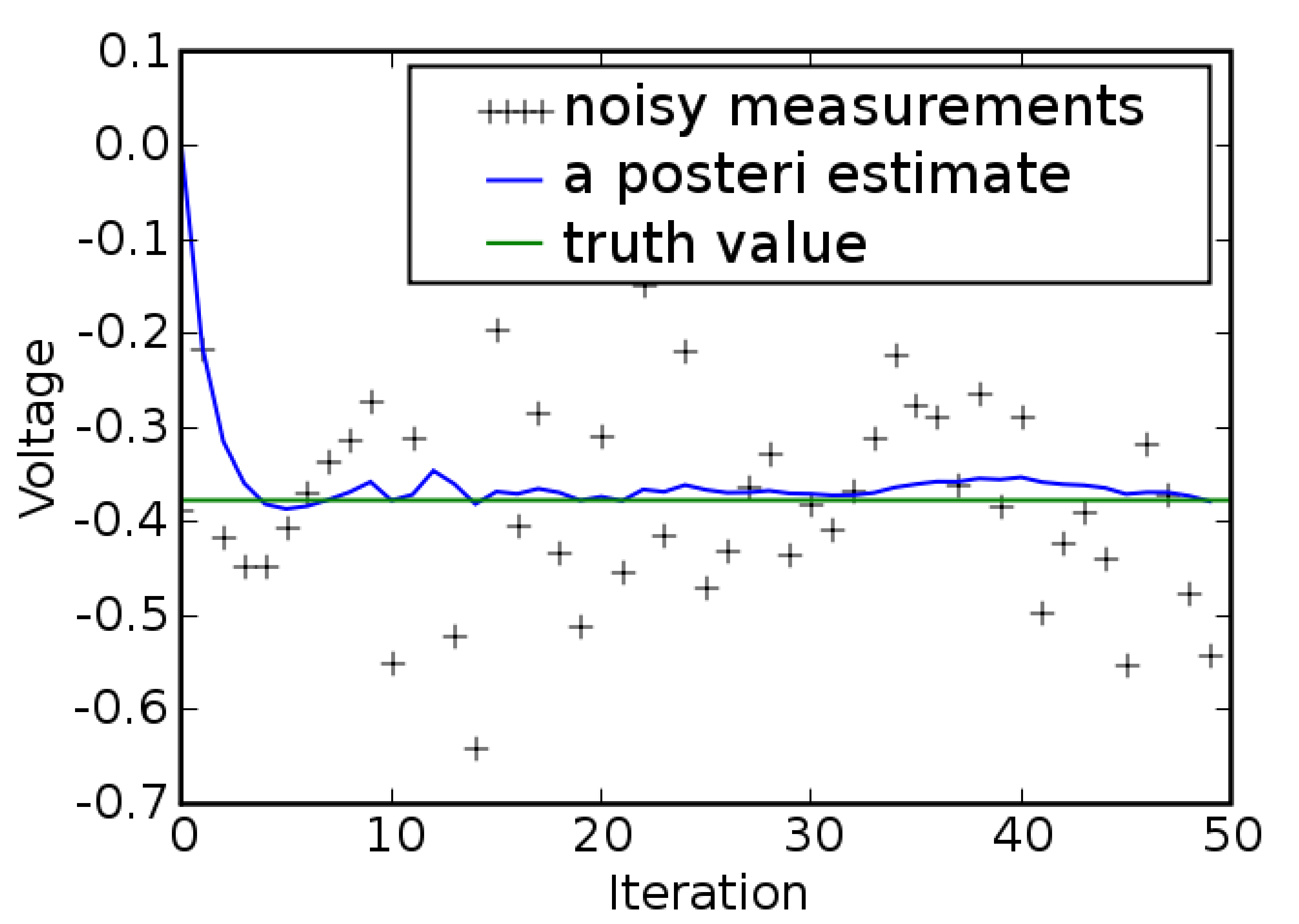


LQR in Python

- **Python implementation of LQR:**
  - For continuous systems
  - For discrete systems

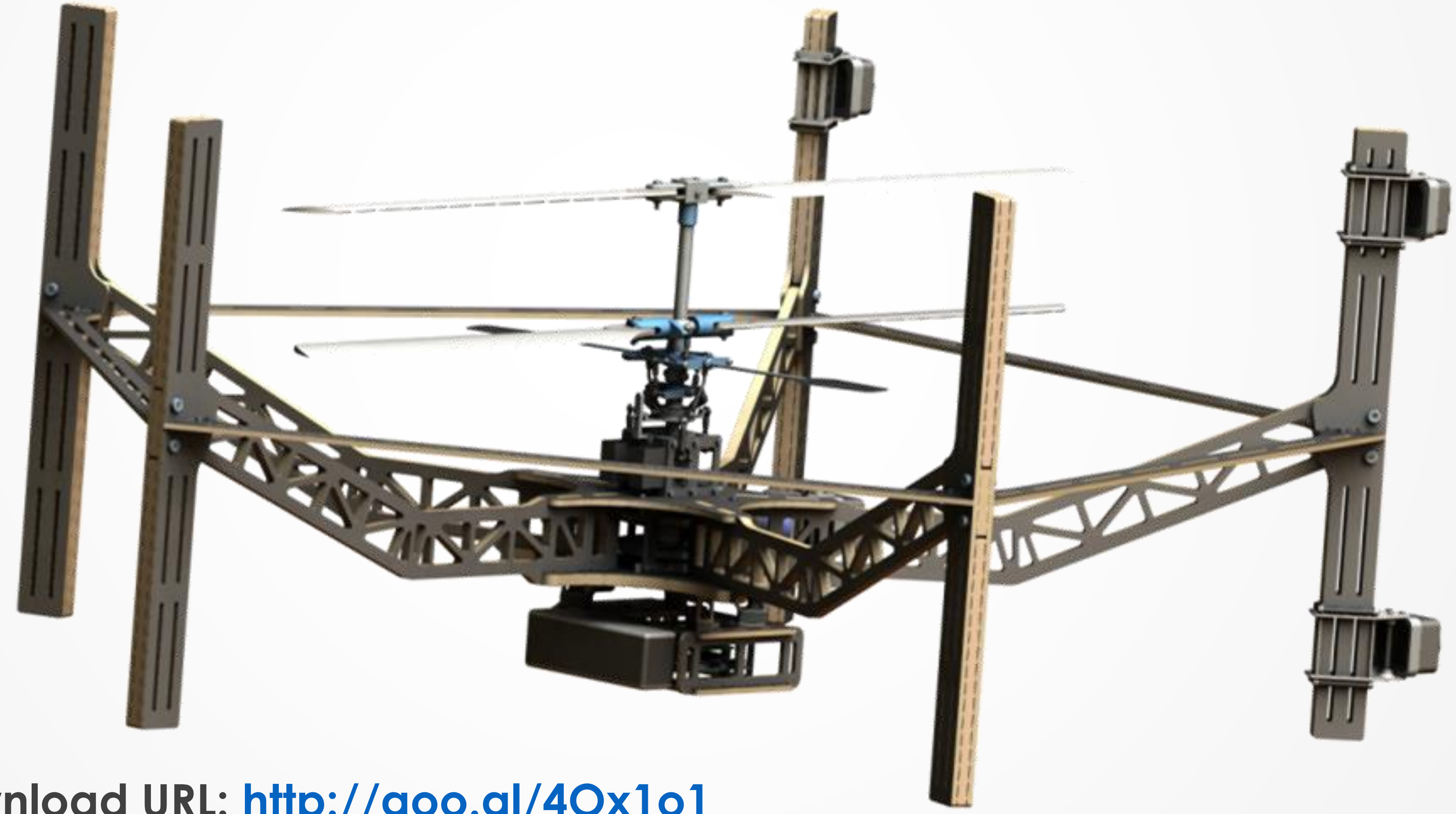
```python
from __future__ import division, print_function

import numpy as np
import scipy.linalg

def lqr(A,B,Q,R):
    """Solve the continuous time lqr controller.

    dx/dt = A x + B u

    cost = integral x.T*Q*x + u.T*R*u
    """
    #ref Bertsekas, p.151

    #first, try to solve the ricatti equation
    X = np.matrix(scipy.linalg.solve_continuous_are(A, B, Q, R))

    #compute the LQR gain
    K = np.matrix(scipy.linalg.inv(R)*(B.T*X))

    eigVals, eigVecs = scipy.linalg.eig(A-B*K)

    return K, X, eigVals

def dlqr(A,B,Q,R):
    """Solve the discrete time lqr controller.


    x[k+1] = A x[k] + B u[k]

    cost = sum x[k].T*Q*x[k] + u[k].T*R*u[k]
    """
    #ref Bertsekas, p.151

    #first, try to solve the ricatti equation
    X = np.matrix(scipy.linalg.solve_discrete_are(A, B, Q, R))

    #compute the LQR gain
    K = np.matrix(scipy.linalg.inv(B.T*X*B+R)*(B.T*X*A))

    eigVals, eigVecs = scipy.linalg.eig(A-B*K)

    return K, X, eigVals
```

Plot expected by your assignment



More Tools for Research

- **Coaxial Helicopter System Identification, Modeling and Control**



- **Download URL: http://goo.gl/4Ox1o1**

Find out more

- http://www.kostasalexis.com/pid-control.html
- http://www.kostasalexis.com/lqr-control.html
- http://www.kostasalexis.com/linear-model-predictive-control.html
- http://ctms.engin.umich.edu/CTMS/index.php?example=InvertedPendulum&section=ControlStateSpace
- http://www.kostasalexis.com/literature-and-links.html

Thank you!

Please ask your question!