ABSTRACT

A COMPARISON OF SOFTWARE ENGINES FOR SIMULATION OF CLOSED-LOOP CONTROL SYSTEMS

by
Sanket D Nikam

A wide array of control system design and simulation software engines is available in market. It includes MATLAB-Simulink, LabVIEW, Maple-MapleSim, Scilab-Scicos, VisSim and Mathematica-Control Professional Suite (CPS). Among all of them MATLAB-Simulink is dominant and widely used software engine. The main aim of this study is to implement different state space control methods for non-linear Furuta pendulum system in each one of them and to compare performance against MATLAB-Simulink.

Different parameters like learning curve, interoperability, flexibility, control design tools, documentation and tech support are considered for efficiency comparison. It is shown that MapleSim has multi-body intuitive physical modeling (acausal) approach faster than Simulink with unique control animation feature. It is found that MapleSim has the ability to generate differential equations from acausal modeling. It was verified that differential equations generated by MapleSim were similar to original equations. Scilab-Scicos is cost-efficient being open source engine with all control design and simulation capability similar to Matlab-Simulink. LabVIEW has better front end and back end for control design simulation at the cost of steep learning curve. VisSim has complete symbolic modeling approach with great flexibility and ease of learning. Mathematica's Control System Professional does not have symbolic modeling capability. It is observed that CPS has a cumbersome approach for modeling non linear systems.

A COMPARISON OF SOFTWARE ENGINES FOR SIMULATION OF CLOSED-LOOP CONTROL SYSTEMS

by
Sanket D Nikam

A Thesis
Submitted to the Faculty of
New Jersey Institute of Technology
in Partial Fulfillment of the Requirements for the Degree of
Master of Science in Electrical Engineering

Department of Electrical and Computer Engineering

May 2010

APPROVAL PAGE

A COMPARISON OF SOFTWARE ENGINES FOR SIMULATION OF CLOSED-LOOP CONTROL SYSTEMS

Sanket D Nikam

19 May 2010

Dr. Bernard Friedland, Dissertation Advisor — Date
Distinguished Professor of Electrical and Computer Engineering, NJIT

19 MAY 2010

Dr. David Haessig, Committee Member — Date
Adjunct Professor of Electrical and Computer Engineering, NJIT

5/19/2010

Dr. Mengchu Zhou, Committee Member — Date
Professor of Electrical and Computer Engineering, NJIT

BIOGRAPHICAL SKETCH

Author: Sanket Deepak Nikam

Degree: Master of Science

Date: May 2010

Undergraduate and Graduate Education:

- Master of Science in Electrical Engineering,
  New Jersey Institute of Technology, Newark, NJ, 2010

- Bachelor of Engineering in Instrumentation & Control,
  Vishwakarma Institute of Technology, Pune University, India, 2007

Major: Electrical Engineering

Dedicated to my parents, family members and teachers whose motivation has always been everlasting inspiration in my every endeavor...

ACKNOWLEDGMENT

I would like to express deep sense of gratitude towards Dr. Bernard Friedland for believing in my potential and providing me an opportunity to work on this thesis. I am really grateful to Dr. Friedland for motivating me to work on this thesis. His valuable guidance throughout the course of thesis is backbone of this success story. I would take this opportunity to thank Dr. David Haessig and Dr. Mengchu Zhou for being members of thesis committee and their guidance throughout the course of study.

I would like to specially thank Dr. Gilbert Lai, Mr. Ted Shapiro and Mr. Gavin Fitzpatrick from Maplesoft for their kind technical support and guidance during evaluation of MapleSim. I am really obliged to Mr. Peter Darnell from Visual Solutions for providing exclusive guidance regarding VisSim. Also, I would like to thank Dr. Ramine Nikoukhah from INRIA for his valuable advice during troubleshooting of Furuta Pendulum application in Scilab-Scicos. I would like to sincerely thank whole team from National Instruments including Dr. Jeannie Falcon, Mr. Tom Robbins and Mr. Andy Chang for resolving my complex queries related to control design and simulation in LabVIEW. Mr. Andy Dorsett and Dr. Aravind Hanasoge from Mathematica helped me a lot to learn more about Control System Professional suite. I would like to thank both of them for giving better insight about capabilities of Control System Professional. My sincerest gratitude to roommates and friends for their immense moral support. Last but not least I am very much indebted to my father Dr. Deepak Nikam and my mother Mrs. Vaijayanti Nikam because of whom I could pursue my graduate studies in United States of America.

TABLE OF CONTENTS

TABLE OF CONTENTS
(Continued)

LIST OF TABLES

LIST OF FIGURES

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF FIGURES
(Continued)

Figure **Page**

LIST OF SYMBOLS

φ	Arm Angle
θ	Pendulum Angle
$\dot{\varphi}$	Arm Velocity
$\dot{\theta}$	Pendulum Velocity

CHAPTER 1

INTRODUCTION

1.1 Objective

The objective of the investigations reported in this thesis is to compare control design and simulation efficiency of various software engines such as MATLAB-Simulink, LabVIEW, VisSim, Scilab-Scicos, Maple-MapleSim and Mathematica-CSP. For comparison study the Furuta pendulum with non-linear dynamics was chosen as comparison platform. MATLAB-Simulink was chosen as reference being widely used and dominant software engine for closed-loop control design and simulation. Different state space Control techniques such as Full State Feedback with pole placement, Full State Feedback with LQR and Kalman Full Order Observer were considered and were implemented in each of these software engines. Based on different parameters like learning curve, flexibility, control design tools, documentation, tech support and cost; control design and simulation results of these different engines were compared with MATLAB-Simulink.

1.2 Background Information

There are many software engines available for analysis and simulation of control and dynamic systems. MATLAB-Simulink has been most dominant and widely accepted software for control design, control simulation, analysis and modeling of dynamic systems. Other softwares have been evolved with different approaches for modeling, design and simulation of control for dynamic systems. Some software engines have

advantages and disadvantages over each other. Most of the control system engineers are not familiar with key features offered by other software engines other than MATLAB-Simulink. An attempt has been made in this thesis to highlight prominent features of other softwares in comparison with MATLAB-Simulink as far as modern control techniques such as state space Control are concerned. It is expected that comparison study in this thesis would help control engineers to understand capabilities, advantages and disadvantages of different control design and simulation softwares. Based on outcomes of this thesis it would be easier for control engineers to pick correct and efficient software engine as per their requirements of application. It would also serve as comprehensive feedback to vendors of these various software engines. It could be treated as case study to introspect these software engines for concerned vendors. For this case study Furuta Pendulum a Non-linear dynamic system with considerable amount of complexity has been used to demonstrate control design and simulation comparison study with reference to MATLAB-Simulink.

CHAPTER 2

DYNAMICS OF NON-LINEAR FURUTA PENDULUM

In this chapter, the physical description and dynamics of non-linear Furuta pendulum were presented. Physical descriptions consist of block diagram, physical arrangement and different physical parameters associated with Furuta pendulum. This chapter also focuses on equations of motion derived using Lagrangian method for Furuta pendulum.

2.1 Physical System Descriptions

The Furuta pendulum, or rotational inverted pendulum, consists of a driven arm which rotates in the horizontal plane and a pendulum attached to that arm which is free to rotate in the vertical plane. Figure 2.1 shows physical arrangement of Furuta pendulum system.

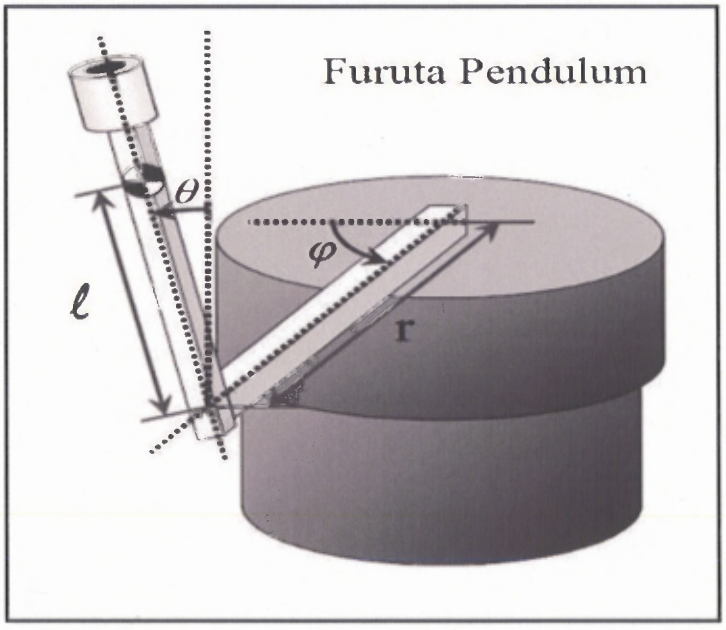


Figure 2.1 Furuta pendulum physical system.

The motion of the system is uniquely defined by the angular displacement φ of the arm from the reference point and the angle θ that the pendulum rod makes with respect to the vertical plane. The Furuta pendulum was first developed at the Tokyo Institute of Technology by Katsuhisa Furuta and his colleagues. The pendulum is underactutated and extremely non-linear due to the gravitational forces and the coupling arising from the Coriolis and centripetal forces. For the Furuta pendulum, m is the mass of the pendulum bob, J is the moment of inertia of the arm, r is the radius of the arm, L is the length of the pendulum rod, φ is the angular displacement of the arm, θ is the angle of the pendulum rod with respect to vertical plane and u=e - the motor voltage. [1]

2.2 Equations of Motion by Lagrangian Method

The equations governing the motion of a complicated mechanical system, such as an inverted pendulum on rotational arm, can be expressed very efficiently through the use of a method developed by the eighteenth-century French mathematician Lagrange. The differential equations that result from the use of this method are known as Lagrange's equations and are derived from Newton's laws of motion.

The fundamental principle of Lagrange's equations is the representation of the system by a set of generalized coordinates having been selected as φ and θ. After having defined the generalized coordinates, the kinetic energy T is expressed in terms of these coordinates and their derivatives $\dot{\varphi}$ and $\dot{\theta}$, and the potential energy V is expressed in terms of the generalized coordinates [2]. Next the Lagrangian function is formed,

$$L = T(\varphi, \theta, \dot{\varphi}, \dot{\theta}) - V(\varphi, \theta)$$

And finally, the Lagrange's equations for this system are written,

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\varphi}}\right) - \frac{\partial L}{\partial \varphi} = \tau \tag{2.1}$$

$$\frac{d}{dt}\left(\frac{\partial L}{\partial \dot{\theta}}\right) - \frac{\partial L}{\partial \theta} = 0 \tag{2.2}$$

$$K.E. = \frac{1}{2}mL^2\dot{\theta}^2 + \frac{1}{2}(J + mL^2\sin^2\theta)\dot{\varphi}^2 - mLr\dot{\varphi}\dot{\theta}\cos\theta$$

$$P.E. = mgL(\cos\theta - 1)$$

L=K.E. – P.E.

The Lagrangian function for Furuta Pendulum is written as,

$$L = \frac{1}{2}mL^2\dot{\theta}^2 + \frac{1}{2}(J + mL^2\sin^2\theta)\dot{\varphi}^2 - mLr\dot{\varphi}\dot{\theta}\cos\theta - mgL(\cos\theta - 1)$$

$$\frac{\partial L}{\partial \dot{\varphi}} = (J + mL^2\sin^2\theta)\dot{\varphi} - mLr\dot{\theta}\cos\theta$$

$$\frac{\partial L}{\partial \varphi} = 0$$

$$\frac{\partial L}{\partial \dot{\theta}} = mL^2\dot{\theta} - mLr\dot{\varphi}\cos\theta$$

$$\frac{\partial L}{\partial \theta} = mL^2\sin\theta\cos\theta\dot{\varphi}^2 + mLr\dot{\varphi}\dot{\theta}\sin\theta + mgL\sin\theta$$

$$(1) \Rightarrow (J + mL^2 \sin^2\theta)\ddot{\varphi} - mLr\cos\theta\ddot{\theta} + mL^2 \sin 2\theta\dot{\varphi}\dot{\theta} + mLr\sin\theta\dot{\theta}^2 = -a\dot{\varphi} + u$$

$$(2) \Rightarrow -mLr\cos\theta\ddot{\varphi} + mL^2\ddot{\theta} - \frac{mL^2}{2}\sin 2\theta\dot{\varphi}^2 - mgL\sin\theta = 0$$

Writing the above equations in matrix form [2],

$$\begin{bmatrix} J + mL^2\sin^2\theta & -mLr\cos\theta \\ -mLr\cos\theta & mL^2 \end{bmatrix}\begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} -mLr\sin\theta\dot{\theta}^2 - mL^2\sin 2\theta\dot{\varphi}\dot{\theta} - a\dot{\varphi} + u \\ \frac{mL^2}{2}\sin 2\theta\dot{\varphi}^2 + mgL\sin\theta \end{bmatrix} \tag{2.3}$$

$$M\begin{bmatrix} \ddot{\varphi} \\ \ddot{\theta} \end{bmatrix} = \begin{bmatrix} f1 \\ f2 \end{bmatrix}$$

$$M\ dV = \begin{bmatrix} f1 \\ f2 \end{bmatrix}$$

$$where\ f1 = -mLr\sin\theta\dot{\theta}^2 - mL^2\sin 2\theta\dot{\varphi}\dot{\theta} - a\dot{\varphi} + u$$

$$f2 = \frac{mL^2}{2}\sin 2\theta\dot{\varphi}^2 + mgL\sin\theta$$

dV = acceleration of the arm and the pendulum rod

For control design and simulation purposes following physical parameter values were considered:

J=0.001N-m-sec^2 L=0.2m r=0.3m m=0.05kg g=9.8m/sec^2

Equation 2.3 was implemented in each software engine as discussed in Chapter 3.

CHAPTER 3

RESULTS OF VARIOUS SOFTWARE ENGINES

Chapter 3 deals with control design and simulation steps involved in implementation of State space control techniques like Full State Feedback with pole placement, Full State Feedback with LQR and Full Order Observer with Kalman Filter. This chapter presents control design and simulation results of various softwares engines such as MATLAB-Simulink, LabVIEW, Scilab-Scicos, Maple-MapleSim, VisSim and Mathematica-Control System Professional (CSP) suite.

3.1 MATLAB-Simulink

MATLAB is a high-level technical computing language and interactive environment for algorithm development, data visualization, data analysis, and numeric computation. Simulink is an environment for multi domain simulation and Model-based design for dynamic and embedded systems. MATLAB has dedicated Control System Toolbox which has been used to design state space control for the Furuta pendulum. It provides an interactive graphical environment and a customizable set of block libraries that let you design, simulate, implement, and test a variety of time-varying systems, including communications, controls, signal processing, video processing, and image processing. Section 3.1 explains in detail open loop analysis as well as closed loop state space control analysis for non-linear Furuta pendulum system.

3.1.1 Open Loop Analysis

Open loop analysis starts with modeling of Furuta pendulum system in Simulink based on available dynamics which are explained in Chapter 2. Simulink software models, simulates, and analyzes dynamic systems. It enables you to pose a question about a system, model the system, and see what happens. With Simulink, it is easy to build models from scratch, or modify existing models to meet system needs. Simulink supports linear and non-linear systems, modeled in continuous time, sampled time, or a hybrid of the two.

Simulink provides a graphical user interface (GUI) for building models as block diagrams, allowing drawing models as with pencil and paper. Simulink also includes a comprehensive block library of sinks, sources, linear and non-linear components, and connectors. If these blocks do not meet system needs, however, it also allows creating customized blocks. The interactive graphical environment simplifies the modeling process, eliminating the need to formulate differential and difference equations in a language or program. Model-based design is a process that enables faster, more cost-effective development of dynamic systems, including control systems. In Model-based design, a system model is at the center of the development process, from requirements development, through design, implementation, and testing. The model is an executable specification that is continually refined throughout the development process. After model development, simulation shows whether the model works correctly. [3]

Initial steps of this process are performed outside of the Simulink software before building Furuta pendulum model. Furuta pendulum system has been defined in Section 2.1. In Section 2.2 whole system has been developed in terms of mathematical

equations of motion with associated system parameters. In order to model Furuta pendulum in Simulink it is necessary to create M-file (furuta.m) containing function 'furuta'. Function 'furuta' defines mathematical equations for acceleration components (dv) related to arm and pendulum. M-file is a script containing MATLAB commands. MATLAB has separate Editor to create and debug M-files, which are programs to run MATLAB functions. The Editor provides a graphical user interface for text editing as well as for M-file debugging. It also contains Furuta pendulum system parameters and definition of state variables.

M-file (inside MATLAB Fcn block in Figure 3.1.1):-

```
%%  function dv=furuta(x)
function dv=furuta(x)
%%  Data %%
J=.001;
L=.2;
r=.3;
m=.05;
g=9.8;
c1=m*L*r;
c2=m*L^2;
c3=c2/2;
c4=m*g*L;
a=.01;
%% end of data
%%%%%%%%%%%% definitions of variables
ph=x(1);      % phi not used
th=x(2);      % theta
dph=x(3);     % phi dot
dth=x(4);     % theta dot
%%%%%%%%%%%  right-hand side
f1=-c1*sin(th)*dth^2 -c2*sin(2*th)*dth*dph - a*dph + u ;
f2=c3*sin(2*th)*dph^2 +c4*sin(th);
f=[f1;f2];
M=[J+c2*(sin(th))^2+m*r^2,-c1*cos(th);
      -c1*cos(th),c2];
dv=inv(M)*f ;
end
```

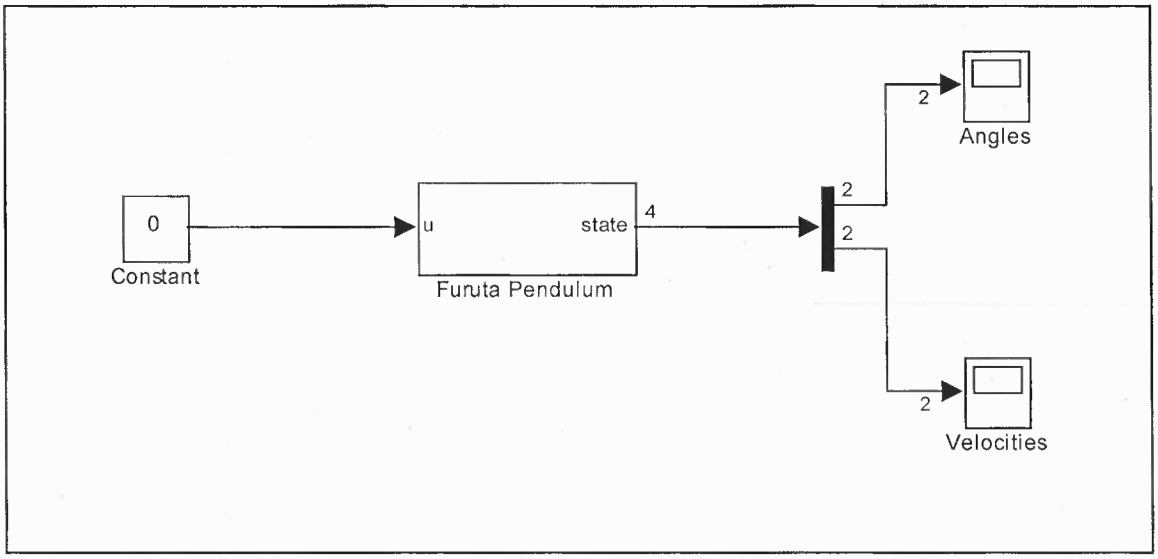


Figure 3.1 Simulink open loop block diagram.

Figure 3.1 shows Simulink block diagram for open loop model of Furuta pendulum system. It consists of Furuta pendulum subsystem. A subsystem is a set of blocks that have been replaced by a single block called a Subsystem block. A Subsystem block represents a subsystem of the system that contains it. The number of input ports drawn on the Subsystem block icon corresponds to the number of in port blocks in the subsystem. Similarly, the number of output ports drawn on the block corresponds to the number of out port blocks in the subsystem.

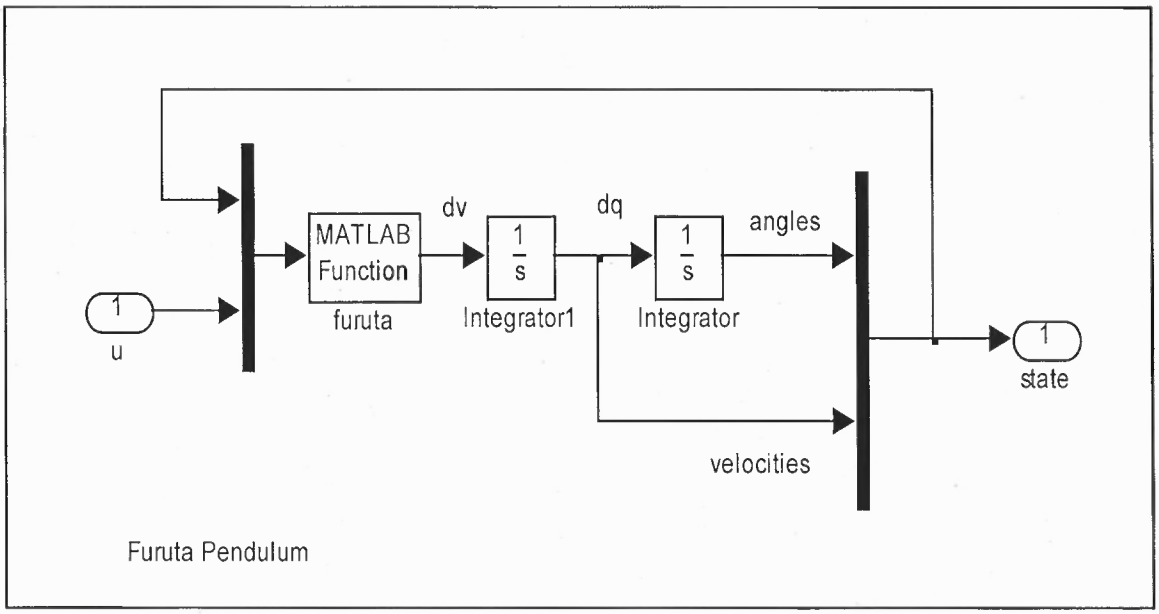


Figure 3.2 Simulink model for non linear Furuta pendulum system.

Figure 3.2 shows Simulink model for non linear Furuta pendulum dynamic system. It consists of various Simulink blocks like MATLAB Fcn block, Integrators and Mux. The MATLAB Fcn block applies the specified MATLAB 'furuta' function to the input. The output of the function must match the output dimensions of the block or an error occurs. This MATLAB Fcn block contains MATLAB function 'furuta' defined earlier in furuta.m file. The dialogue box of MATLAB Fcn block has been used to specify MATLAB function 'furuta'.

The Integrator block outputs the integral of its input at the current time step. The Integrator equation represents the output of the block y as a function of its input u and an initial condition y_0, where y and u are vector functions of the current simulation time t. As shown in Figure 3.2 Simulink model of the Furuta pendulum employs two cascaded Integrator blocks to give positions and velocities associated with arm and pendulum respectively. Input to first Integrator block is function 'dv' which is matrix of dimension

2*1. Output of first integrator gives velocities of arm and pendulum. Similarly, output of second Integrator block gives positions of arm and pendulum. The Mux block combines its inputs into a single vector output. An input can be a scalar or vector signal. All inputs must be of the same data type and numeric type. The elements of the vector output signal take their order from the top to bottom, or left to right, input port signals. So output of Mux block after second integrator is a state vector containing arm angle, pendulum angle, arm velocity and pendulum velocity. It feeds back this vector along with control signal 'u' to MATLAB Fcn using another Mux block. The Scope block displays its input with respect to simulation time. [3]

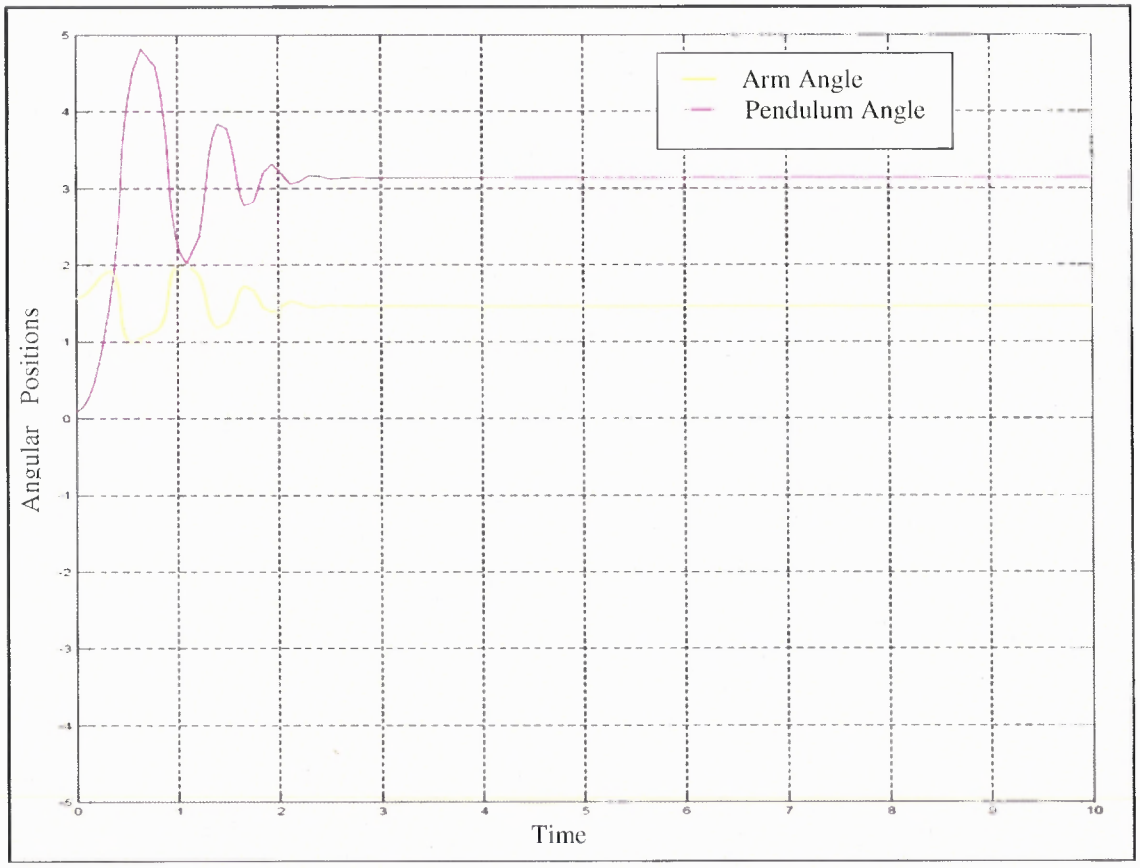


Figure 3.3 Simulink open loop output for the arm angle and the pendulum angle with initial conditions [1.57, 0.1] on angles.

Once modeling of Furuta Pendulum system with Simulink has been done, system is all set for running simulation. The simulation was run for 10 seconds. As shown in Figure 3.3, transient responses for the positions of the arm and the pendulum have been obtained. As per initial condition the arm position takes off from 1.57 radians and after few oscillations it settles down to its stable vertically downward position. The pendulum rod starts with 0.1 radians as per given initial condition and it moves through angle of 3.14 radians and it settles there after tumbling down from the initial position. Since there is no control in this simulation, the arm could not balance the inverted pendulum with the given initial conditions. Initial conditions are specified on second Integrator i.e., [1.57, 0.1] which gives out positions. Open loop simulation validated facts about positions of arm and pendulum rod as per requirements.

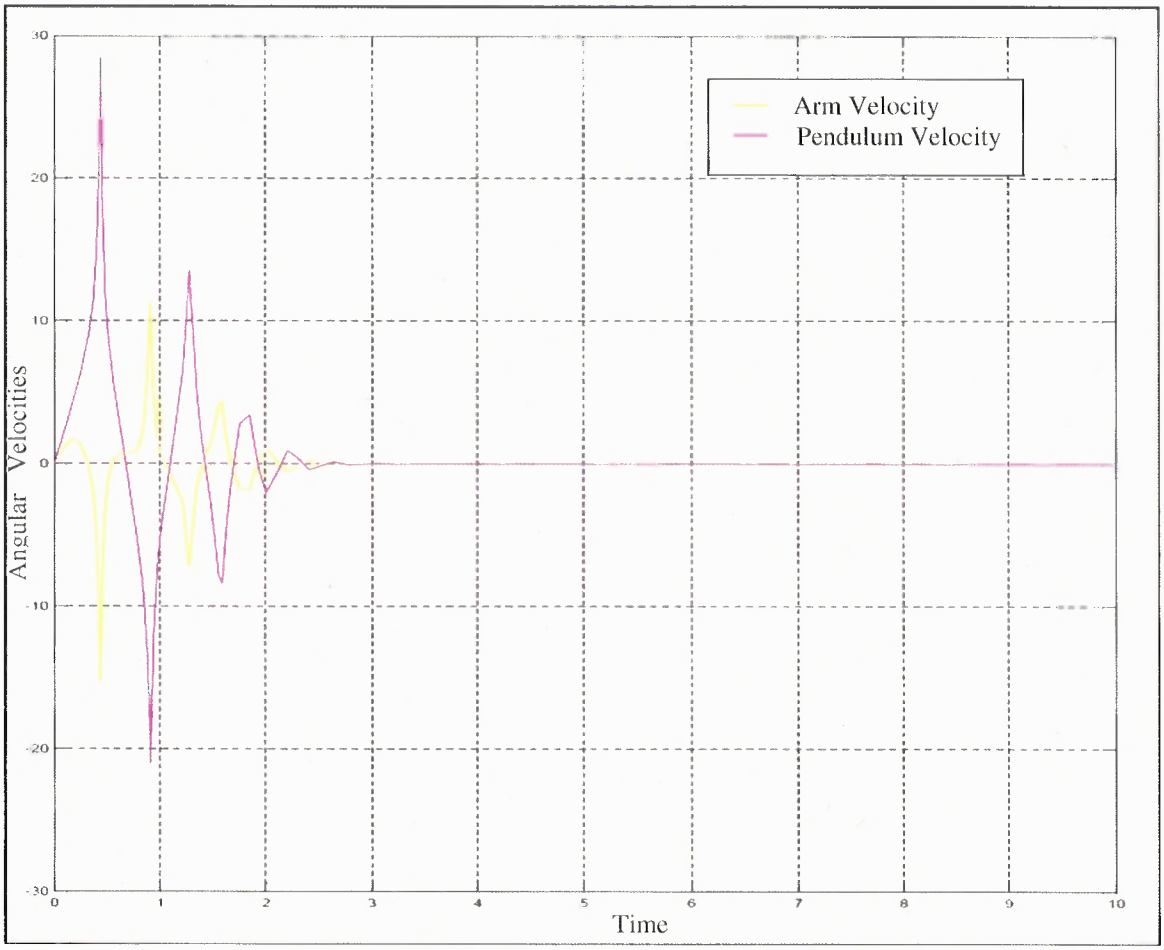


Figure 3.4 Simulink open loop output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0.1] on angles.

Figure 3.4 shows transient responses for arm and pendulum velocities which have been obtained after running simulation for 10 seconds. As per Furuta pendulum open loop behavior both arm and inverted pendulum should settle down after some time. As obtained in Figure 3.4 both arm velocity and pendulum velocity are settled to zero approximately after 2.7 seconds with few initial vigorous oscillations. The velocity behavior of Furuta pendulum has been validated with Simulink open loop simulation.

Linearization:

MATLAB Control System Toolbox code to obtain the linearized state space dynamics of the system is 'linmod'. Linmod computes a linear state space model by linearizing each block in a model individually. Following command has been executed to obtain linearized State space model of Furuta Pendulum. [4] [5]

```
>>[A,B,C,D]=linmod('FurutaDyn')
```

'Linmod' obtains linear models from systems of ordinary differential equations described as Simulink models. Inputs and outputs are denoted in Simulink block diagrams using Inport and Outport blocks. The linearized dynamics of the Simulink open loop 'FurutaDyn' model as shown in Figure 3.1.2 are obtained as below:

$$A=\begin{pmatrix}0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\\ 0 & 147 & -10 & 0\\ 0 & 269.5 & -15 & 0\end{pmatrix} \qquad B=\begin{pmatrix}0\\ 0\\ 1000\\ 1500\end{pmatrix}$$

$$C=\begin{pmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 0 & 0 & 0 & 1\end{pmatrix} \qquad D=\begin{pmatrix}0\\ 0\\ 0\\ 0\end{pmatrix}$$

`[A,B,C,D] = linmod(' FurutaDyn ', x, u)` obtains the linearized model of system around an operating point with the specified state variables x and the input u. Since x and u are omitted, the default values are zero.

Stability:

The stability of a system is determined by the location of its poles. MATLAB command `eig(A)` returns a vector of the eigen values of matrix `A`. [5]

```
>> eig(A)

ans =

         0

  -21.4831

   13.2097

   -1.7267
```

Since one of the eigen values is in the right half of s-plane the Furuta pendulum is unstable, confirming the physics. The rank of Controllability matrix is equal to the order of A. Hence there is no uncontrollable state and the system is completely controllable. The rank of Observability matrix is equal to 4 i.e., the order of A. Hence there is no unobservable state and the system is completely observable. Since Furuta pendulum system is completely controllable as well as observable it is all set for closed loop analysis with state space control techniques.

3.1.2 Full State Feedback design by Pole Placement

Section 3.1.2 presents Full State Feedback control design with Pole Placement for the Furuta pendulum system. It also shows simulation results obtained with Pole Placement control design. For Pole Placement given the multi-input Furuta pendulum system and a vector 'ecl' of desired self-conjugate closed-loop pole locations, MATLAB command `'place'` computes a gain matrix 'G' such that the state feedback u= -Gx places the closed-loop poles at the locations of 'ecl'. In other words, the eigen values of A-BG match the entries of 'ecl'. Complete M-file to compute Full State Feedback gain 'G' is as shown below. [5]

M-file:

```
%%%% Linearized Model Using linmod %%%%%%%
[A,B,C,D]=linmod('FurutaDyn')
%%%%%%%%%%%% Pole Placement %%%%%%%%%%%%%
eig(A)
ecl=[-10+10j -10-10j -1 -5.56]
G=place(A,B,ecl)
Output:
G =
   -0.0227    0.4193   -0.0390    0.0371
```

G = place(A,B,ecl) computes a feedback gain matrix 'G' that achieves the desired closed-loop pole locations -10+10j, -10-10j, -1, and -5.56, assuming all the inputs of the plant are control inputs. The length of 'ecl' must match the row size of A. The MATLAB command `'place'` works for multi-input systems and is based on an algorithm that uses the extra degrees of freedom to find a solution that minimizes the sensitivity of the closed-loop poles to perturbations in A or B. [6]

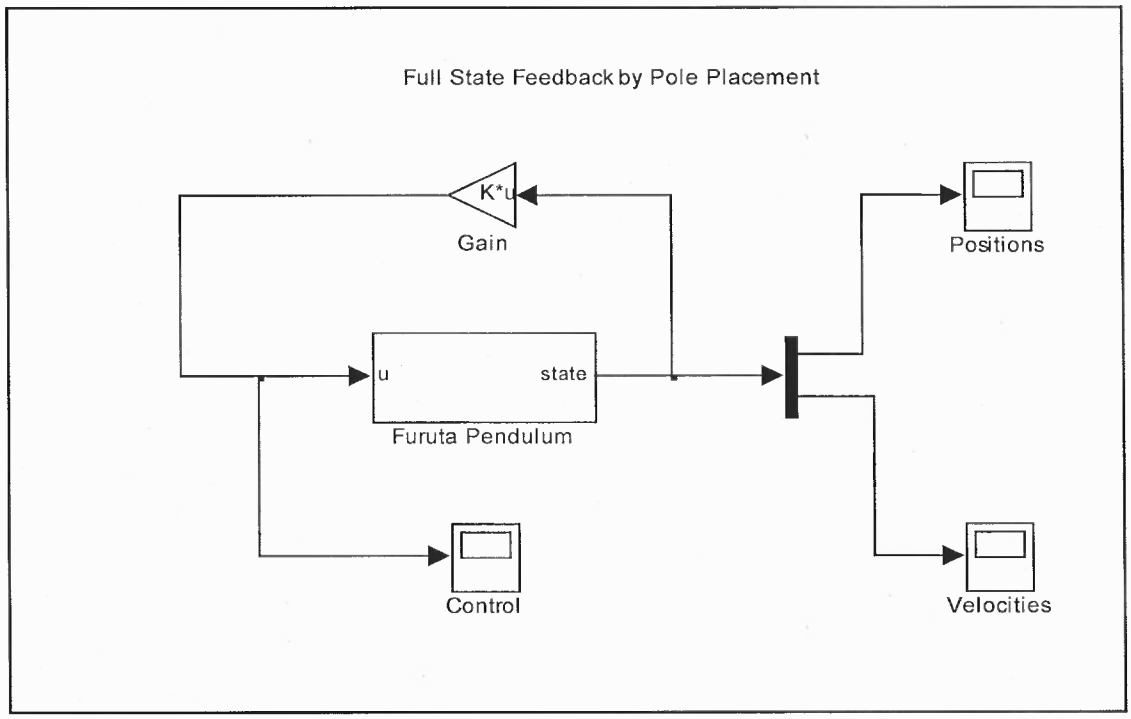


Figure 3.5 Simulink block diagram for Full State Feedback design by Pole Placement.

Simulink block diagram for Pole Placement control is as shown in Figure 3.5. Furuta pendulum subsystem contains non linear model of Furuta system. The gain block multiplies the input by a constant value (gain). The input and the gain can each be a scalar, vector, or matrix. The value of the gain has been specified in the gain parameter. The multiplication parameter lets you specify element-wise or matrix multiplication. For matrix multiplication, this parameter also lets you indicate the order of the multiplicands.

The gain vector G = [-0.0227 0.4193 -0.0390 0.0371] will be multiplied with state vector x. [6]

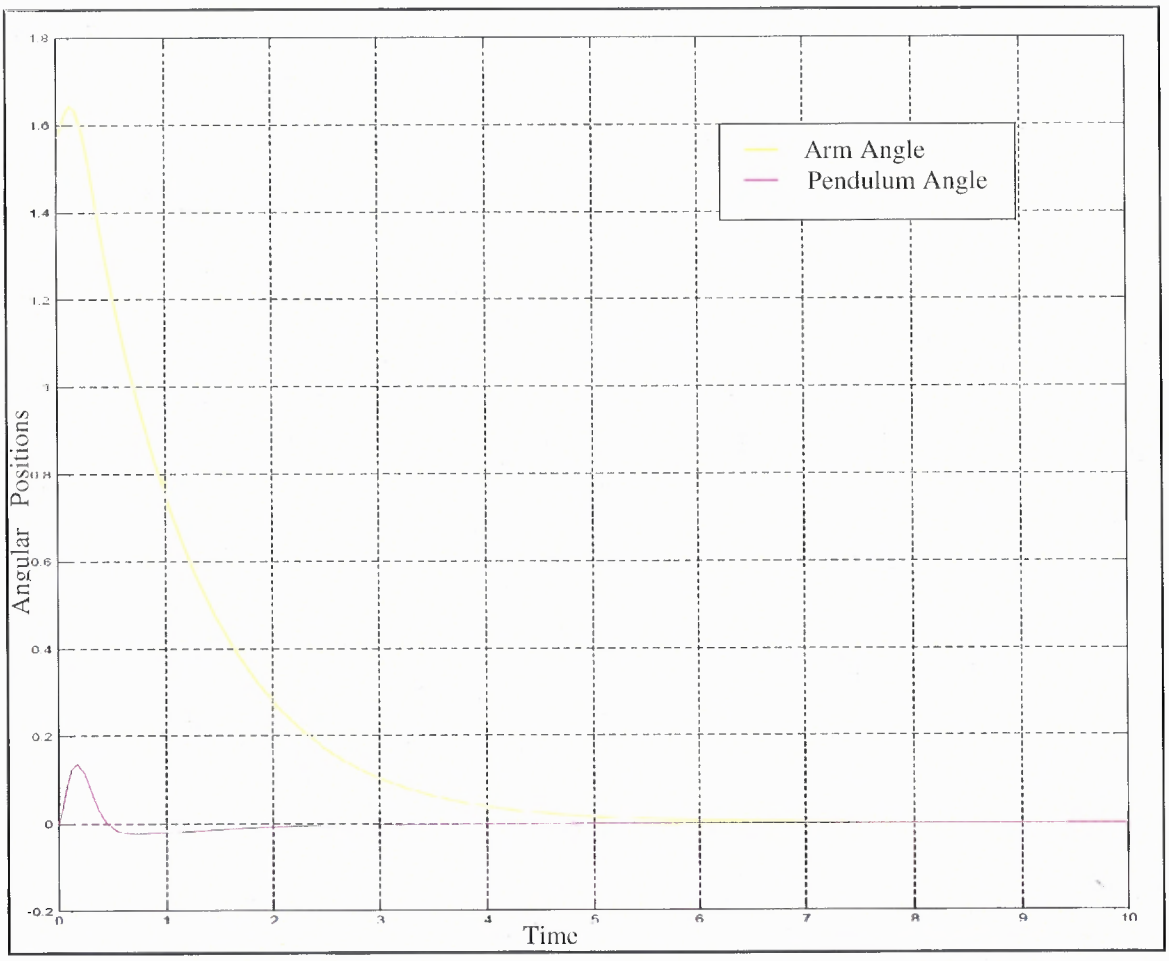


Figure 3.6 Simulink Pole Placement output for the arm angle and the pendulum angle with initial conditions [1.57, 0] on angles.

Above Figure 3.6 shows transient responses for arm and pendulum positions with Pole Placement. When the arm is made to return from 90^{o} ($\pi/2$ radians) offset, it first moves in the opposite direction by a small angle and finally comes to steady state in about 6 seconds. The pendulum first falls down through a small angle and finally settles down in about 2 seconds. Settling time of arm is longer than pendulum because arm has to compensate for change in positions of pendulum.

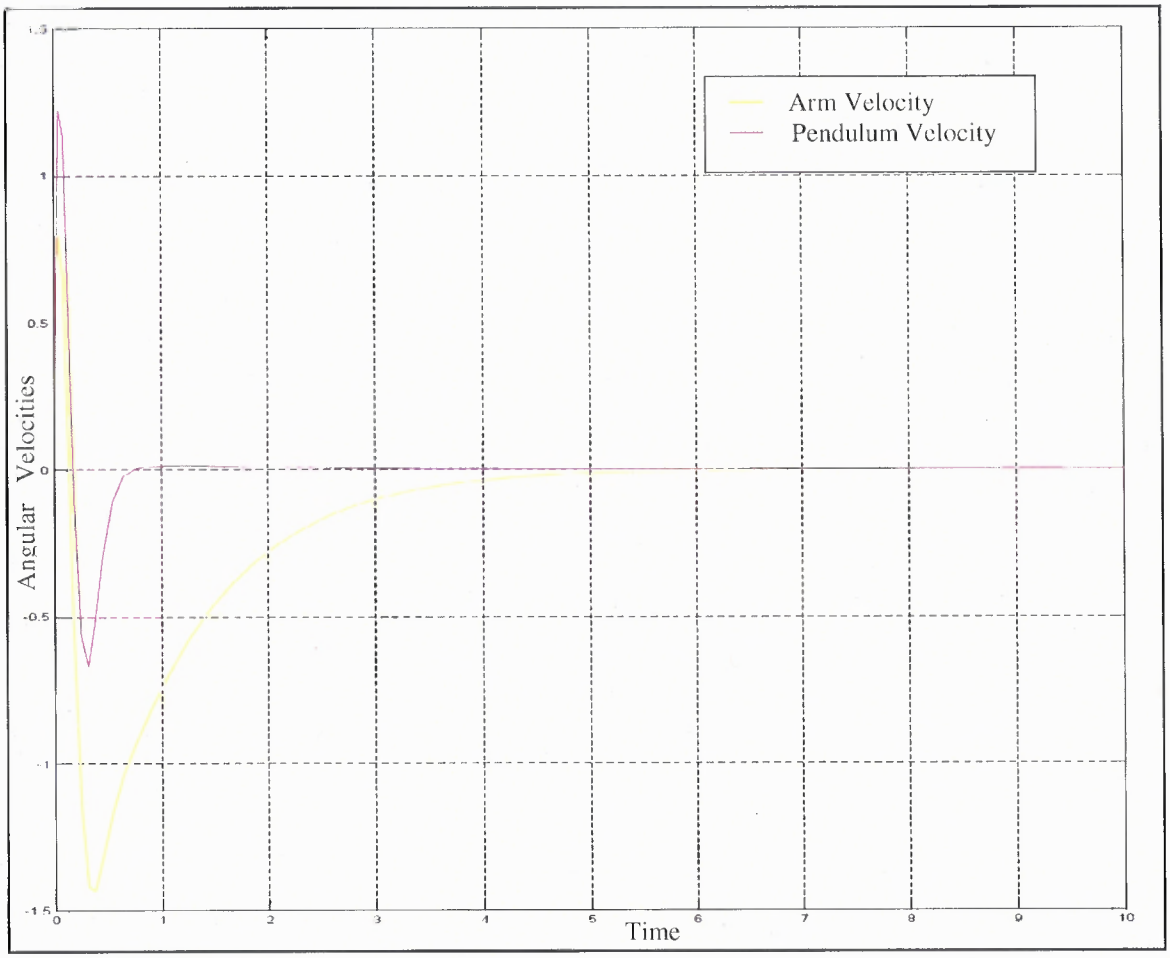


Figure 3.7 Simulink Pole Placement output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0] on angles.

Transient responses for arm velocity and pendulum velocity have been observed for Pole Placement as shown in above Figure 3.7. Velocity components for both of them are initially high but it settles down in accordance with position components. Clearly, arm velocity goes to zero in about 6 seconds and pendulum velocity goes to zero in about 2 seconds. Both position and velocity profiles validate fact that Pole Placement control design worked successfully to balance inverted pendulum on rotational arm.

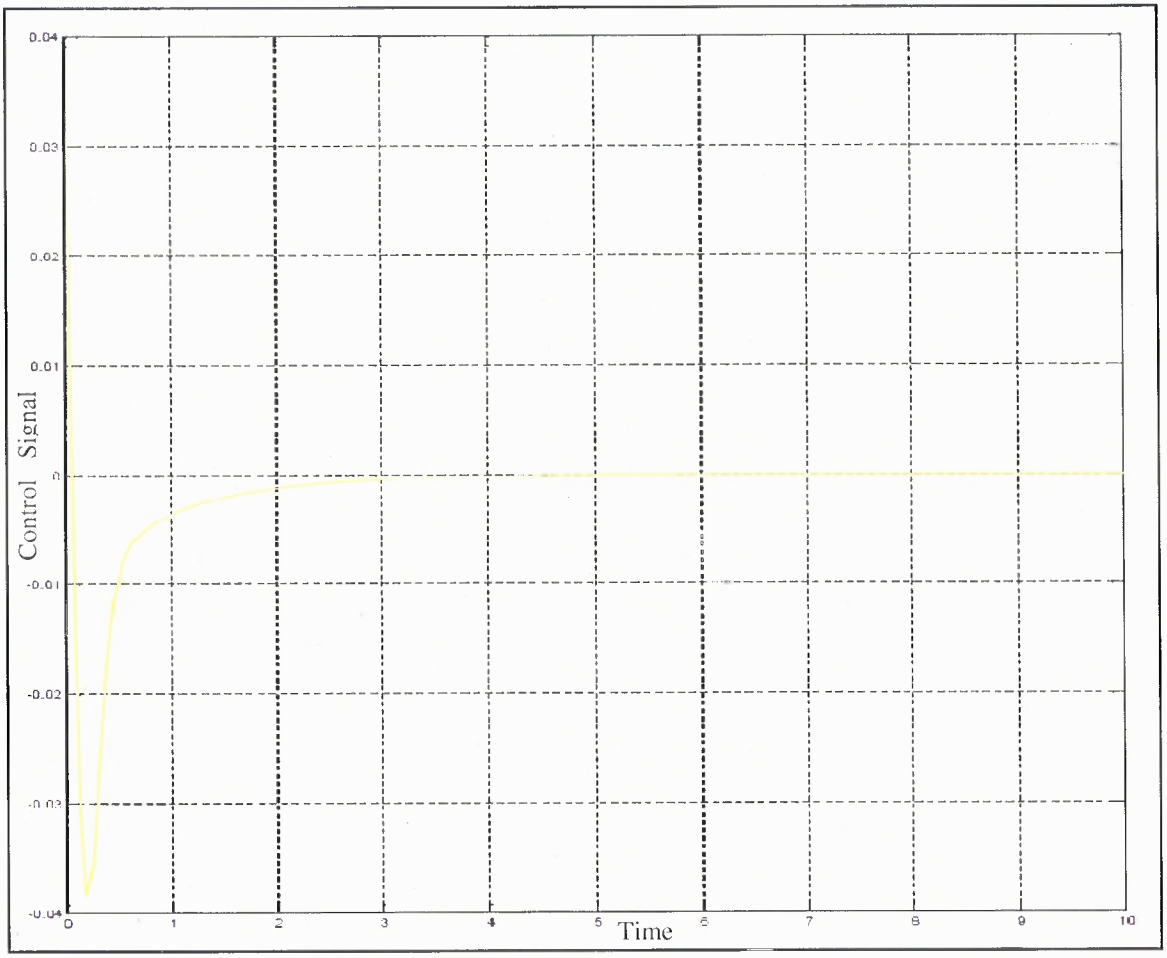


Figure 3.8 Simulink output for control signal generated with Pole Placement.

Above Figure 3.8 presents control signal profile obtained with Pole Placement design. The control signal is initially a little high, but it soon becomes zero. The arm position comes to steady state in about the same time as the control signal exists i.e., 6 seconds. This shows the perfect coordination of the states with the control signal.

3.1.3 Full State Feedback design by LQR

Section 3.1.3 describes Full State Feedback control design using Linear-quadratic (LQ) state-feedback regulator for state-space system. It also presents simulation results for LQR control design. M-file for LQR design is as shown below.

M-file:

```
%%%%%%   Linearized Model Using linmod  %%%%
[A,B,C,D]=linmod('FurutaDyn')
%%%%%%%%%%%% Full-State Feedback Using LQR
q1=1000
q2=100000
Q=diag([q1,q2,0,0])
R=1
[G,M,Elq]=lqr(A,B,Q,R)
Output:
Q =

        1000       0         0       0
           0    100000       0       0
           0       0         0       0
           0       0         0       0


R =

     1


G =

  -31.6228 339.9081 -25.6565   17.7497
```

MATLAB command `[G,M,Elq]=lqr(A,B,Q,R)` calculates the optimal gain matrix G. In addition to the state-feedback gain G, `lqr` returns the solution S of the associated Riccati equation and the closed-loop eigen values e = eig(A-B*G). Here 'G' is derived from 'S' using M. State weighing matrix Q has been chosen such that it weighs

pendulum angle greater than that of an arm angle since pendulum angle has greater impact on overall system performance. [4] [5] [6]

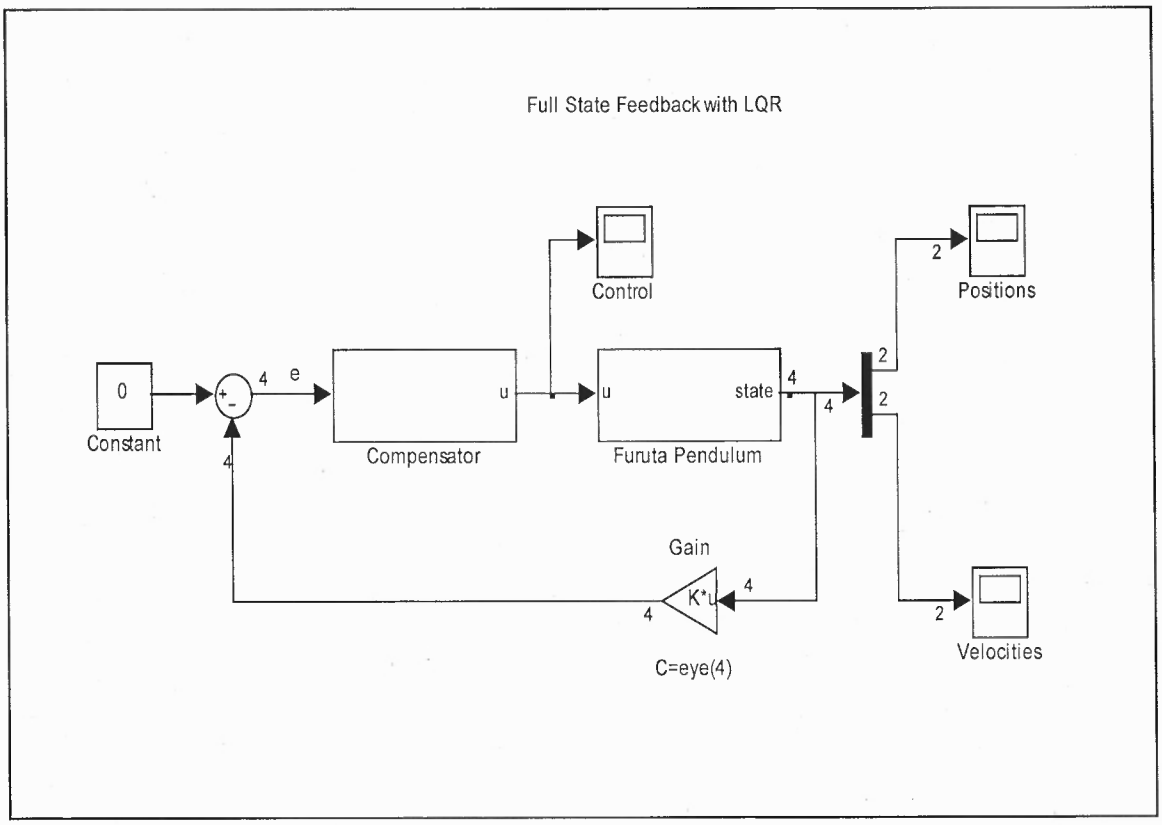


Figure 3.9 Simulink block diagram for Full State Feedback design by LQR.

Simulink block diagram for LQR control is as shown in Figure 3.9. Gain block contains 'C' matrix which feeds back all output states with negative feedback. Compensator subsystem contains another Gain block which contains optimal gain G calculated using above M-file. Constant block provides zero as reference input.

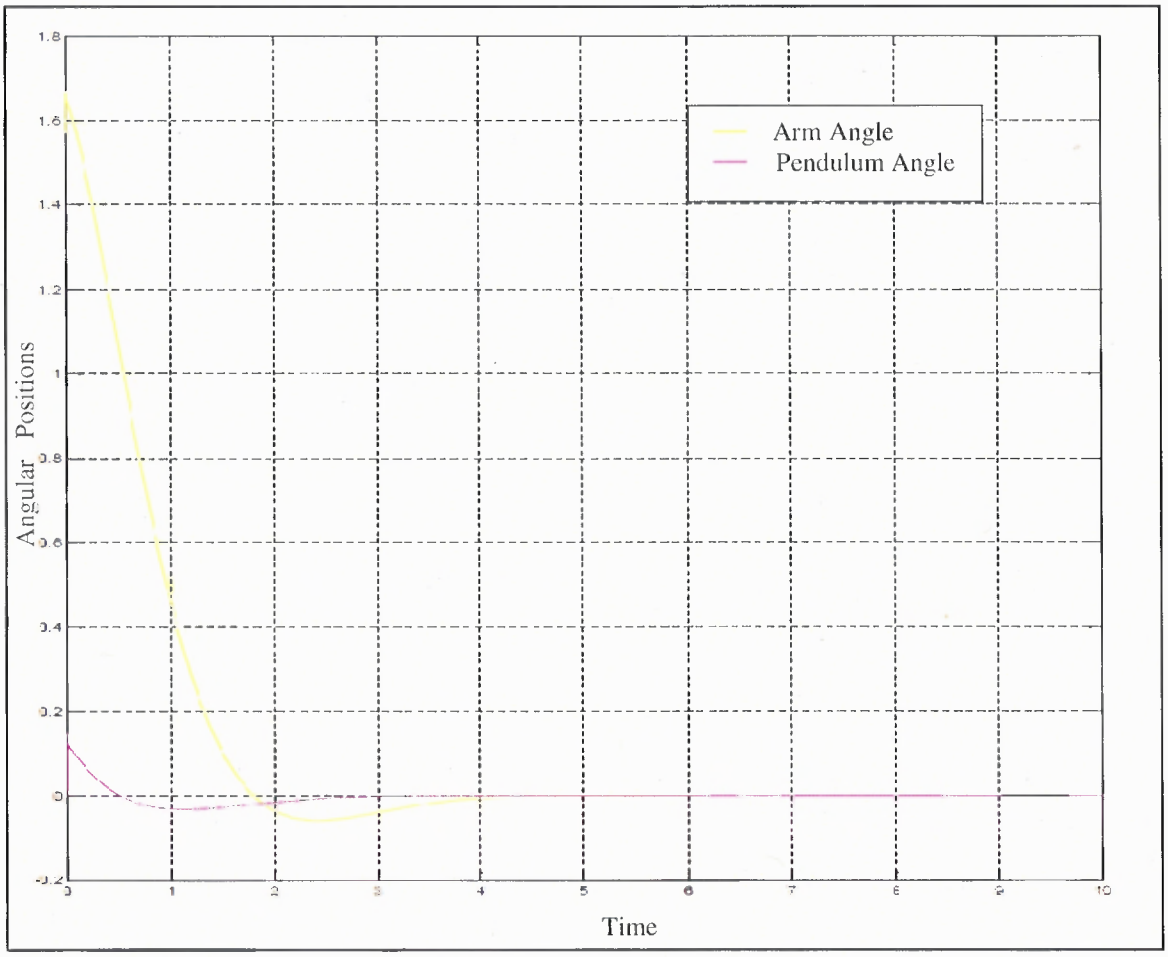


Figure 3.10 Simulink LQR output for the arm angle and the pendulum angle with initial conditions [1.57, 0] on angles.

Above Figure 3.10 shows transient responses for arm and pendulum positions with LQR control. When the arm is made to return from 90^{0} ($\pi/2$ radians) offset, it first moves in the opposite direction by a small angle and finally comes to steady state in about 4 seconds. The pendulum first falls down through a small angle and finally settles down in about 2 seconds. It has been observed for LQR design an arm position comes to zero quicker than Pole Placement control.

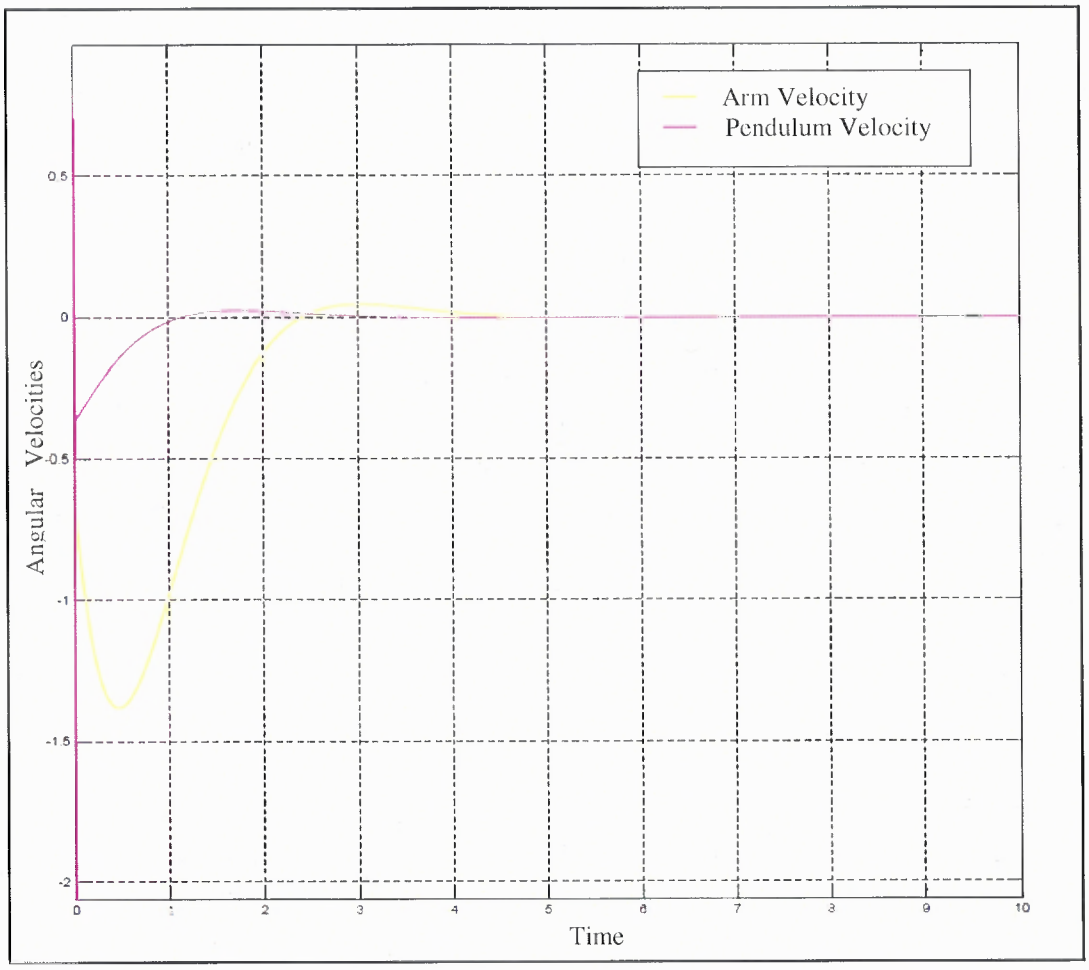


Figure 3.11 Simulink LQR output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0] on angles.

Transient responses for arm velocity and pendulum velocity have been observed for LQR as shown in above Figure 3.11. Velocity components for both of them are initially high in opposite direction but it settles down quickly in accordance with position components. Clearly, arm velocity goes to zero in about 2.2 seconds and pendulum velocity goes to zero in about 1.1 seconds. It shows that LQR control design is faster than that of Pole Placement control. Both position and velocity profiles validate fact that LQR control design worked successfully to balance inverted pendulum on rotational arm.

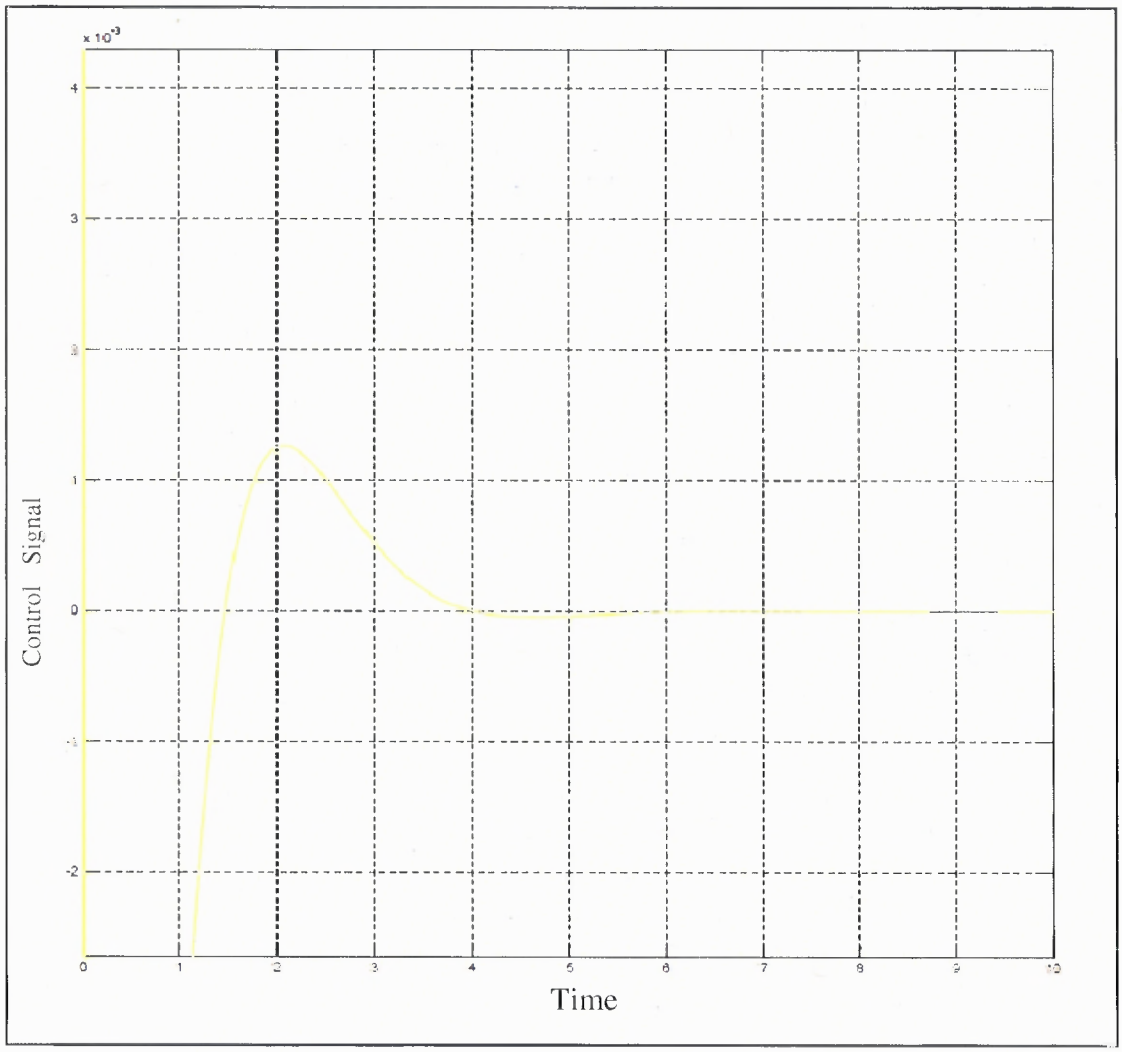


Figure 3.12 Simulink output for control signal generated with LQR.

Figure 3.12 shows profile of control signal generated with LQR design. Controls signal looks pretty higher initially and distorted around zero level. Pole Placement control signal was much smoother than that of LQR control.

3.1.4 Full Order Observer Design using Kalman Filter

Section 3.1.4 presents Full Order Observer design with Kalman filter. It also describes simulation results for Kalman Observer. M-file to design Full State Feedback gain and Observer State space dynamics is as shown below.

M-file:

```
% %%%%%% Full State Feedback Using LQR
q1=1000
q2=100000
Q=diag([q1,q2,0,0])
R=1
[G,M,Elq]=lqr(A,B,Q,R)
% % % %%%  Full Order Kalman Filter
V=0.02
W=eye(2)
C=[1 0 0 0;0 1 0 0]
[K, P, EKF]=lqe(A,B,C,V,W)
Cc=G
Ac=A-B*G-K*C
Bc=K
Dc=[0,0]
Ecom=eig(Ac)
comp=ss(Ac,Bc,Cc,Dc)
tf(comp)
```

MATLAB command `'lqe'` designs Kalman estimator for continuous-time systems with unbiased process noise w and measurement noise v with co-variances. In above M-file `[K, P, EKF]= lqe(A,B,C,V,W)` returns the observer gain matrix K such that the stationary Kalman filter produces an optimal state estimate x_e of x using the sensor measurements y. [4] [5]

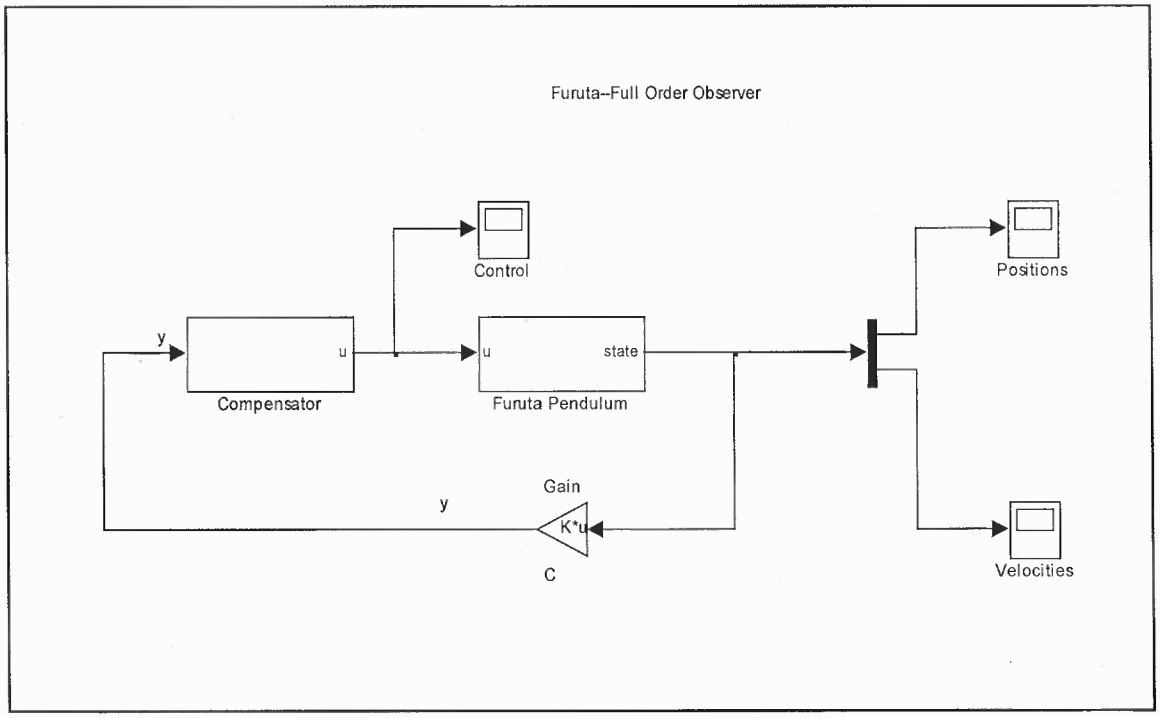


Figure 3.13 Simulink block diagram for full order Observer design with Kalman Filter.

The gain block contains matrix C= [1 0 0 0; 0 1 0 0] which feedbacks an arm and pendulum position to Kalman Observer. Compensator sub-system contains state space model block as shown in Figure 3.13 for an Observer dynamics. Kalman Observer dynamics Ac, Bc, Cc, Dc have been specified inside parameter spaces of state space model. Observer system matrix contains state feedback gain matrix 'G' and Observer gain matrix 'K' i.e. Ac=A-B*G-K*C. [6]

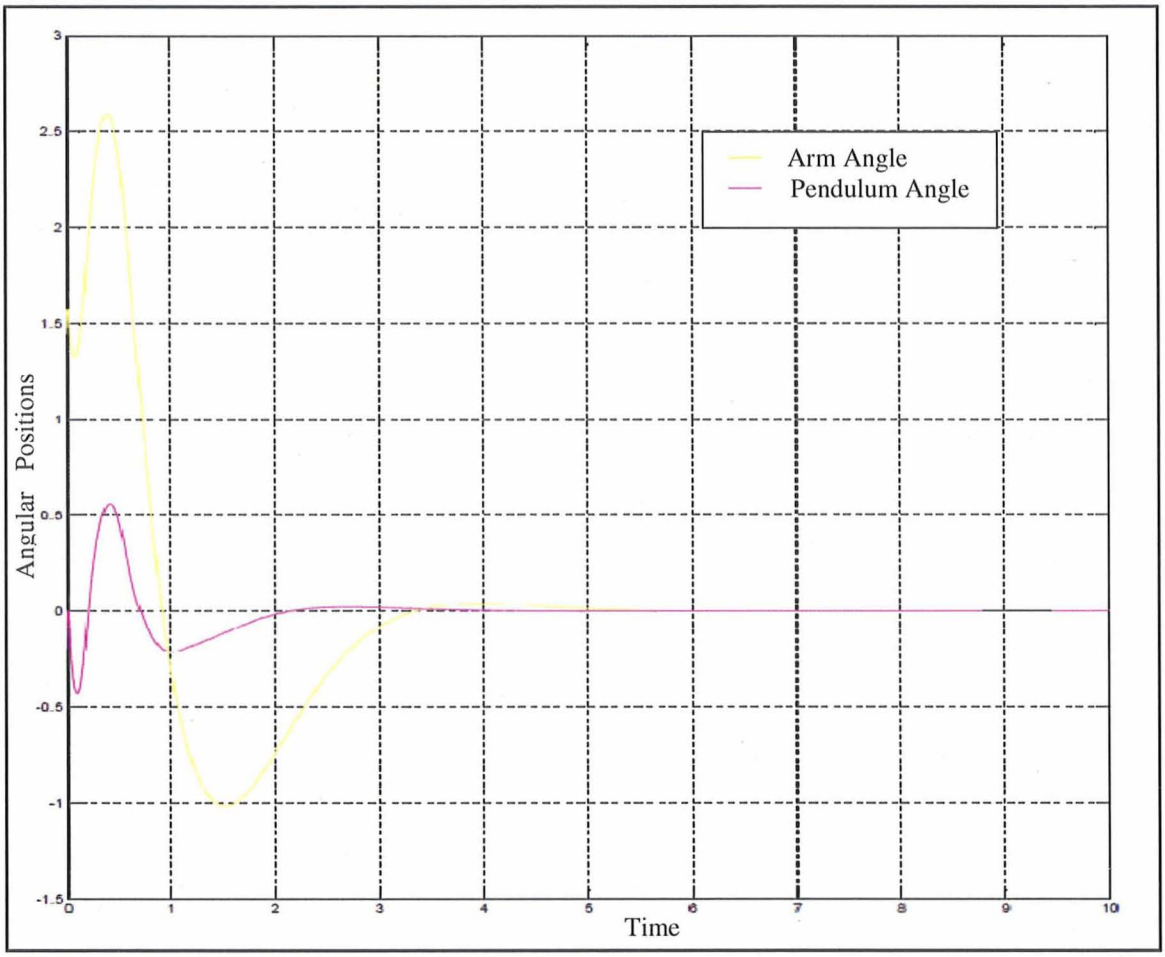


Figure 3.14 Simulink Kalman Observer output for the arm angle and the pendulum angle with initial conditions [1.57, 0] on angles.

Transient responses for an arm position and pendulum position with Kalman Full Order Observer in feedback are as shown in Figure 3.14. Assuming both sensors of equal accuracy, spectral density v is taken to be of a small value. With an initial condition of 90^{o} ($\pi/2$ radian), the arm moves to a small angle in the opposite direction and then comes to steady state in about 3.2 seconds. The pendulum also deflects by some angle before coming to zero in about of 2 seconds. Since both positions are settled at zero it verifies fact that Kalman Observer design successfully balanced inverted pendulum on an arm.

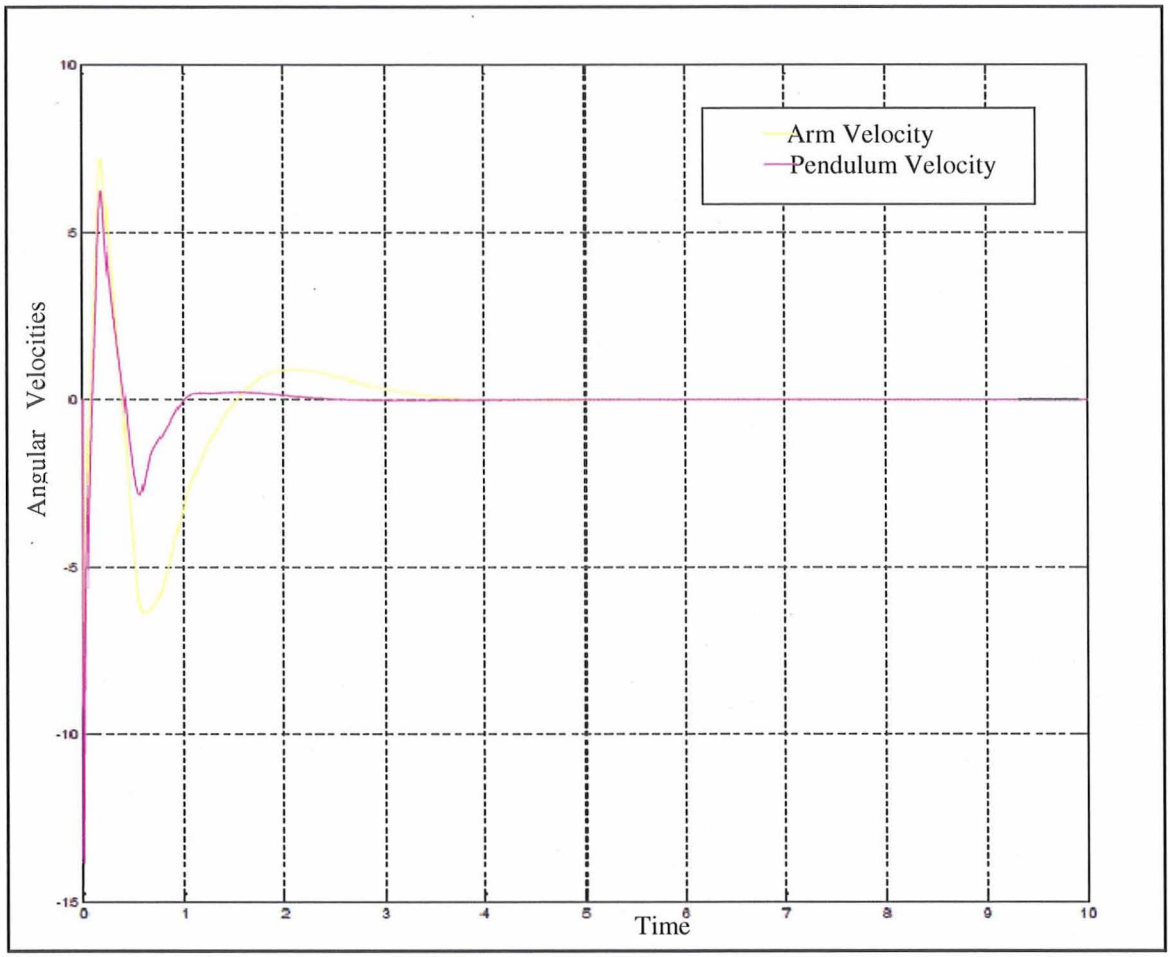


Figure 3.15 Simulink Kalman Observer output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0] on angles.

Figure 3.15 shows transient responses for an arm velocity and pendulum velocity with Kalman Observer in feedback. These are estimated arm and pendulum velocity states from an arm and pendulum positions feedbacks. Position and velocity profiles validate fact that Kalman Observer can estimate velocity states based on position feedbacks and in turn it could successfully balance inverted pendulum on an arm.

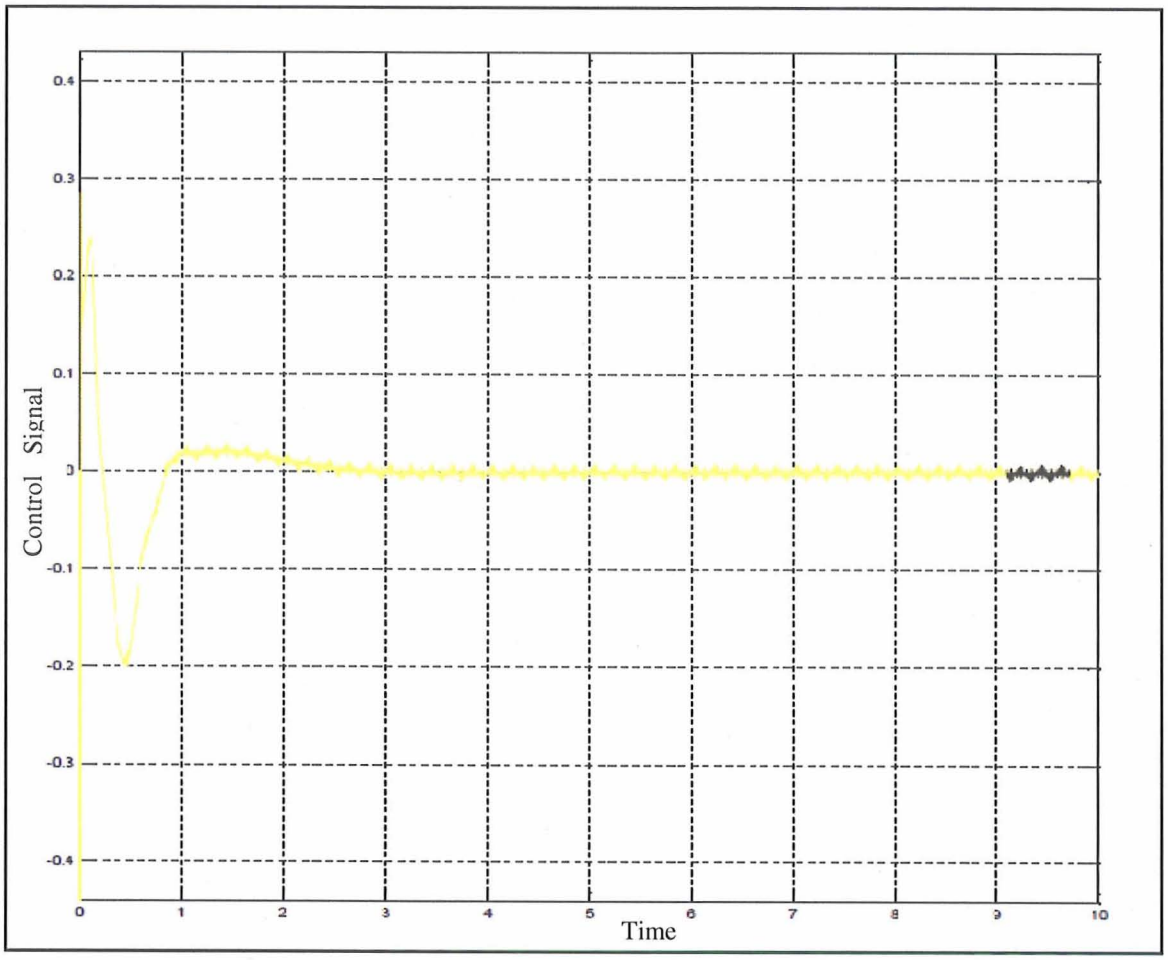


Figure 3.16 Simulink output for control signal generated with Kalman Observer.

Above Figure 3.16 shows control signal generated by Kalman Full Order Observer. The control initially follows a transient with oscillations about zero. It finally reaches zero at 2.2 seconds. At this moment a steady state is attained by the system. Control signal looks pretty high initially. It has been observed that states settle to steady state in accordance with Observer control signal.

3.2 LabVIEW

The Laboratory Virtual Instrument Engineering Workbench (LabVIEW) is a software package from National Instruments. The LabVIEW platform provides specific tools and models to solve specific applications ranging from designing control algorithms to deploying to hardware and can target any number of platforms from the desktop to embedded devices. LabVIEW has dedicated control design and simulation tool boxes for the frequency domain analysis, State space analysis and stochastic analysis. [7]

3.2.1 Open Loop Analysis

Control design and simulation are based on Virtual Instrumentation (VI) approach in LabVIEW. Virtual instrumentation is applicable in different types of applications, starting from the design to prototyping and deployment. LabVIEW streamlines the system design with a single graphical development platform. VI approach is based on the Model-based control design. It involves the four phases as mentioned below:

1. developing and analyzing a model to describe a plant
2. designing and analyzing a controller for the dynamic system
3. simulating the dynamic system
4. deploying the controller [8] [9]

Figure 3.17 shows different ways in which LabVIEW toolkits offer solutions to each step in the Model-based control design process. In the case of Furuta pendulum, Control Design and Simulation toolkits have been used to demonstrate different State space control techniques.

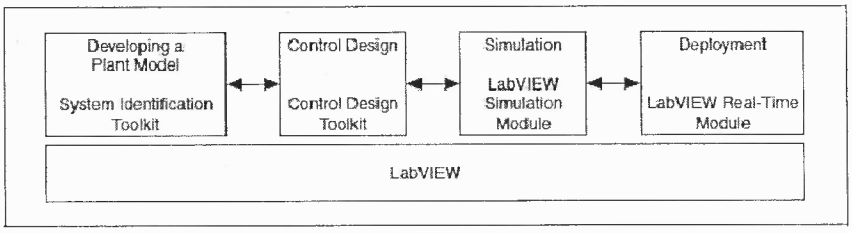


Figure 3.17 Using LabVIEW in Model-Based control design. [8]

The Control Design toolkit contains following palettes which provide full range of functions useful for the state space control design. LabVIEW palettes are similar to Simulink block library containing multi domain tools.

- The State space Model Analysis palette, with the following functions [8]:
  - Controllability Matrix
  - Observability Matrix
  - Grammians
  - Canonical State-Space Realization
  - Balance State-Space Model (Diagonal)
  - Balance State-Space Model (Grammians)
  - Controllability Staircase
  - Observability Staircase
  - State Similarity Transform
- The state feedback design palette, with the following functions:
  - Ackermann
  - Pole Placement
  - Linear Quadratic Regulator
  - Kalman Gain

- State Estimator
- State-Space Controller
- Augment Output with States

Developing the Furuta Pendulum Model using LabVIEW:

LabVIEW programs are called *virtual instruments* (VIs). Controls are inputs and the indicators are outputs. Each VI contains three main parts:

Front Panel – How the user interacts with the VI.

Block Diagram – The code that controls the program.

Icon/Connector – Means of connecting a VI to other VIs.

LabVIEW allows the user to build an interface by means of a set of tools and objects. The user interface is known as the front panel. The front panel allows adding code using graphical representations of the functions to control the front panel objects. The block diagram contains this code. In some ways, the block diagram resembles a flowchart. Users interact with the front panel when the program is running. Users can control the program, change inputs, and see the data updated in real time. Controls are used for the inputs such as, adjusting a slide control to set an alarm value, turning a switch on or off, or to stop a program. Indicators are used as outputs. These may include data, program states, and other information. Every front panel control or indicator has a corresponding terminal on the block diagram. When a VI is run, values from the controls flow through the block diagram, where they are used in the functions on the diagram, and the results are passed into other functions or indicators through 'wires'. The Controls palette has been used to place controls and indicators on the front panel. The Controls

palette is available only on the front panel. The Functions palette has been used to build the block diagram. The Functions palette is available only on the block diagram. [9] [7]

Figure 3.18 shows block diagram of VI for the Furuta pendulum. It consists of different blocks like non-linear Furuta model subsystem, Index Array Function, Build array Function and Sim Time Waveform Function. Further non-linear Furuta model has been expanded in Figure 3.20. It consists of Integrators, MathScript node, Index Arrays and Build Array functions and controls for initial conditions on the arm angle and the pendulum angle.

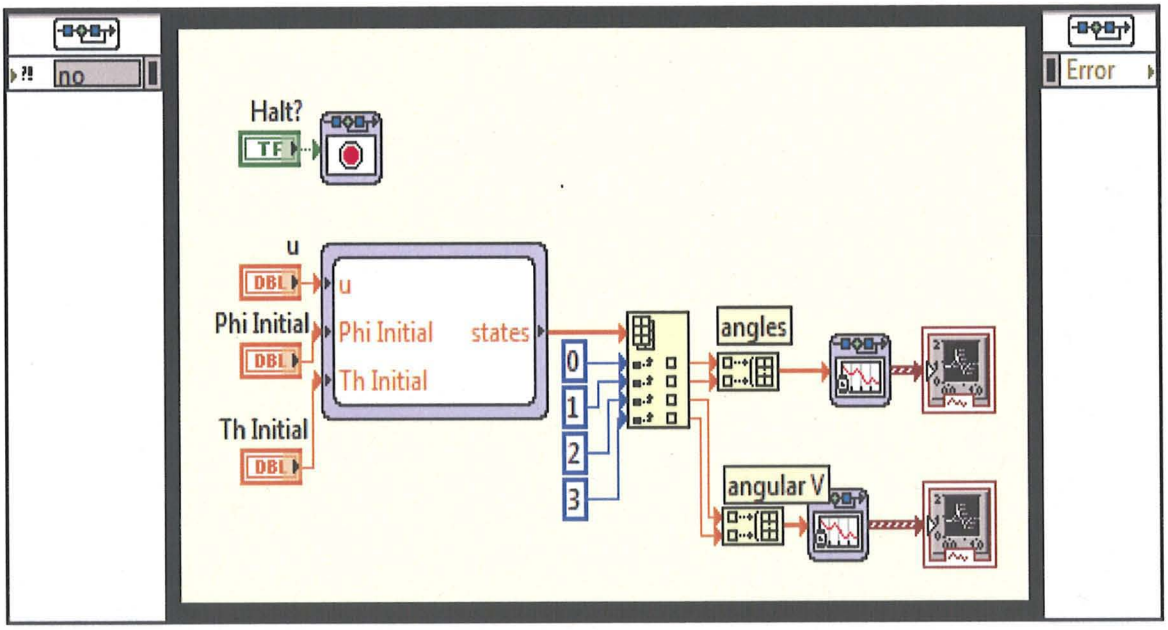


Figure 3.18 Open Loop block diagram of Furuta Pendulum VI.

The MathScript window provides an interactive environment where the equations can be prototyped and the calculations can be made. The MathScript node enhances LabVIEW by adding a native text-based language for the mathematical algorithm implementation in the graphical programming environment. The M-file scripts created in other math software such as MATLAB can be run with MathScript. The MathScript

allows you to pick the syntax you are most comfortable with to solve the problem. Equations can be instrumented with the MathScript node for parameter exploration, simulation, or deployment in a final application. MathScript has ability to import and export m-files by right-clicking on the node. [7]

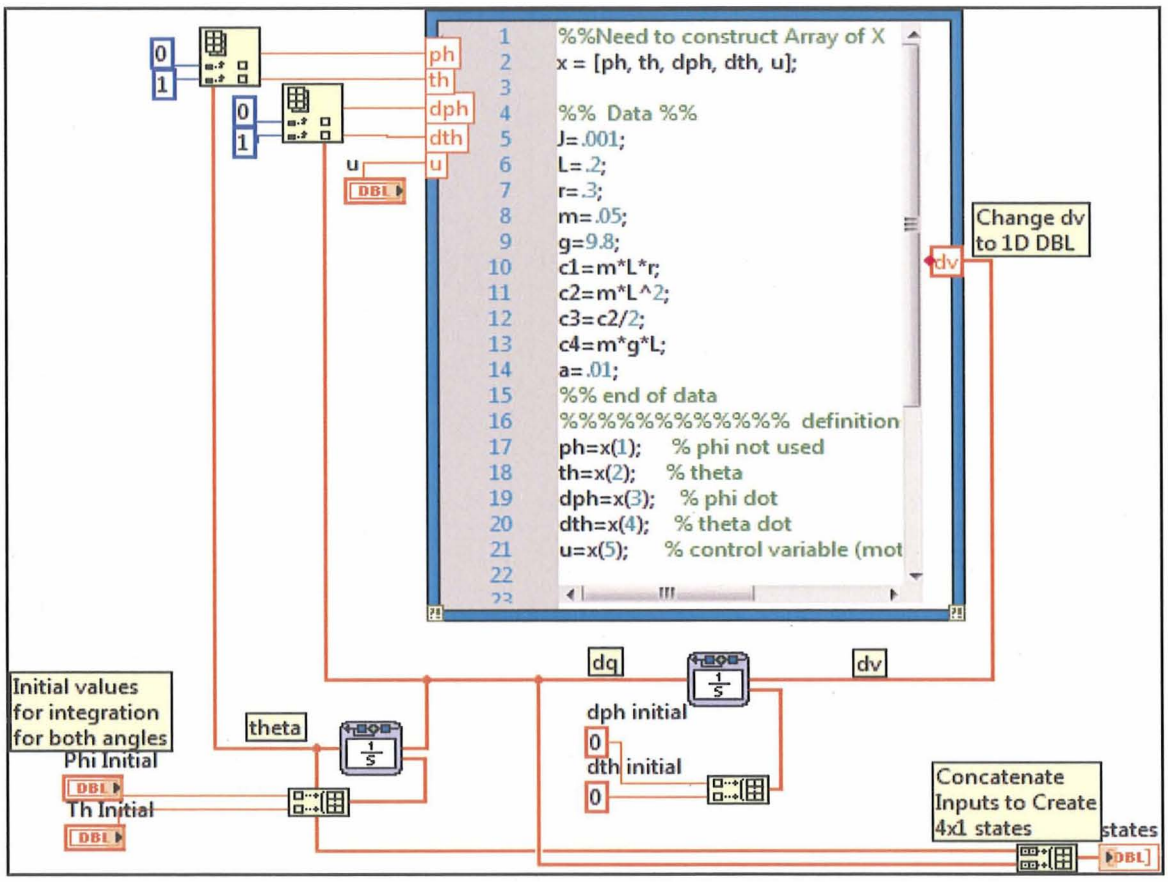


Figure 3.19 Block diagram of Furuta Pendulum subsystem.

Figure 3.19 shows MathScript node which contains similar M-file script that has been used inside MATLAB Fcn block of Simulink. The MathScript node defines the dynamics and equations of motions of the non-linear Furuta Pendulum. The MathScript node doesn't require to form function 'Furuta' like the MATLAB Fcn block. The MathScript node requires specifying inputs and outputs for equations of motions of the

Furuta pendulum system. The inputs and output for the Furuta pendulum MathScript node are defined in the scalar form. It is necessary to define proper data type for the output 'dv' of the MathScript node. As shown in Figure 3.20 ph, th, dph, dth and u are the inputs and dv is the output as defined in Section 2.2. Data type of 'dv' has been set to 1D array. Acceleration component 'dv' is a 2*1 matrix which goes through series of cascaded Integrators.

The first integrator generates the angular velocities and the second integrator generates the angular positions. The integrator integrates a continuous input signal using the ordinary differential equation (ODE) solver specified in simulation. Integrator accepts vector as well as scalar input similar to Simulink integrator. The initial conditions have been defined as user defined control on front panel for the arm angle and the pendulum angle. Outputs of both the Integrators have been given to the Build Array Function block. It concatenates multiple arrays or appends elements to an n-dimensional array. Build Array Function allows adding inputs by right clicking on it. Build Array Function operates in one of the two modes depending on selection of 'Concatenate Inputs' from the shortcut menu. After selection of 'Concatenate Inputs', the function appends all inputs in order, forming an output array of the same dimensionality as the highest dimension array input wired. It is similar to Mux block of Simulink. Output of Build Array Function gives an array of 4*1 dimension containing four states of the Furuta pendulum system. Further, outputs of these Integrators have been feedback to MathScript node via Index Array Functions.

When the state array which is an output of the Build Array Function has been wired to this Index Array function, the function resizes automatically to display index

inputs for each dimension in the array. One Index Array Function has been used for each of the Integrator function. 0 and 1 have been specified as indexes for both Index Array Functions. These indexes extract individual position and velocity components to be given as inputs to MathScript node. Outside of the Furuta Pendulum subsystem another Index Array Function has been used which generates the position and the velocity states of the Furuta pendulum. These states have been displayed using the Sim Time Waveform Function and waveform chart. The Sim Time Waveform Function plots a value versus the simulation time on a waveform chart. Sim Time Waveform function is similar to scope block in Simulink. [7]

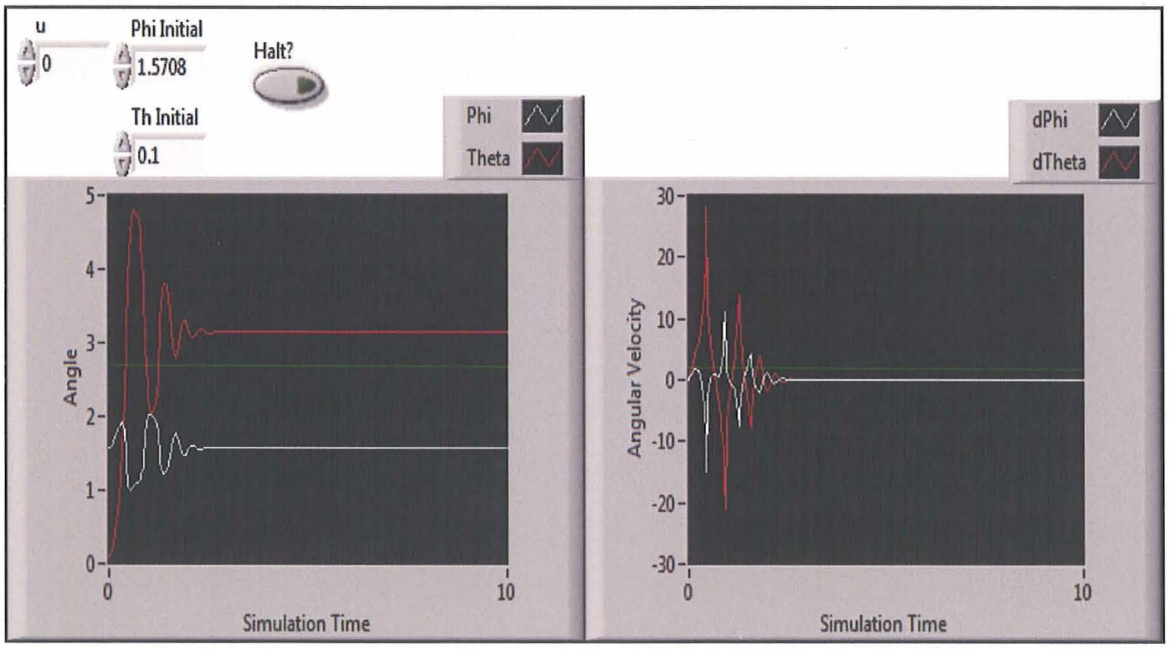


Figure 3.20 Front panel of the open loop Furuta Pendulum VI.

Figure 3.20 shows front panel of LabVIEW for the open loop Furuta Pendulum VI. It consists of waveform charts, user control for control signal 'u' and initial condition controls for the arm angle and the pendulum angle. For open loop response simulation of the Furuta Pendulum control signal 'u' has been set to 0. With initial conditions 1.5708

radian on the arm angle and 0.1 radian on the pendulum angle, wave charts on front panel of LabVIEW showed same transient responses as Simulink scope. The pendulum fall down by moving through an angle of 3.14 radian. After few oscillations arm settled down to its initial position. Also, both the arm velocity and the pendulum velocity settled down to zero which corresponds to the angular positions.

Linearization:

The Control Design and Simulation tool box of LabVIEW contains the tool 'Linearize Subsystem' for linearization of subsystem. As shown in Figure 3.20 the non-linear model of Furuta Pendulum has been saved as another VI known as furuta non-linear model. The Linearize Subsystem tool dialogue box allows selecting this new VI for linearization process. This tool also provides various options to provide trimmed operating points, initial inputs, initial outputs and initial states. After clicking on 'Linearization' tab it gives state space model of the Furuta Pendulum system as shown in Figure 3.21. It doesn't match with the state space model obtained in MATLAB with 'linmod' command. [7]

$$\begin{bmatrix} dx/dt \\ y(t) \end{bmatrix} = \left[\begin{array}{cccc:ccc} -10 & 0 & 0 & 147 & 1000 & 0 & 0 \\ -15 & 0 & 0 & 269.5 & 1500 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \\ \hdashline 0 & 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 1 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 & 0 \end{array}\right] \begin{bmatrix} x(t) \\ u(t) \end{bmatrix}$$

$$u0 = \begin{bmatrix} 0 & 0 & 0 \end{bmatrix}$$

$$y0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

$$x0 = \begin{bmatrix} 0 & 0 & 0 & 0 \end{bmatrix}$$

Figure 3.21 State-space model of the Furuta Pendulum by the 'Linearize Subsystem' tool of LabVIEW.

3.2.2 Full State Feedback design by Pole Placement:

The Section 3.2.2 presents full state feedback control design by pole placement for the Furuta Pendulum system. The Control Design and Simulation tool box of LabVIEW has been used for Pole Placement design. This section also shows simulation results obtained with LabVIEW for Pole Placement control design.

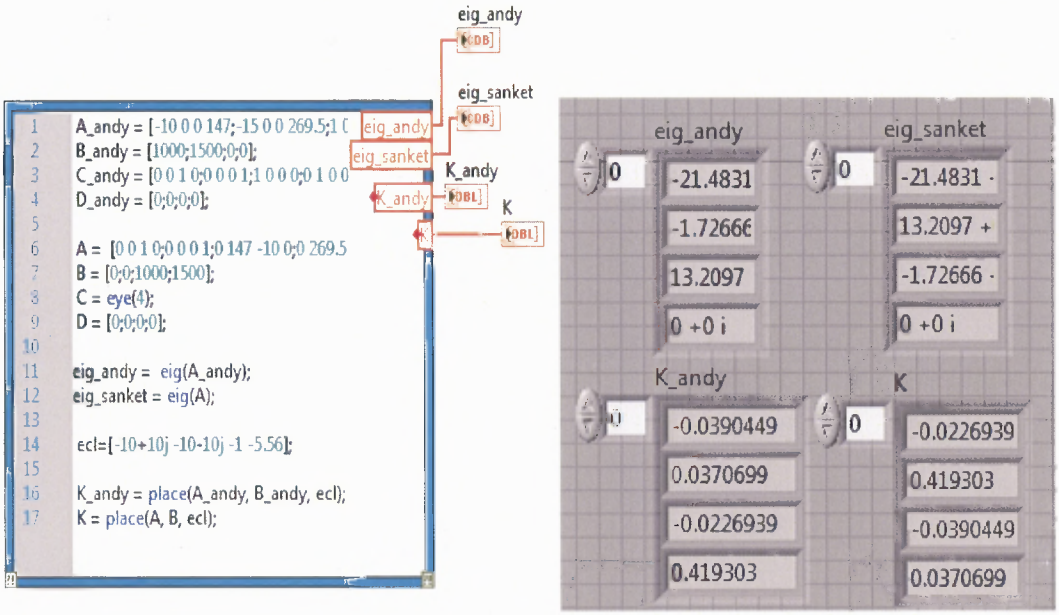


Figure 3.22 MathScript design steps for the Pole Placement control.

As shown in Figure 3.22 MathScript node has been used to calculate the full state feedback gain 'k'. MathScript node accepts MATLAB functions similar to the M-file that has been used to calculate full state feedback gain 'G'. Front panel shown in Figure 3.23 provides comparison between caculation of full state feedback gain using state space model obtained with MATLAB as well as LabVIEW. It has been verified with results that controller gain values are same with reshuffled order. With LabVIEW user can not control state variable numbers that is output matrix 'C'. After remapping of gain vector, controller gain k= [-0.0227 0.4193 -0.0390 0.0371] has been used with the CD State Feedback Controller VI to implement state feedback control law $u = -Kx$. The CD State Feedback Controller VI implements a state-space controller where the controller action equals –Controller Gain * States.

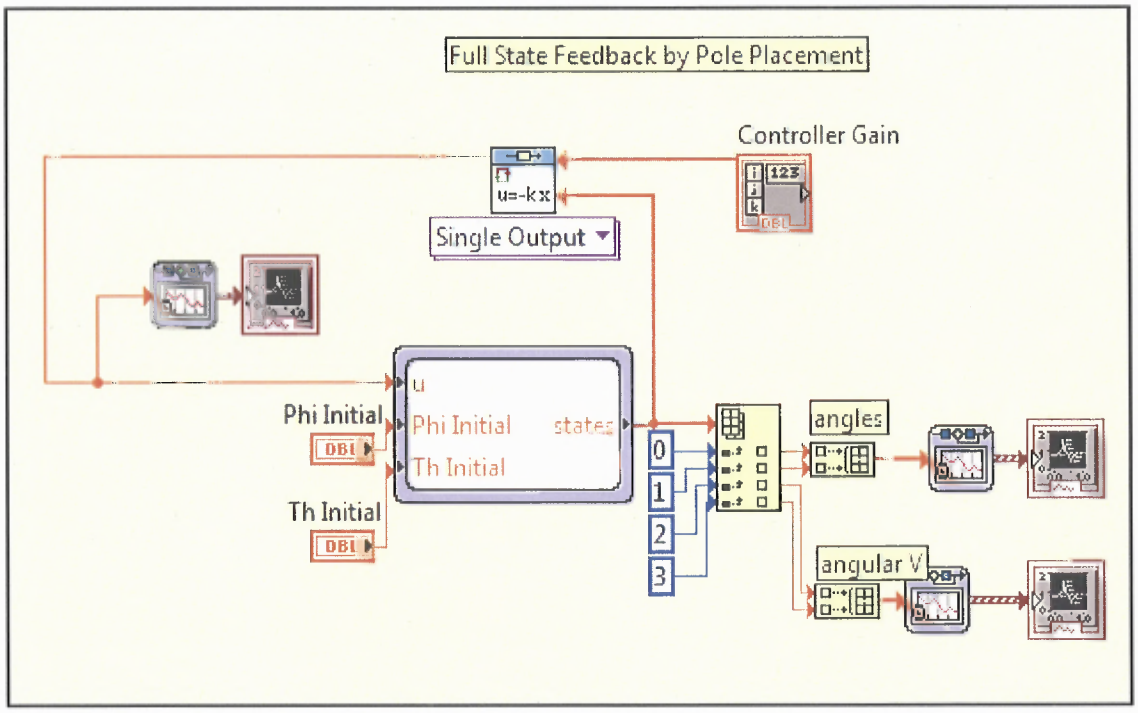


Figure 3.23 Block diagram of the Pole Placement VI for the Furuta Pendulum.

Figure 3.23 shows the block diagram of the pole placement VI for the Furuta Pendulum system. It is similar to open loop VI of the Furuta Pendulum except CD State Feedback Controller VI in feedback loop. Figure 3.25 shows front panel of the Pole Placement VI for the Furuta Pendulum system. It shows transient responses of the arm position (Phi), pendulum position (Theta), arm velocity (dPhi), pendulum velocity (dTheta) and control signal (u). All these transient responses are same as compared to Simulink.

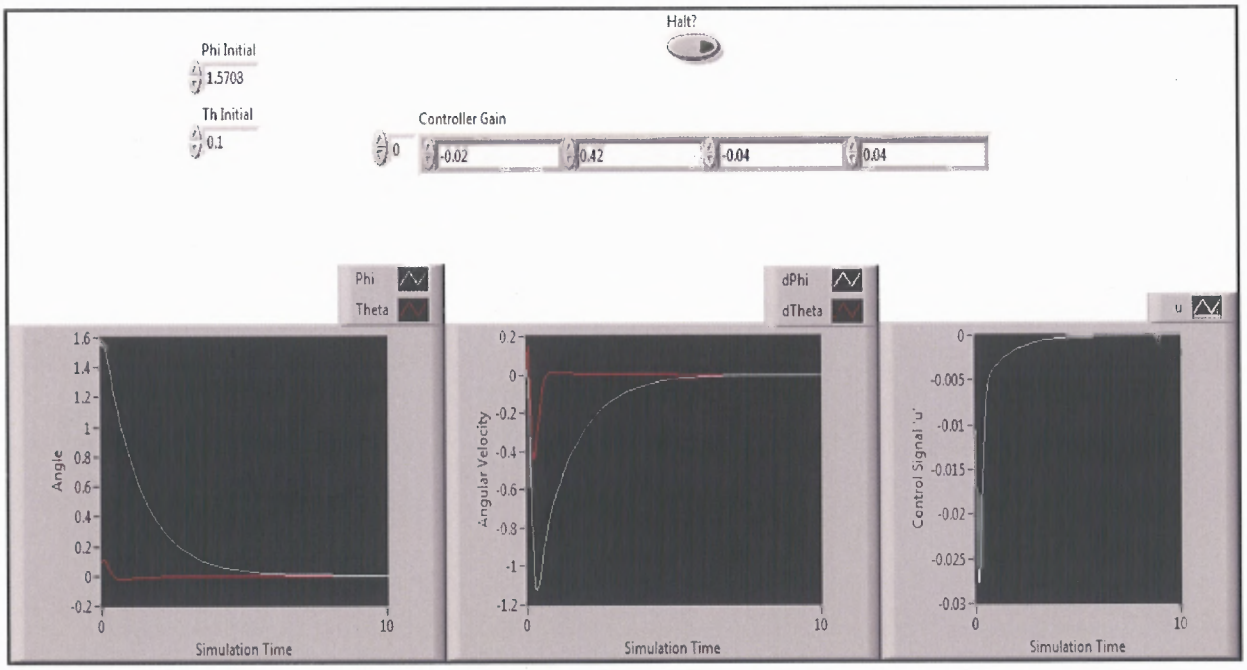


Figure 3.24 Front panel of the Pole Placement VI for the Furuta Pendulum.

3.2.3 Full State Feedback design by LQR:

The Section 3.2.3 describes Full State Feedback control design using Linear-quadratic (LQ) state-feedback regulator for state-space system. It also presents simulation results for LQR control design. Figure 3.25 shows MathScript node block which has been used to calculate full state feedback gain 'G' using linear quadratic equations. The 'lqr' command from MATLAB is compatible with MathScript node. As a result, whole M-file code which was used in MATLAB has been imported in MathScript node. The front panel shown in Figure 3.25 gives LQR controller gain vector that is G= [-25.6565 17.7497 -31.6228 339.9081]. It is similar as compared to MATLAB LQR full state feedback gain vector except reshuffled order. This change in order is due to difference in state space model. After remapping of gains, G = [-31.6228 339.9081 -25.6565 17.7497] has been used as LQR controller gain.

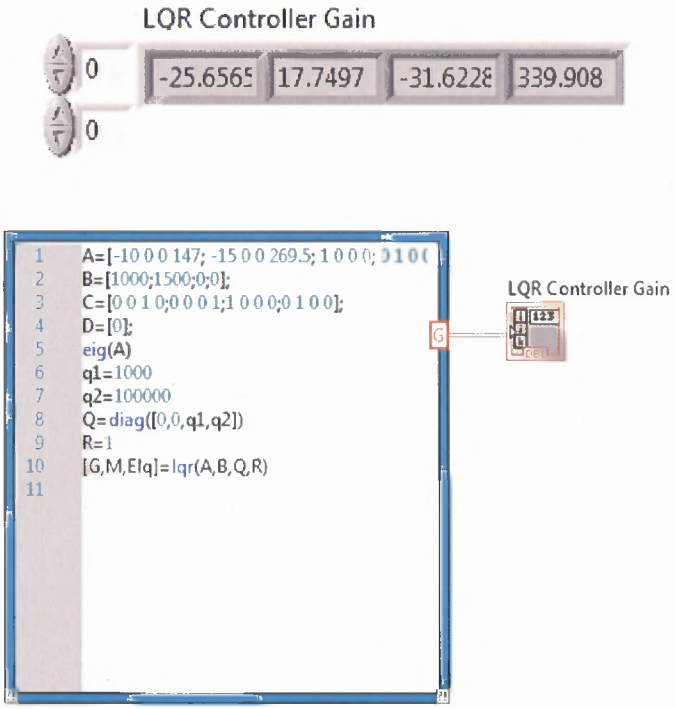


Figure 3.25 MathScript design steps for LQR control.

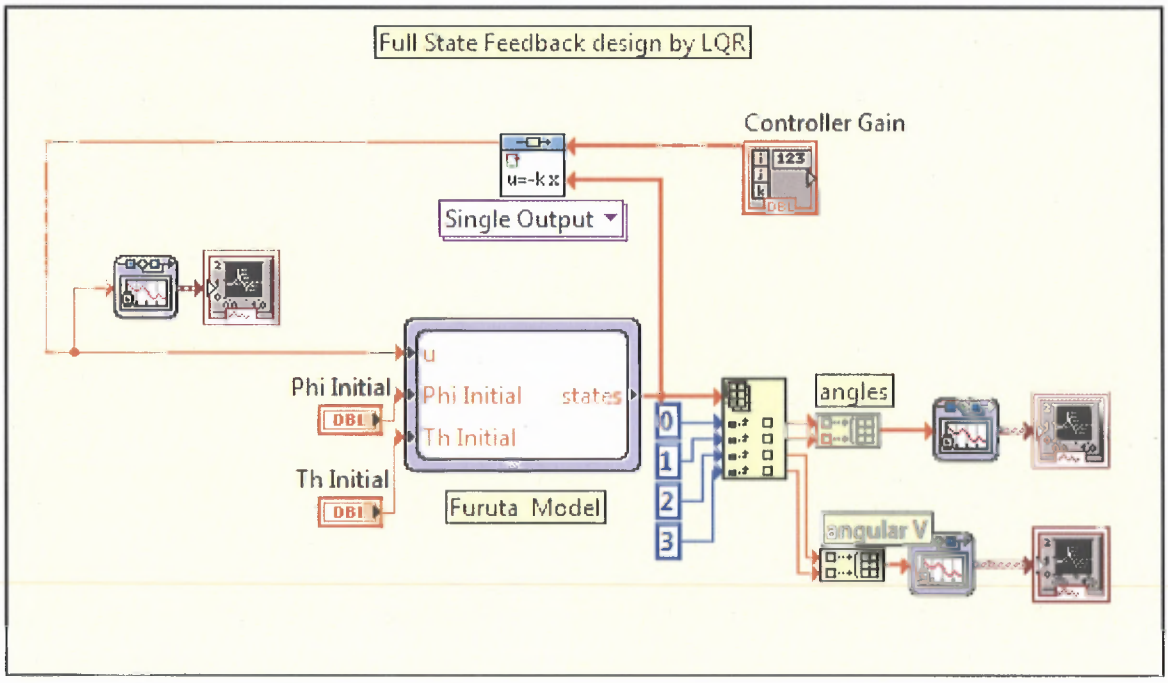


Figure 3.26 Block diagram of the LQR VI for the Furuta Pendulum.

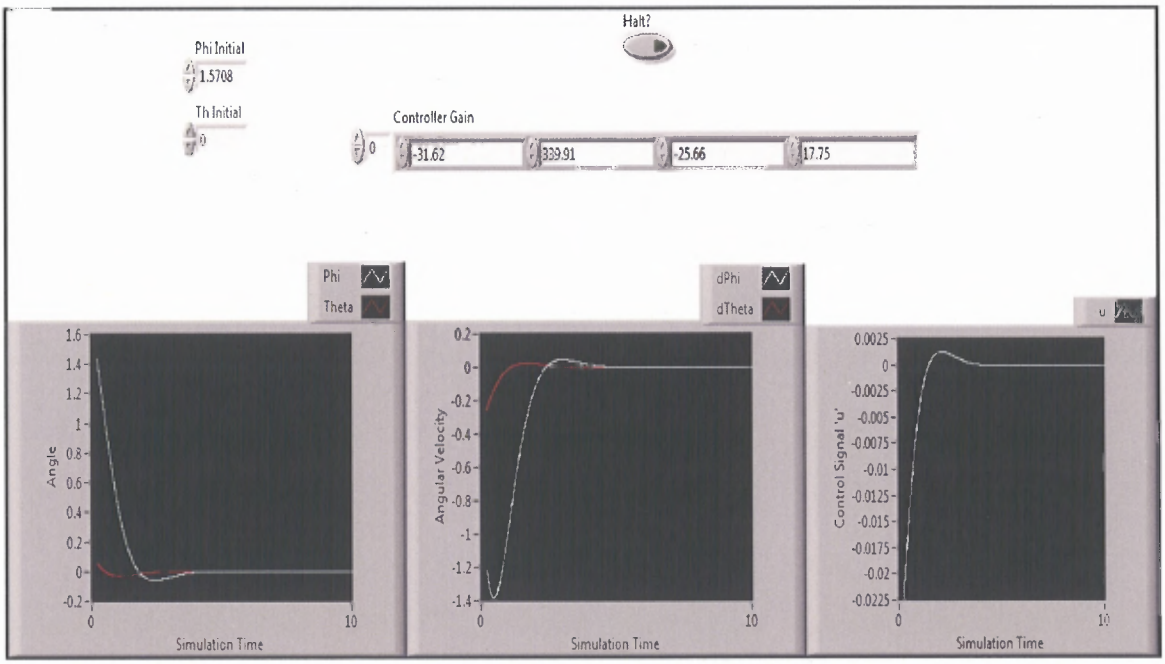


Figure 3.27 Front panel of the LQR VI for the Furuta Pendulum.

Figure 3.26 shows the block diagram of the LQR VI for the Furuta Pendulum system. It is same as the Pole Placement VI of the Furuta Pendulum with CD State Feedback Controller VI in feedback loop. Figure 3.27 shows front panel of the LQR VI for the Furuta Pendulum system. It shows transient responses of the arm position (Phi), pendulum position (Theta), arm velocity (dPhi), pendulum velocity (dTheta) and control signal (u). All these transient responses are same as obtained in Simulink.

3.3 Scicos-Scilab

The Section 3.3 presents the open loop and the close loop analysis of the Furuta Pendulum system using an open source Scicos-Scilab software engines. Scicos is a graphical dynamical system modeler and simulator developed in the Metalau project at INRIA, Paris-Rocquencourt center. Scicos is developed in and distributed with the scientific software package ScicosLab. ScicosLab is a free software package providing a multi-platform environment for scientific computation. ScicosLab is an extended Scilab version, the latest stable and tested version of Scicos . With Scicos, it is possible to create block diagrams to model and simulate the dynamics of hybrid dynamical systems and compile models into executable code.

Scilab is a high-level, numerically oriented programming language. The language provides an interpreted programming environment, with matrices as the main data type. By utilizing matrix-based computation, dynamic typing, and automatic memory management, many numerical problems may be expressed in a reduced number of code lines. As the syntax of Scilab is similar to MATLAB, Scilab includes a source code translator for assisting the conversion of code from MATLAB to Scilab. Scilab is available free of cost under an open source license. [10]

3.3.1 Open Loop Analysis

The Section 3.3.1 presents the open loop analysis of the Furuta Pendulum using ScicosLab 4.4b7. It shows modeling of the Furuta Pendulum using Scicos tools. Scicos (Scilab Connected Object Simulator) is a Scilab package for modeling and simulation of dynamical systems including both continuous and discrete sub-systems. It is quite

similar to Simulink and LabVIEW Simulation module. The blocks that are used to build the mathematical model to be simulated are organized in palettes. Figure 3.28 shows Scicos block diagram of the open loop Furuta pendulum model. It has been devised using different blocks like `Constant` block, `SuperBlock`, `Demux` and `Scope` from palettes. The SuperBlock opens up a new Scicos window for editing a new block diagram. This diagram describes the non linear model of Furuta Pendulum. This SuperBlock is similar to subsystem block of Simulink. The Constant block is a constant value generator. It generates control signal with zero value which is input to the Furuta Pendulum SuperBlock. The Scope block is used to plot the position and velocity signals as the simulation runs. The outputs of the SuperBlock are states which are passed through Demux block. It splits the state vector into angular positions and angular velocities. [10] [12]

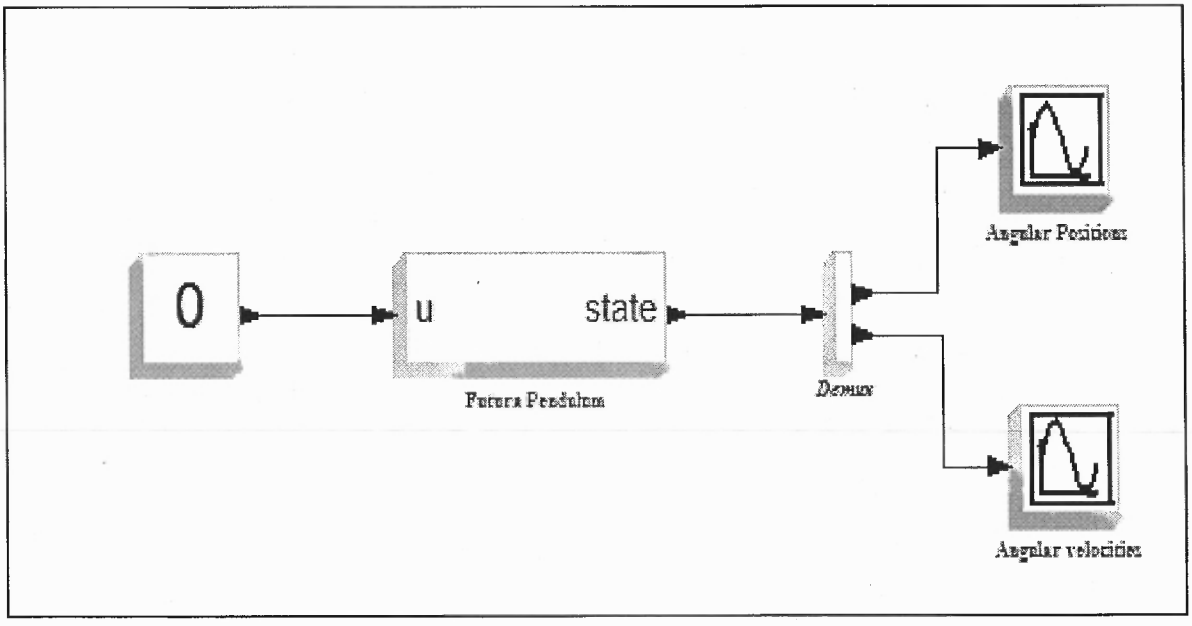


Figure 3.28 Scicos open loop block diagram of the Furuta Pendulum.

Figure 3.29 shows the Furuta Pendulum SuperBlock. It consists of different blocks like Scifunc, Integrators and Mux. The integrator is used to time-integrate signals that are time-derivatives of the state-variables of the system. The integrator outputs are then state-variables of the system. The initial conditions for the arm and the pendulum are set on the second integrator. The Mux block (on the Branching palette) is used to merge number of scalar signals into to be plotted in one Scope. The Scifunc block can realize any type of Scicos block. The function of this block is defined interactively using dialogue boxes and in Scilab language. During simulation, these instructions are interpreted by Scilab; the simulation of diagrams that include these types of blocks is slower. It is similar to MATLAB Fcn block available in Simulink. The Scifunc block contains non linear dynamics of and equations of motion of the Furuta pendulum system. [12] [13]

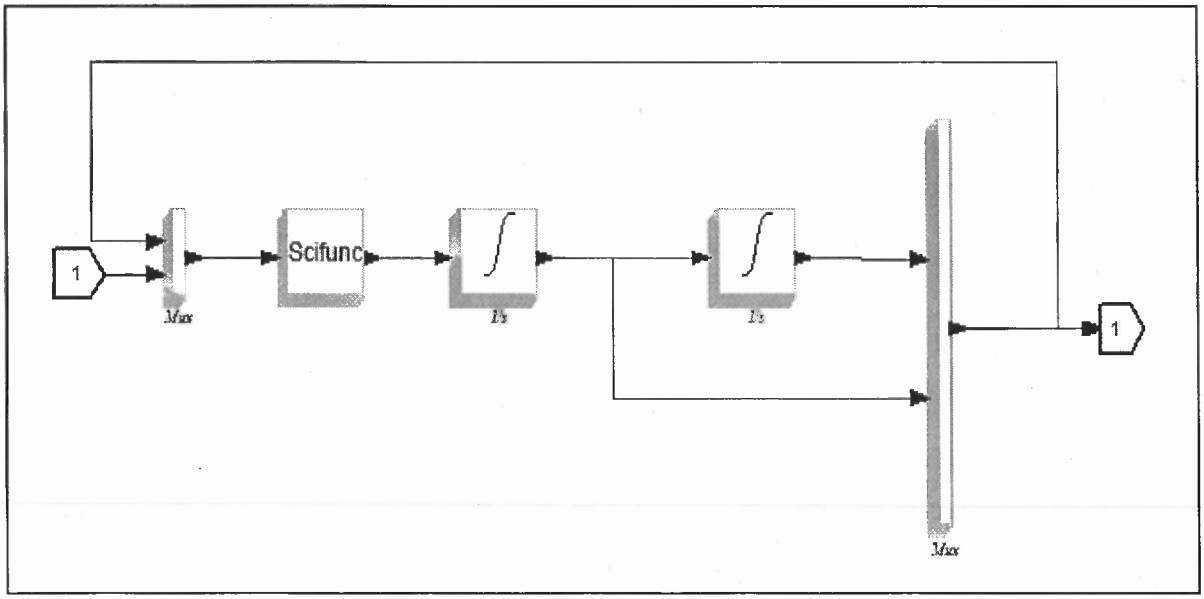


Figure 3.29 Scicos SuperBlock containing non linear model of the Furuta Pendulum.

Different parameters like number of input-out ports have been specified inside dialogue box of Scifunc block. It also contains following Scilab code describing equations of motions of the Furuta pendulum. [10] [11]

```
J=.001;
L=.2;
r=.3;
m=.05;
g=9.8;
c1=m*L*r;
c2=m*L^2;
c3=c2/2;
c4=m*g*L;
a=.01;
//
x=u1
ph=x(1);      // phi
th=x(2);      // theta
dph=x(3);    // phi dot
dth=x(4);     //theta dot
u=x(5);
f1=-c1*sin(th)*dth^2 -c2*sin(2*th)*dth*dph - a*dph + u ;
f2=c3*sin(2*th)*dph^2 +c4*sin(th);
f=[f1;f2];
M=[J+c2*(sin(th))^2+m*r^2,-c1*cos(th);
   -c1*cos(th),c2];
dv=inv(M)*f ;
y1=dv
```

Figure 3.31 shows the transient responses for positions of the arm and the pendulum. As per initial condition the arm position takes off from 1.57 radians and after few oscillations it settles down to its initial position. The Pendulum rod starts with 0.1 radians as per given initial condition and it moves through angle of 3.14 radians and it settles there after tumbling down from initial position. This position behavior of the Furuta Pendulum is exactly same as observed in Simulink open loop simulation.

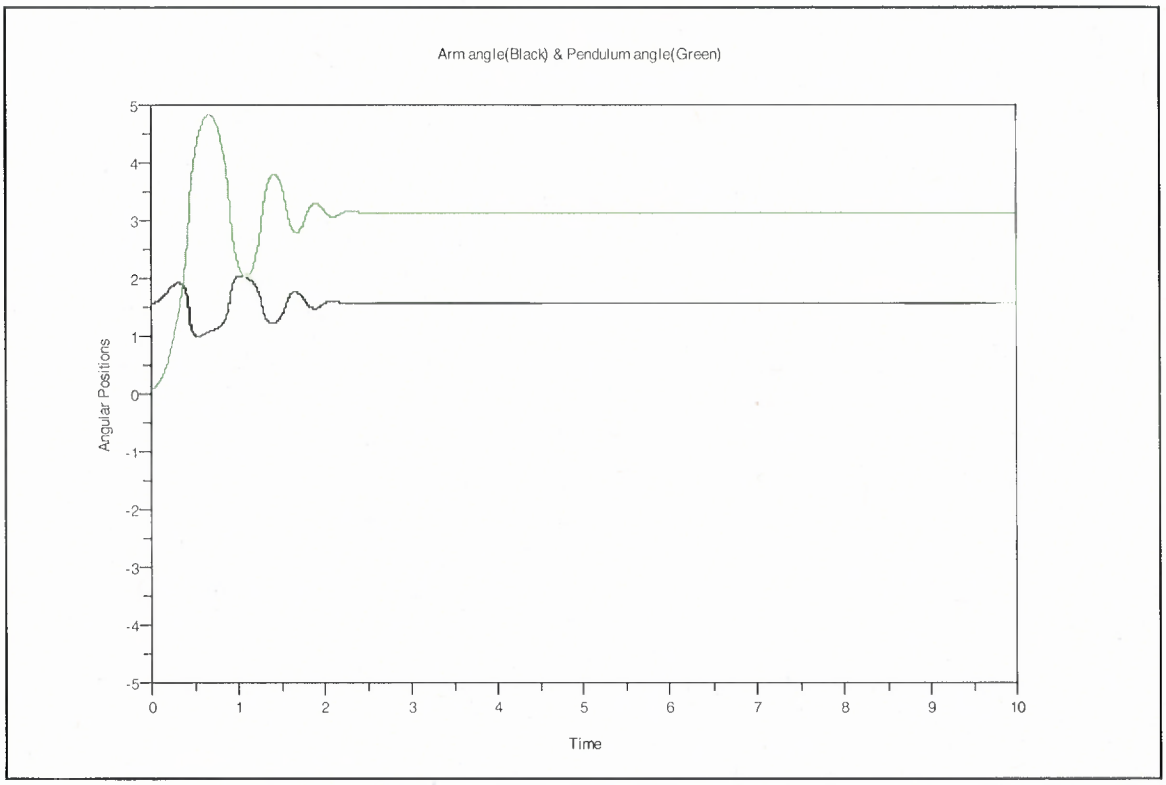


Figure 3.30 Scicos open loop output for an Arm angle (Black) and a Pendulum angle (Green) with initial conditions [1.57, 0.1] on angles.

Figure 3.30 shows transient responses for the arm and the pendulum velocities which have been observed after running simulation for 10 seconds. As per Furuta Pendulum open loop behavior both arm and inverted pendulum should settle down after some time. As shown in Figure 3.31 both the arm velocity and pendulum velocity are settled to zero approximately after 2.7 seconds with few initial vigorous oscillations. The velocity behavior of the Furuta pendulum in Sciocs is exactly same as observed with Simulink open loop simulation. Both position and velocity open loop behavior of the Furuta system in Scicos is exactly similar to Simulink open loop outputs.

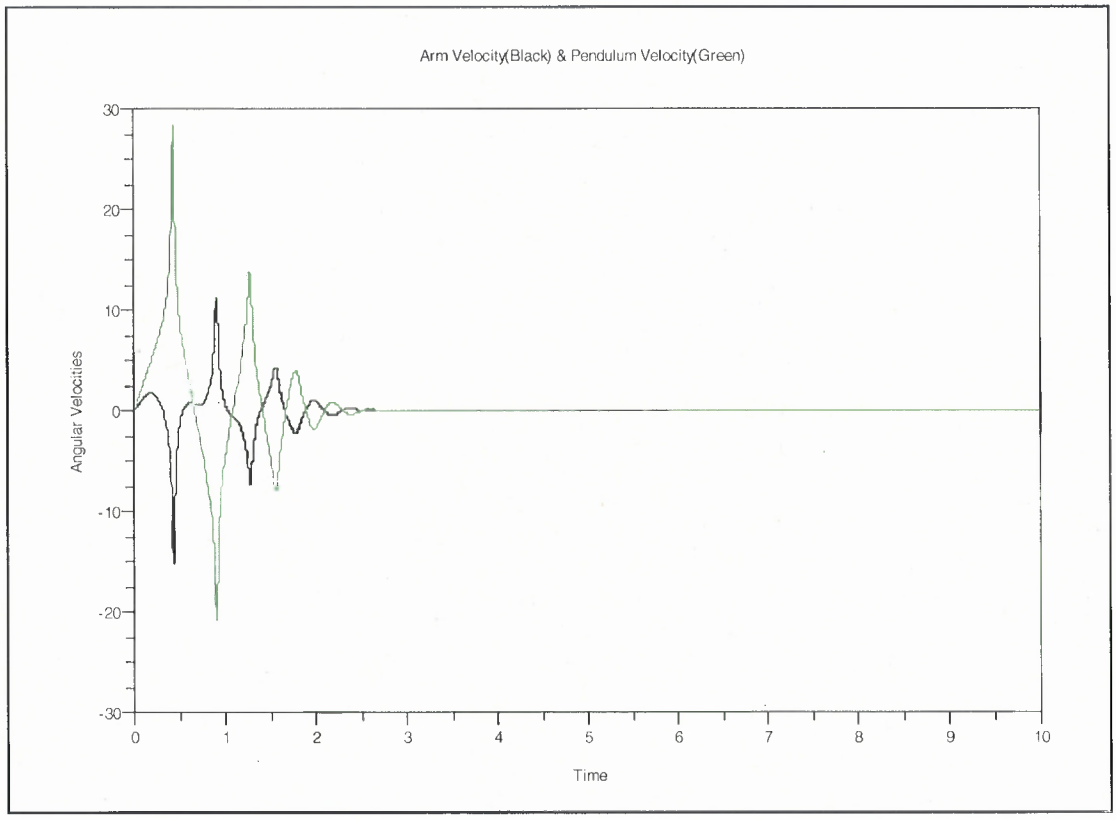


Figure 3.31 Scicos open loop output for the arm velocity (Black) and the pendulum velocity (Green) with initial conditions [1.57, 0.1] on angles.

Linearization:

Scilab has 'lincos' command similar to 'linmod' command in MATLAB for linearization. The lincos command constructs a linear state-space system by linearizing a model given as a Scicos diagram. The output is a Scilab data structure of type continuous-time state-space linear system. The state space model is obtained by executing below Scilab commands. The state space model obtained in Scilab is same as that of LabVIEW. [10]

Scilab Code:

```
-load pend.cos

-lincos(scs_m)
```

```
ans  =

      ans(1)    (state-space system:)

!lss  A  B  C  D  X0  dt  !


ans(2) = A matrix =

 - 10.     0.      0.     147.
 - 15.     0.      0.     269.5
   1.      0.      0.     0.
   0.      1.      0.     0.

      ans(3) = B matrix =

   1000.
   1500.
   0.
   0.

      ans(4) = C matrix =

   0.     0.     1.     0.
   0.     0.     0.     1.
   1.     0.     0.     0.
   0.     1.     0.     0.

      ans(5) = D matrix =

   0.
   0.
   0.
   0.

      ans(6) = X0 (initial state) =

   0.
   0.
   0.
   0.

      ans(7) = Time domain =

   c
```

3.3.2 Full State Feedback design by Pole Placement

The Section 3.3.2 presents full state feedback control design with Pole Placement for the Furuta Pendulum system. The Control Design Toolbox of Scilab has been used for Pole Placement design. It also shows simulation results obtained with Scicos for Pole Placement control design. Block parameters can be modified by opening the block dialogs. This can be done using the Open/set button. Most blocks have dialog menus which can be used to set or modify block parameters. These parameters can be defined using valid Scilab expressions. Scilab variables can be used in the definition of these expressions if they are already defined in the context of diagram. These expressions are memorized symbolically, and then evaluated. [11]

The context of the diagram can be edited by selecting the Context button. The context is evaluated by the Eval button. This is necessary only if the context modification includes a change in the value of a variable previously used in the definition of a block parameter. Scilab has 'ppol' command which returns a gain matrix K such that the eigen values of A-B*K are desired pole locations. The pair (A, B) must be controllable. If the desired poles are complex numbers then it must appear in conjugate pairs. The Scilab code to compute gain matrix K has been shown below. It has been used in the context of the Figure 3.32 to run the Pole Placement control. It is quite similar to M-file script of MATLAB. [11] [14]

Scilab Code:

```
A=[-10 0 0 147; -15 0 0 269.5; 1 0 0 0; 0 1 0 0];
B=[1000;1500;0;0];
k=ppol(A,B,[-10+10*%i,-10-10*%i,-1,-5.56]);
```

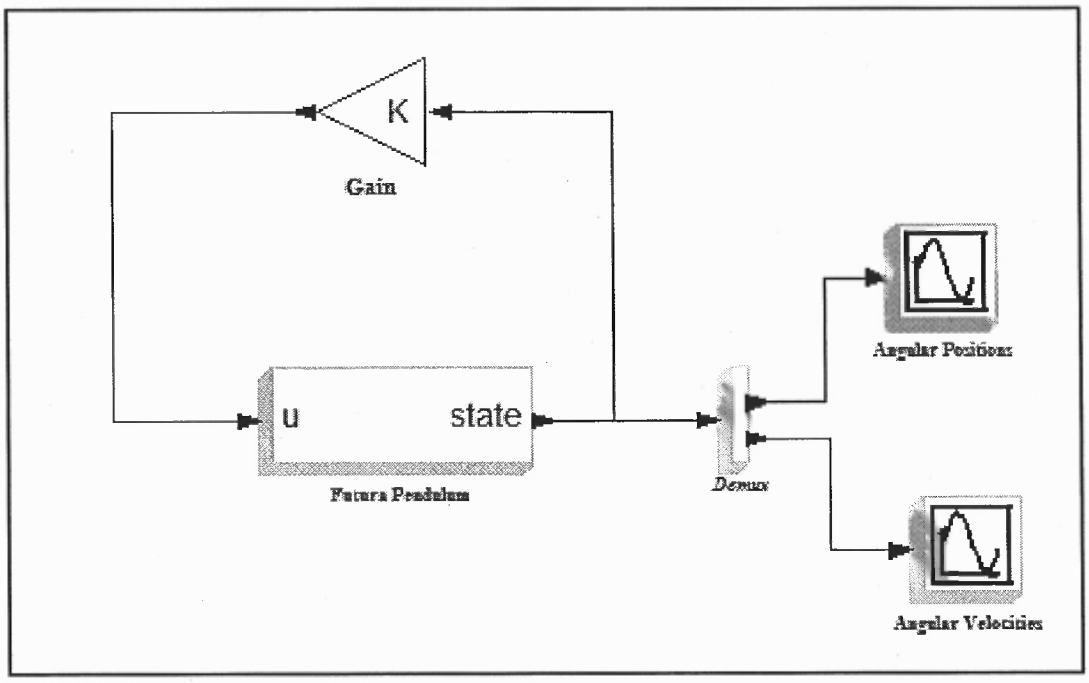


Figure 3.32 Scicos block diagram for Full State Feedback design by Pole Placement.

Figure 3.32 shows the block diagram of Pole Placement control for the Furuta Pendulum system. It is same as compared to the Simulink Pole Placement block diagram. The Gain block multiplies the input state vector by matrix gain. After remapping of gain matrix, K = [-0.0227 0.4193 -0.0390 0.0371] has been used to implement state feedback control law $u = -Kx$. Figure 3.33 shows transient responses of the arm position and the pendulum position. The position behavior of the Scicos transients is same as that of Simulink. Figure 3.34 shows velocity transients of the Furuta pendulum system for the designed Pole Placement control. It is same as obtained in Simulink. The Scope block of Scicos has Graphics Editor which provides great flexibility for axes labeling, simulation time, polyline color and defining figure title. This is an advantage over Simulink scope block. [12] [14]

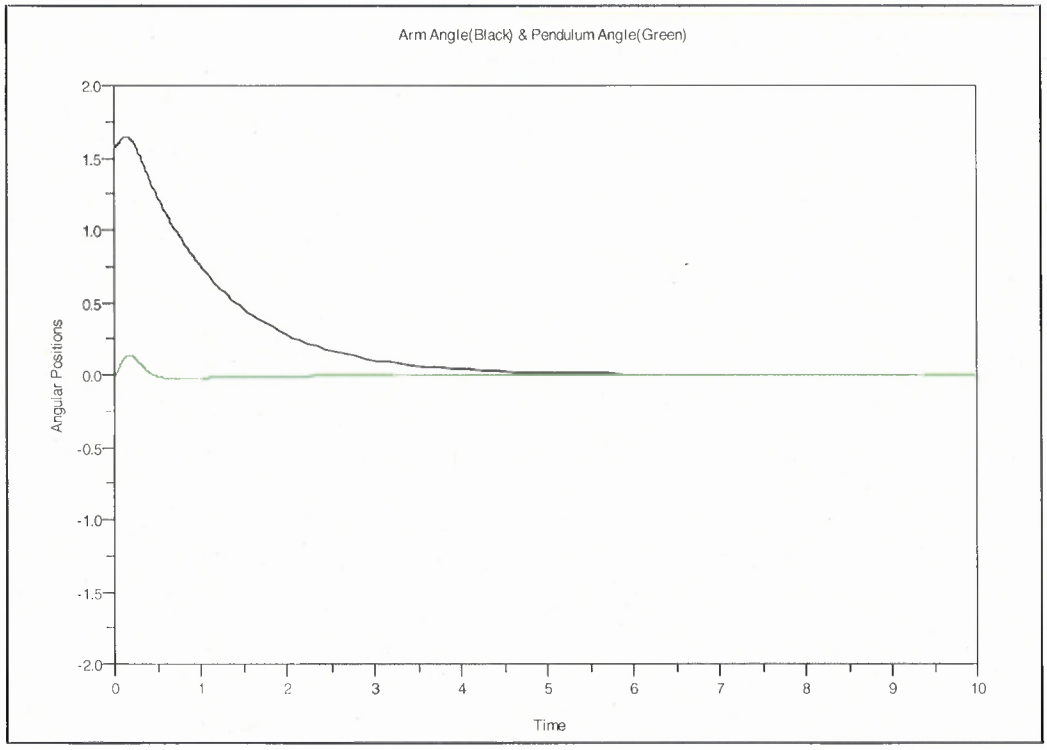


Figure 3.33 Scicos Pole Placement output for the Arm angle (Black) and the Pendulum angle (Green) with initial conditions [1.57, 0] on angles.

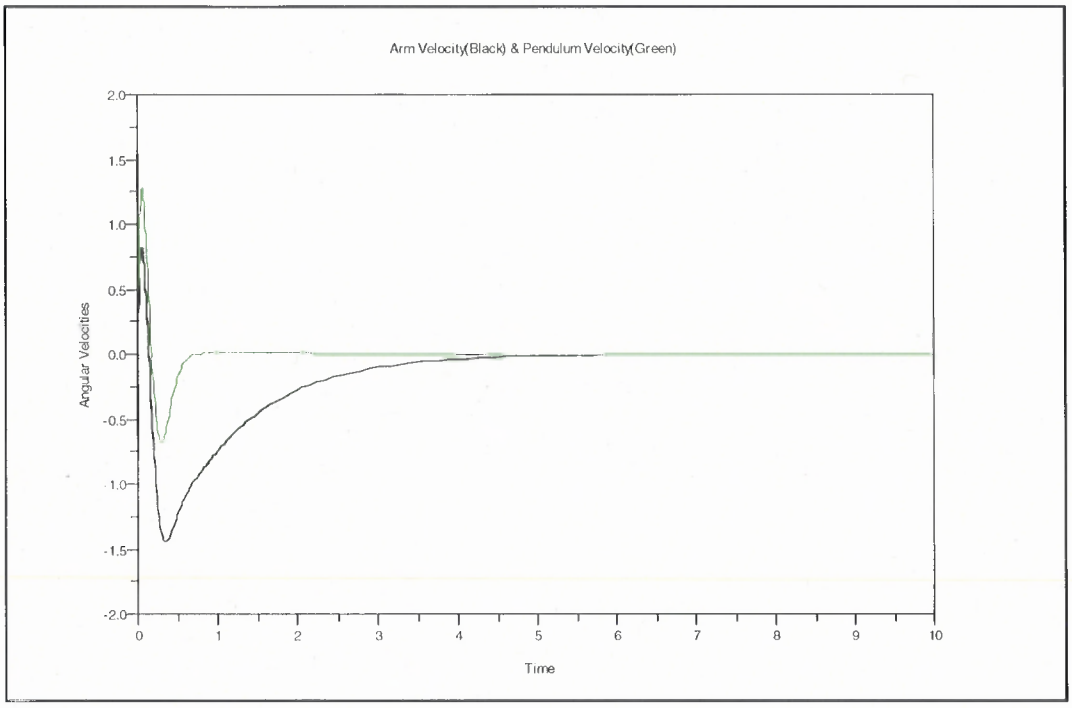


Figure 3.34 Scicos Pole Placement output for the Arm velocity (Black) and the pendulum velocity (Green) with initial conditions [1.57, 0] on angles.

3.3.3 Full State Feedback design by LQR:

The Section 3.3.3 describes Full State Feedback control design using Linear-quadratic (LQ) state-feedback regulator for state-space system. The Control Design Toolbox of Scilab has been used to design LQR control. It also presents simulation results for LQR control simulated using Scicos blocks. The Scilab code to design LQR control is as shown below. It has been used in the context of the Figure 3.37. [14]

```
A=[-10  0  0  147;  -15  0  0  269.5;  1  0  0  0;  0  1  0  0];
B=[1000;1500;0;0];
q1=1000
q2=100000
Q=diag([0,0,q1,q2]);
R=1;
Big=sysdiag(Q,R);
[w,wp]=fullrf(Big);
C1=wp(:,1:4);
D12=wp(:,5:$);                          //[C1,D12]'*[C1,D12]=Big
P=syslin('c',A,B,C1,D12);
[K,X]=lqr(P)

K =

  -25.6565   17.7497 -31.6228 339.9081
```

The Scilab command 'lqr' computes the linear optimal LQ full-state gain for the plant P12=[A,B2,C1,D12] in continuous or discrete time. P12 is a syslin list (e.g. P12=syslin('c',A,B2,C1,D12)). The cost function is l2-norm of z'*z with z=C1 x + D12 u i.e. [x,u]' * BigQ * [x;u]

where $BigQ = \begin{bmatrix} C1' \\ D12' \end{bmatrix} * \begin{bmatrix} C1 & D12 \end{bmatrix} = \begin{bmatrix} Q & S \\ S' & R \end{bmatrix}$

The gain K is such that A + B2*K is stable. X is the stabilizing solution of the Riccati equation. After remapping of gain matrix, K = [-31.6228 339.9081 -25.6565 17.7497] has been used in the Gain block. Figure 3.35 shows the block diagram of LQR control for the Furuta pendulum system. It is same as the Pole Placement control of the Furuta pendulum with Scicos. [14]

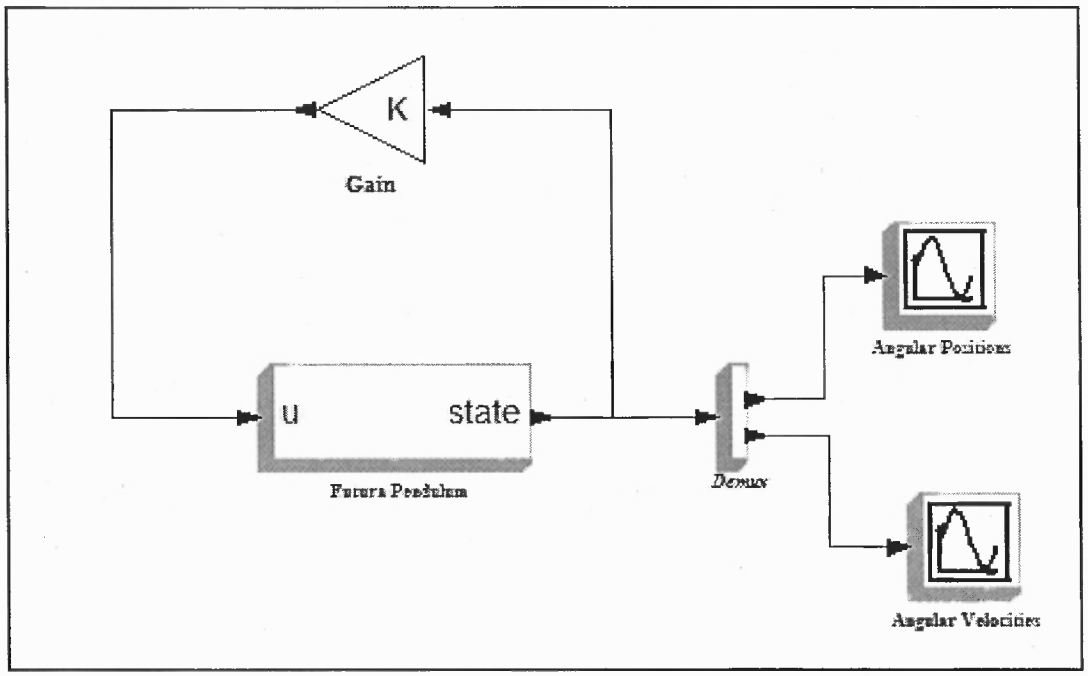


Figure 3.35 Scicos block diagram for Full State Feedback design by LQR.

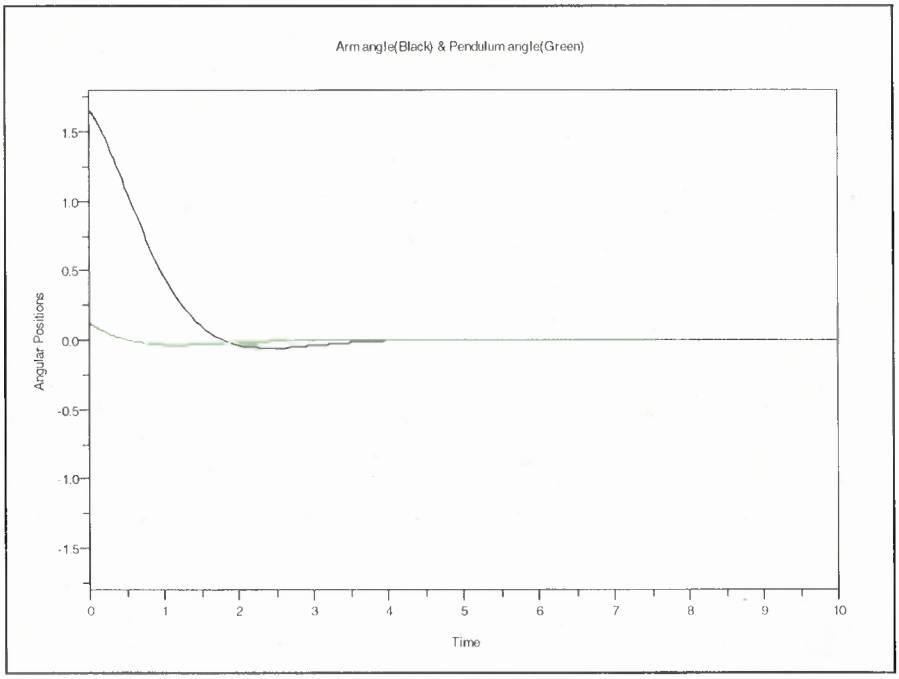


Figure 3.36 Scicos LQR output for the arm angle (Black) and the pendulum angle (Green) with initial conditions [1.57, 0] on angles.

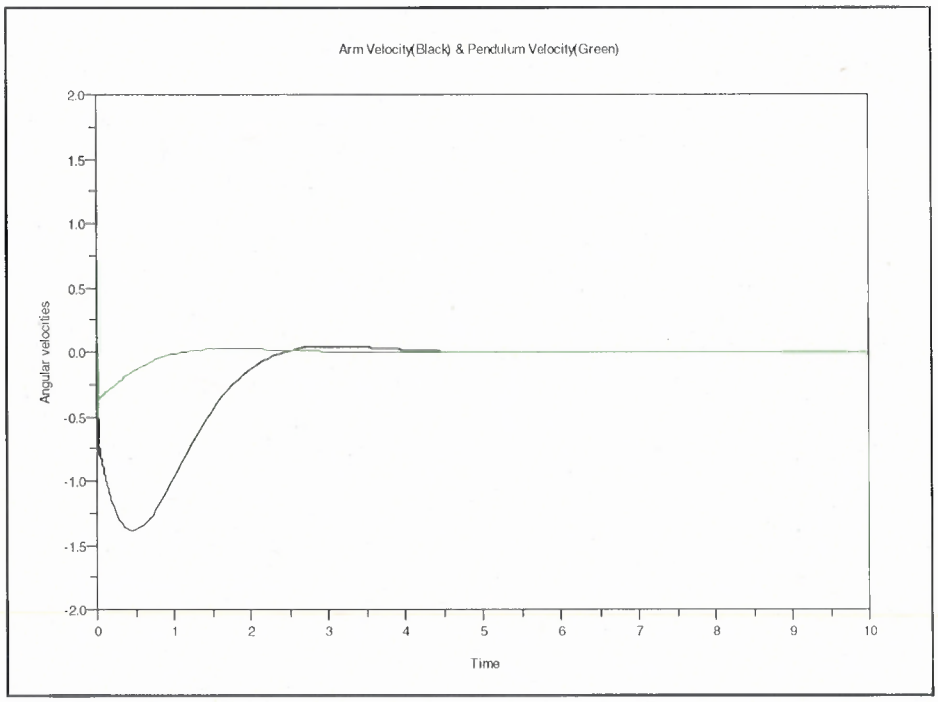


Figure 3.37 Scicos LQR output for the arm velocity (Black) and the pendulum velocity (Green) with initial conditions [1.57, 0] on angles.

Figure 3.36 shows transient responses of the arm position and the pendulum position. It validates the fact that the inverted pendulum has been balanced on the arm. The position behavior of the Scicos transients is same as that of Simulink. Figure 3.37 shows velocity transients of the Furuta pendulum system for the designed LQR control. The arm and the pendulum velocities settle to 0 in accordance with angular positions. All these transient responses are same as obtained with Simulink LQR simulation.

3.3.4 Full Order Observer Design using Pole Placement

The Section 3.3.4 presents Full Order Observer design with Kalman filter. It also describes Scicos simulation results for Kalman Observer. The Scilab code to design feedback gain and Observer state space dynamics is as shown below. It has been used in the context of Figure 3.41.

```
C=[eye(2,2),zeros(2,2)]
scs_m=scs_m.objs(19).model.rpar;
[X,U,Y,XP]=steadycos(scs_m,[],[],[],[],1,1:$);
sys= lincos(scs_m,X,U);
sys=-C*sys
Kc=-ppol(sys.A,sys.B,[-1,-1,-1,-1]*10);
Kf=-ppol(sys.A',sys.C',[-2,-2,-2,-2]*5);Kf=Kf';
Contr=obscont(sys,Kc,Kf);
[Ac,Bc,Cc,Dc]=abcd(Contr)
clear('scs_m','X','Y','XP','Kc','Kp','sys','Contr')
```

The Scilab command 'obscont' returns the observer-based controller associated with a nominal plant P with matrices [A,B,C,D] (syslin list). The full-state control gain is Kc and the filter gain is Kf. These gains have been computed using pole placement. A+B*Kc and A+Kf*C are (usually) assumed stable. This Obsever design is based on pole placement. Figure 3.38 shows block diagram of Observer based control design for

the Furuta Pendulum. The gain block contains matrix C= [1 0 0 0; 0 1 0 0] which feeds back the arm and the pendulum position to Observer. The Observer sub-system contains state space model block which contains observer dynamics Ac, Bc, Cc, Dc. [11] [14]

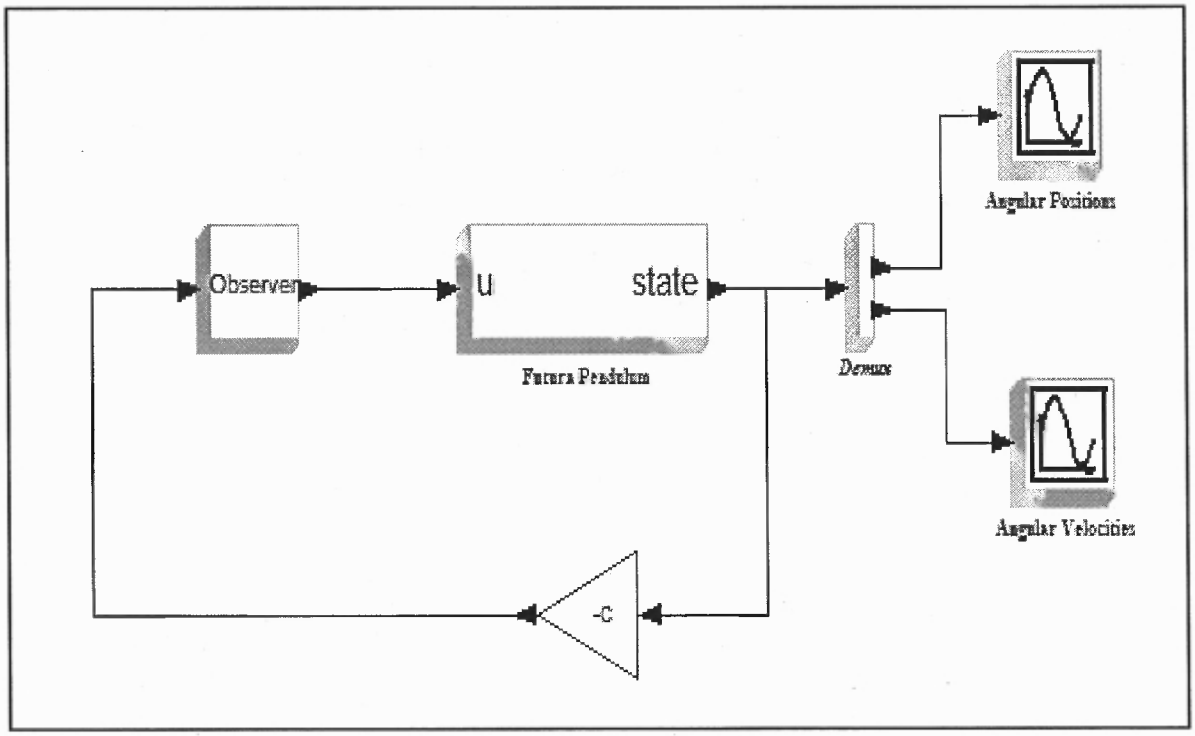


Figure 3.38 Scicos block diagram for full order Observer design with Kalman Filter.

The transient responses for the arm position and the pendulum position with the full order Observer in feedback are as shown in Figure 3.39. With an initial condition of 0.78 radians, the arm moves to a small angle in the opposite direction and then comes to the steady state in about 2 seconds. The pendulum also deflects by some angle before coming to zero in about 2 seconds. Since both positions are settled at zero it verifies the fact that Observer design successfully balanced inverted pendulum on the arm. Figure 3.40 shows transient responses for the arm velocity and the pendulum velocity with Observer in feedback. These are the estimated arm and pendulum velocity states from the arm and the pendulum positions feedbacks. The position and velocity profiles validate the

fact that Observer can estimate velocity states based on position feedbacks and in turn it could successfully balance inverted pendulum on the arm.

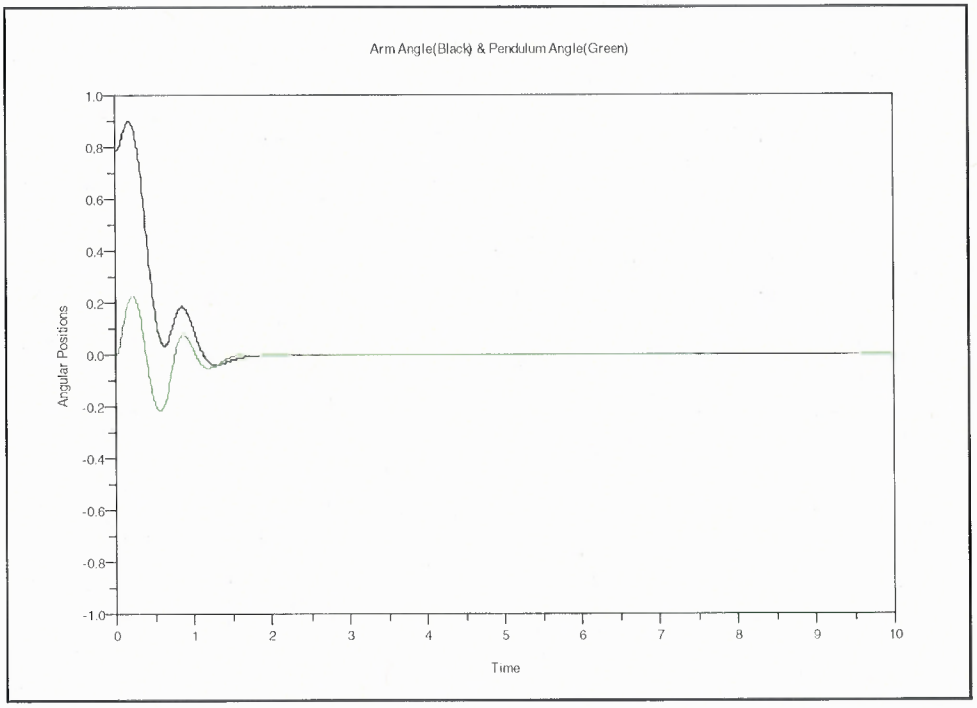


Figure 3.39 Scicos Kalman Observer output for the Arm angle (Black) and the Pendulum angle (Green) with initial conditions [0.78, 0] on angles.

Figure 3.41 shows control signal generated by Full Order Observer. The control initially follows a transient with oscillations about zero. It finally reaches zero after few oscillations. At this moment a steady state is attained by the system. It has been observed that states settle to steady state in accordance with Observer control signal.

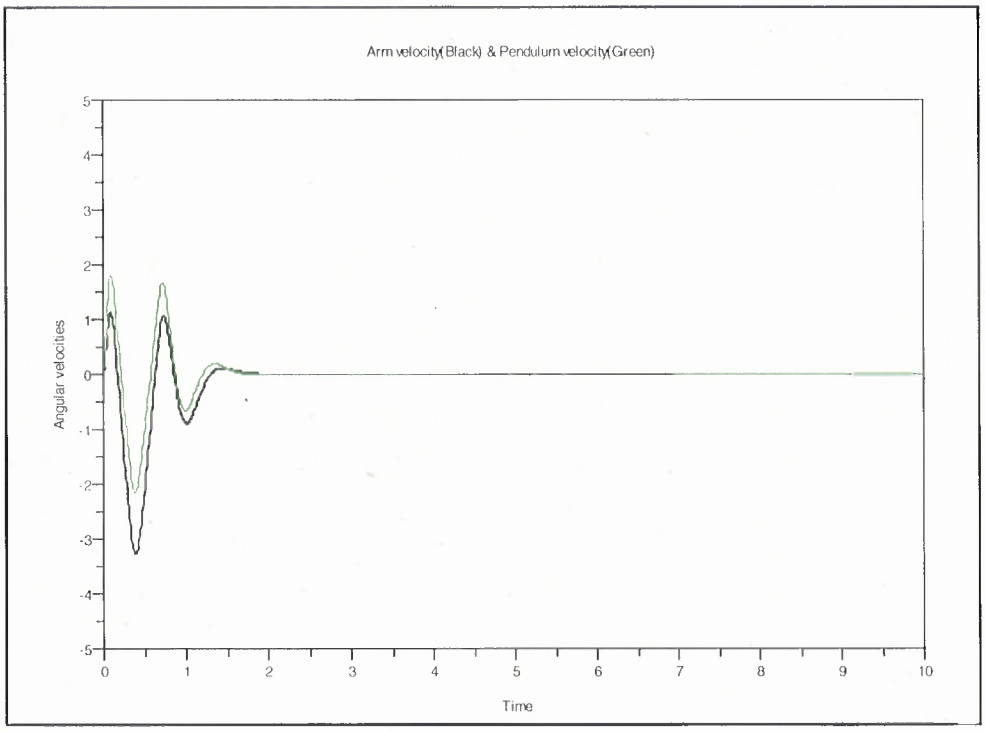


Figure 3.40 Scicos Kalman Observer output for the Arm velocity (Black) and the Pendulum angle (Green) with initial conditions [0.78, 0] on angles.

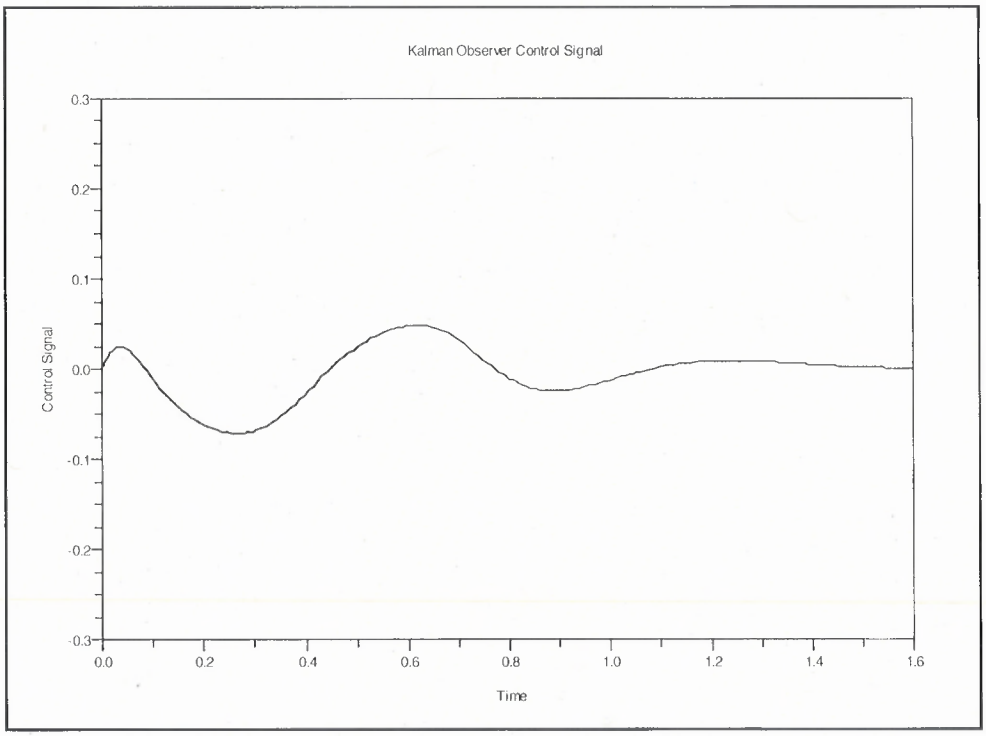


Figure 3.41 Scicos output for the control signal generated with Kalman Observer.

3.4 Maple -MapleSim

The Section 3.4 presents the Maple-MapleSim software packages from Maplesoft which are powerful systems that can be used to solve mathematical problems from simple to complex. The MapleSim is a modeling environment for creating and simulating complex multi-domain physical systems. It allows you to build component diagrams that represent physical systems in a graphical form. Using both symbolic and numeric approaches, MapleSim automatically generates model equations from a component diagram and runs high-fidelity simulations.

3.4.1 Open Loop Analysis

MapleSim has two different approaches for modeling dynamic systems. 1) Conventional mathematical symbolic modeling. 2) Multi body 'Acausal' modeling. The signal-flow approach used by traditional modeling tools requires system inputs and outputs to be defined explicitly. In contrast, MapleSim allows using physical interconnections based on links to connect interrelated components without having to consider how signals flow between them. When simulated by software, block diagrams can either be 'Causal' or 'Acausal'. Acausal model represents physical configuration of system where as Causal model represents mathematical functions of system. Many simulation tools are restricted to causal (or signal-flow) modeling. In these tools, a unidirectional signal, which is essentially time-varying, flows into a block. The block then performs a well-defined mathematical operation on the signal and the result flows out of the other side. This approach is useful for modeling systems that are defined purely by signals that flow in a single direction, such as control systems. Modeling how real physical components

interact requires a different approach. In Acausal modeling, a signal from two connected blocks travels in both directions. With MapleSim it is possible to start with Acausal model and get system equations. It is also possible to start modeling with equations of motion for particular system. [16]

The Furuta Pendulum Acausal Modeling:

MapleSim has this unique feature of Acausal modeling with physical multi body components. Figure 3.42 shows the Acausal model of the Furuta pendulum. It consists of various mechanical components which forms the complete Furuta Pendulum system. A stationary frame with a fixed displacement and orientation relative to ground has been used as support to the arm. A fixed frame is attached rigidly to the mechanical ground. A *Revolute* is a joint which allows one rotational degree of freedom about a given axis. A *Revolute* joint, sometimes called a pin or hinge, with the two bodies and body-fixed reference frames that it connects. [16]

A *Revolute* joint allows a single relative rotation of the two frames; this joint type prevents all other relative rotations and translations. The initial conditions of 1.57 radians and 0.1 radians have been defined on two revolute joints. The rigid body frame with a fixed displacement and orientation relative to a rigid body center of mass (CoM) has been used. It is connected to the revolute to form the arm of the Furuta Pendulum. The Rigid Body Frame is a body-fixed frame that is used to define locations of interest on the body where it is connected. The position and orientation relative to the center of mass must be defined for each body-fixed frame. The Angle Sensor component generates an output signal proportional to the absolute angle of the attached rotational flange. [16]

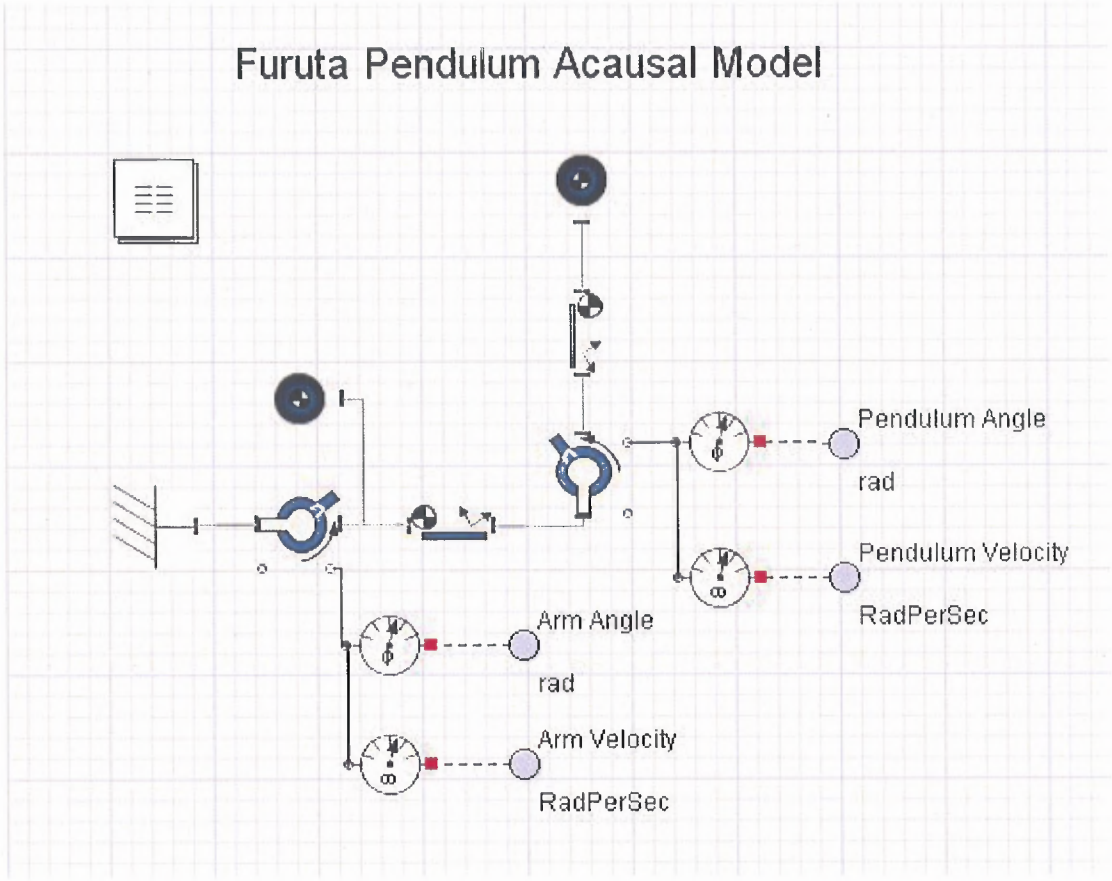


Figure 3.42 MapleSim Acausal model of the Furuta Pendulum.

The Angular Velocity Sensor component generates an output signal proportional to the absolute angular velocity of the attached flange. Same combination of Revolute, Rigid Body Frame, Angular Sensor and Angular Velocity Sensor block has been used to form inverted pendulum connected to the arm. This whole mechanical system represents the Furuta Pendulum. Figure 3.43 shows the position and velocity transients for open loop simulation. These transients are exactly same as obtained in Simulink.

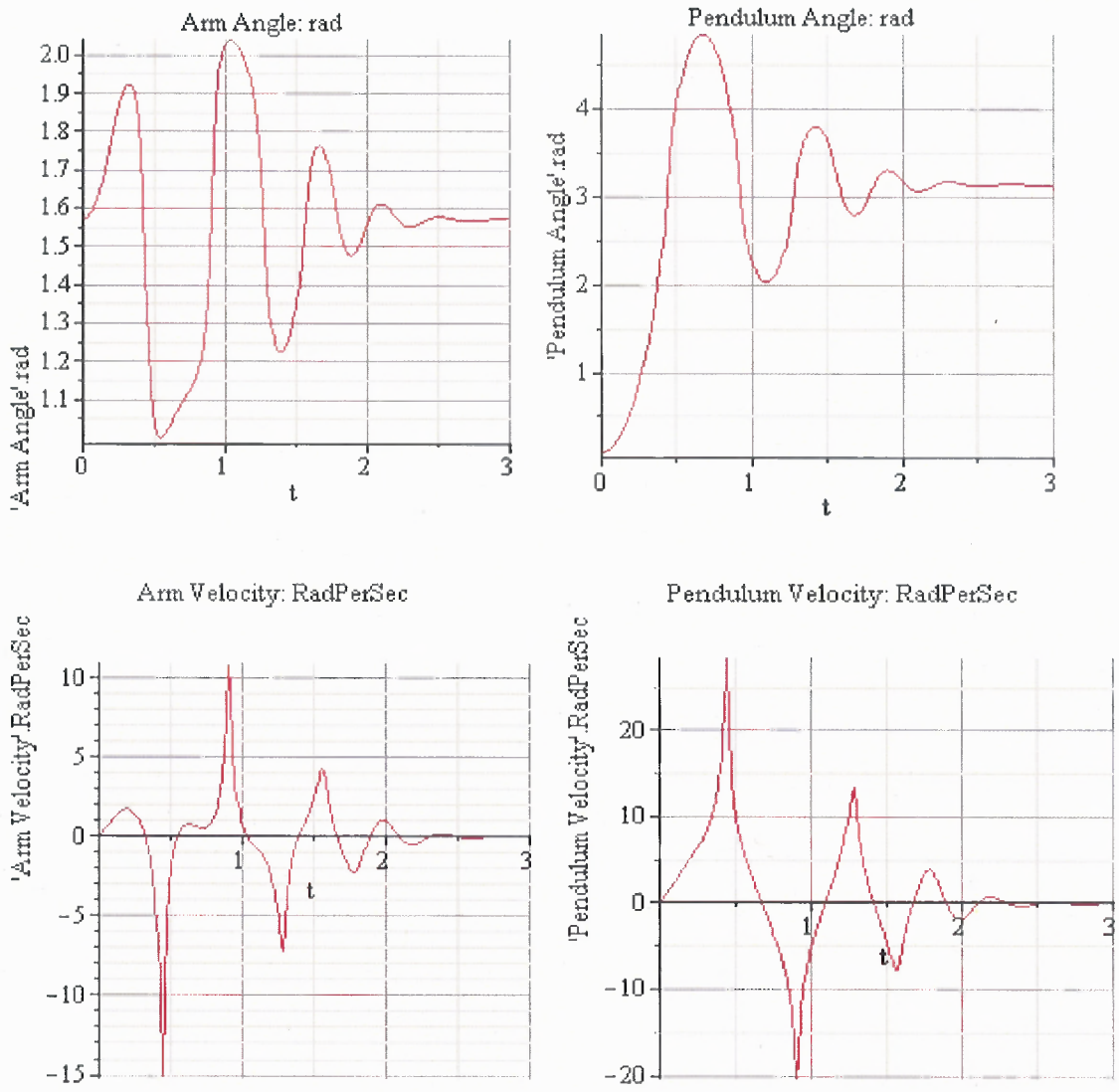


Figure 3.43 MapleSim probe outputs for the open loop Acausal model.

Generating Equations from the Acausal Furuta Pendulum Model:

MapleSim has unique feature of generating system equations from the Acausal model of the system. A topological representation maps readily to its mathematical representation and the symbolic capability of MapleSim automates the generation of system equations. When MapleSim formulates the system equations, several mathematical simplification tools are applied to remove any redundant equations and multiplication by zero or one. The simplification tools then combine and reduce the expressions to get a minimal set of equations required to represent a system without losing fidelity. As a result, it was found that there were discrepancies as compared to original available dynamics of the Furuta Pendulum. Due to this capability of MapleSim, it was possible to analyze and rectify equations of motion for the Furuta Pendulum. The Maple worksheet to develop equations of motion for the Furuta Pendulum has been discussed below.

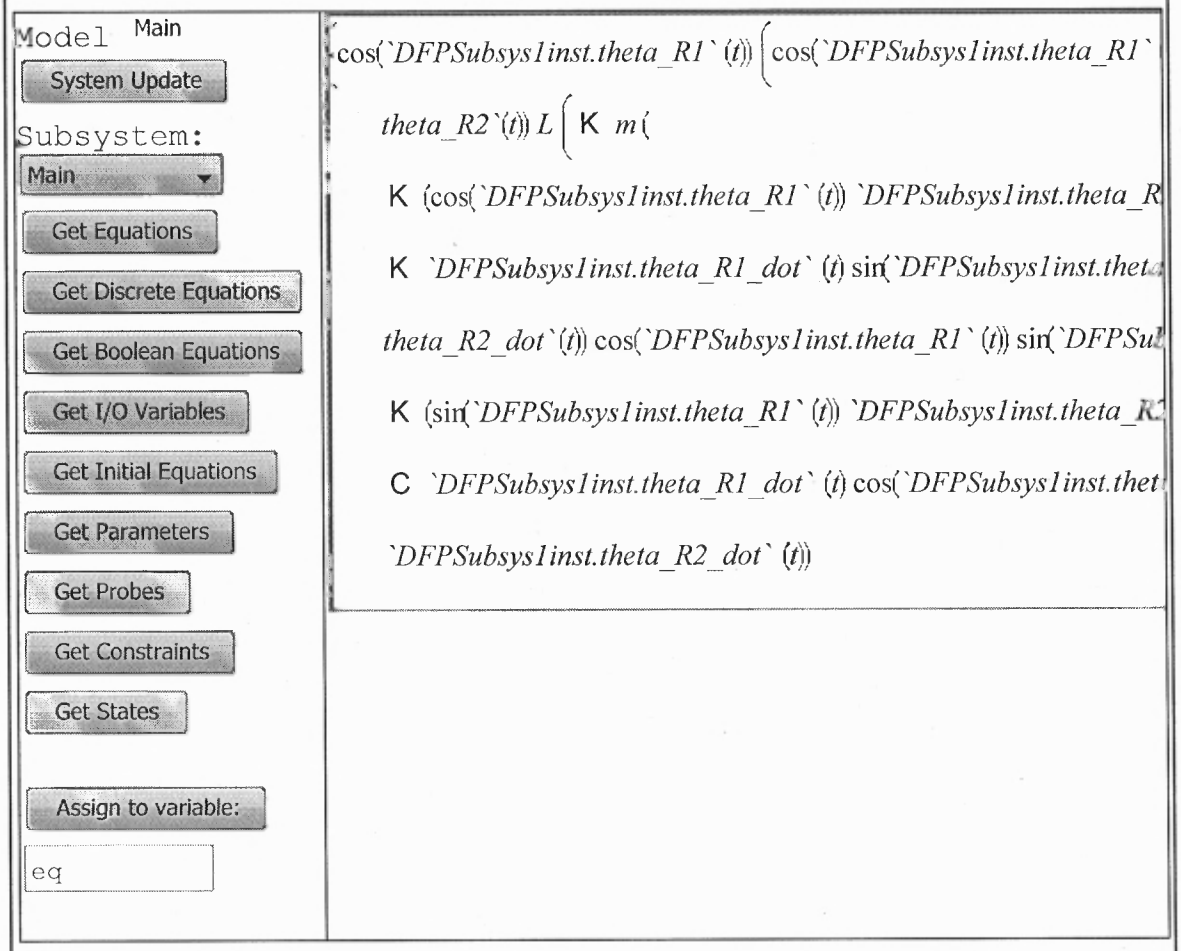
Maple Equation Worksheet [15]:

```
Description

Use this template as a starting point for performing
advanced analysis on MapleSim physical models. This
template allows you to retrieve equations to gain insight
into the behavior of your model.

Note: The ability to retrieve equations is currently
limited to continuous subsystems.
Model Diagram
```

Model Equations



Number of equations in the system:

$nops(eq)$

8

Take a look at the first one:

$eq[1]$

Define simple name for viewing:

$$sMap := \{`DFPSubsys1inst.theta\_R1` = theta1,\ `DFPSubsys1inst.theta\_R2` = theta2,\ `DFPSubsys1inst.theta\_R1\_dot` = theta1d,\ `DFPSubsys1inst.theta\_R2\_dot` = theta2d,\ `DFPSubsys1inst.theta\_R1\_ddot` = theta1dd,\ `DFPSubsys1inst.theta\_R2\_ddot` = theta2dd\}:$$

Apply name change and simplify

```
neweq := subs(sMap, map(simplify, eq)) :
Look at the first two equations:
collect(neweq[1], [theta1dd(t), theta2dd(t), cos, sin])
```

$$-L\,m\cos(\theta 2(t))\,r\,theta1dd(t) + L^2\,m\,theta2dd(t) - L^2\,m\,theta1d(t)^2\sin(\theta 2(t))\cos(\theta 2(t)) - \frac{981}{100}\,L\,m\sin(\theta 2(t))$$

```
collect(neweq[2], [theta1dd(t), theta2dd(t), L^2, m, cos, sin])
```

$$\left(\left(1-\cos(\theta 2(t))^2\right)m\,L^2 + r^2\,m + J\right)theta1dd(t) + r\,m\,theta2d(t)^2\sin(\theta 2(t))\,L + a\,theta1d(t) - r\,m\cos(\theta 2(t))\,L\,theta2dd(t) + 2\sin(\theta 2(t))\,L^2\,m\cos(\theta 2(t))\,theta1d(t)\,theta2d(t)$$

```
The rest of the equations:
map(lprint, neweq[3..-1])
diff(theta1(t), t) = theta1d(t)
diff(theta1d(t), t) = theta1dd(t)
diff(theta2(t), t) = theta2d(t)
diff(theta2d(t), t) = theta2dd(t)
`psi1.deg`(t) = `'AS1::CB'.k`*theta1(t)
`psi2.deg`(t) = `'AS2::CB'.k`*theta2(t)
{}
```

Creating Furuta Custom Modeling Component:

It is possible to create custom modeling components based on mathematical models with equations of motion. It is also possible to create a custom component to contain a particular subsystem by implementing system equations. By using the Custom Component Template, which is a Maple worksheet available through the MapleSim templates dialog box, the non linear Furuta pendulum custom component has been developed. It defines the equations of motion associated with the Furuta System and the parameters that determine the behavior of the system. Different ports have been added to the component and associated port variable mappings have been defined. Further, the

Furuta custom component has been generated and made it available in MapleSim to be used as subsystem in simulation. The Maple worksheet to create the Furuta custom component has been discussed below. [16]

Maple Custom Component Worksheet [15]:

Description

This file implements the nonlinear dynamic equations of the Furuta Pendulum.

Component Description

Enter the name to display in MapleSim after you generate the component. The name must not contain spaces or special characters (for example, & and *).

Component Name: Furuta

Component Equations

System variable:	sys variable name used for storing system object
Parameter variable:	params variable name used for storing component parameters
Initial Conditions variable:	initialconditions variable name used for storing initial equations

Defining Matrix Dynamics Equations

$$matM := \langle \langle J + c2 \cdot \sin(\text{theta}(t))^2 + m \cdot r^2 | -c1 \cdot \cos(\text{theta}(t)) \rangle, \langle -c1 \cdot \cos(\text{theta}(t)) | c2 \rangle \rangle$$

$$\begin{bmatrix} J + c2\,\sin(\theta(t))^2 + m\,r^2 & -c1\,\cos(\theta(t)) \\ -c1\,\cos(\theta(t)) & c2 \end{bmatrix}$$

$$vecF := \langle -c1 \cdot \sin(\text{theta}(t)) \cdot vtheta(t)^2 - c2 \cdot \sin(2 \cdot \text{theta}(t)) \cdot vtheta(t) \cdot vpsi(t) - a \cdot vpsi(t) + u(t), c3 \cdot \sin(2 \cdot \text{theta}(t)) \cdot vpsi(t)^2 + c4 \cdot \sin(\text{theta}(t)) \rangle$$

$$\left[\left[-c1\,\sin(\theta(t))\,vtheta(t)^2 - c2\,\sin(2\,\theta(t))\,vtheta(t)\,vpsi(t) - a\,vpsi(t) + u(t)\right], \left[c3\,\sin(2\,\theta(t))\,vpsi(t)^2 + c4\,\sin(\theta(t))\right]\right]$$

$$vecA := \langle apsi(t), atheta(t) \rangle$$

$$\begin{bmatrix} apsi(t) \\ atheta(t) \end{bmatrix}$$

$$matEq := vecA = LinearAlgebra:\text{-}MatrixInverse(matM)\ .\ vecF$$

$$\begin{bmatrix} apsi(t) \\ atheta(t) \end{bmatrix} = \left[\left[\left(c2\,\left(-c1\,\sin(\theta(t))\,vtheta(t)^2 - c2\,\sin(2\,\theta(t))\,vtheta(t)\,vpsi(t) - a\,vpsi(t) + u(t)\right)\right) \Big/ \left(c2\,J + c2^2\,\sin(\theta(t))^2 + c2\,m\,r^2 - c1^2\,\cos(\theta(t))^2\right) + \frac{c1\,\cos(\theta(t))\,\left(c3\,\sin(2\,\theta(t))\,vpsi(t)^2 + c4\,\sin(\theta(t))\right)}{c2\,J + c2^2\,\sin(\theta(t))^2 + c2\,m\,r^2 - c1^2\,\cos(\theta(t))^2}\right], \left[\left(c1\,\cos(\theta(t))\,\left(-c1\,\sin(\theta(t))\,vtheta(t)^2 - c2\,\sin(2\,\theta(t))\,vtheta(t)\,vpsi(t) - a\,vpsi(t) + u(t)\right)\right) \Big/ \left(c2\,J + c2^2\,\sin(\theta(t))^2 + c2\,m\,r^2 - c1^2\,\cos(\theta(t))^2\right) + \left(\left(J + c2\,\sin(\theta(t))^2 + m\,r^2\right)\left(c3\,\sin(2\,\theta(t))\,vpsi(t)^2 + c4\,\sin(\theta(t))\right)\right) \Big/ \left(c2\,J + c2^2\,\sin(\theta(t))^2 + c2\,m\,r^2 - c1^2\,\cos(\theta(t))^2\right)\right]\right]$$

$$DynEqs := zip(`=`, convert(lhs(matEq),'list'), convert(rhs(matEq),'list'))$$

Kinematic Equations

$$KinEqs := \{apsi(t) = diff(vpsi(t), t), vpsi(t) = diff(\mathrm{psi}(t), t), atheta(t) = diff(vtheta(t), t), vtheta(t) = diff(\mathrm{theta}(t), t)\}$$

$$\left\{apsi(t) = \frac{d}{dt} vpsi(t), atheta(t) = \frac{d}{dt} vtheta(t), vpsi(t) = \frac{d}{dt} \psi(t), vtheta(t) = \frac{d}{dt} \theta(t)\right\}$$

$$eq := [DynEqs[\,], KinEqs[\,]]$$

$$\left[apsi(t) = \left(c2 \left(-c1 \sin(\theta(t))\right) vtheta(t)^2 - c2 \sin(2\,\theta(t))\right) vtheta(t)\, vpsi(t) - a\, vpsi(t) + u(t)\Big)\Big/\Big(c2\, J + c2^2 \sin(\theta(t))^2 + c2\, m\, r^2 - c1^2 \cos(\theta(t))^2\Big) + \frac{c1 \cos(\theta(t)) \left(c3 \sin(2\,\theta(t))\right) vpsi(t)^2 + c4 \sin(\theta(t))\Big)}{c2\, J + c2^2 \sin(\theta(t))^2 + c2\, m\, r^2 - c1^2 \cos(\theta(t))^2},$$

$$atheta(t) = \Big(c1 \cos(\theta(t)) \left(-c1 \sin(\theta(t))\right) vtheta(t)^2 - c2 \sin(2\,\theta(t))\Big) vtheta(t)\, vpsi(t) - a\, vpsi(t) + u(t)\Big)\Big/\Big(c2\, J + c2^2 \sin(\theta(t))^2 + c2\, m\, r^2 - c1^2 \cos(\theta(t))^2\Big) + \Big(\Big(J + c2 \sin(\theta(t))^2 + m\, r^2\Big) \left(c3 \sin(2\,\theta(t))\right) vpsi(t)^2 + c4 \sin(\theta(t))\Big)\Big)\Big/\Big(c2\, J + c2^2 \sin(\theta(t))^2 + c2\, m\, r^2 - c1^2 \cos(\theta(t))^2\Big), apsi(t) = \frac{d}{dt} vpsi(t), atheta(t) = \frac{d}{dt} vtheta(t), vpsi(t) = \frac{d}{dt} \psi(t), vtheta(t) = \frac{d}{dt} \theta(t)\right]$$

Defining the parameters:

$$params := [J = .001, L = .2, r = .3, m = .05, g = 9.8, c1 = m*L*r, c2 = m*L\hat{}2, c3 = c2/2, c4 = m*g*L, a = .01, psi0 = 0, theta0 = 0]$$

$$\left[J = 0.001, L = 0.2, r = 0.3, m = 0.05, g = 9.8, c1 = m\,L\,r, c2 = m\,L^2, c3 = \frac{1}{2}\,c2, c4 = m\,g\,L, a = 0.01, \psi 0 = 0, \theta 0 = 0\right]$$

Resolving parameters into numeric values:

$$params := map(z \rightarrow lhs(z) = eval(lhs(z), solve(params)), params)$$

$$[J = 0.001000000000, L = 0.2000000000, r = 0.3000000000, m = 0.05000000000, g = 9.800000000, c1 = 0.003000000000, c2 = 0.002000000000, c3 = 0.001000000000, c4 = 0.09800000000, a = 0.01000000000, \psi 0 = 0., \theta 0 = 0.]$$

$$initialconditions := [\text{psi}(0) = 1.57,\ \text{theta}(0) = 0.1]$$

$$[\psi(0) = 1.57, \theta(0) = 0.1]$$

$$sys := DynamicSystems[DiffEquation](eq, inputvariable = [u(t)], outputvariable = [\text{psi}(t), \text{theta}(t), vpsi(t), vtheta(t)])$$

Diff. Equation

continuous

4 output(s); 1 input(s)

inputvariable =$[u(t)]$

outputvariable =$[\psi(t), \theta(t), vpsi(t), vtheta(t)]$

Component Analysis

Use any of the following tools to test and analyze the equations that you entered above.

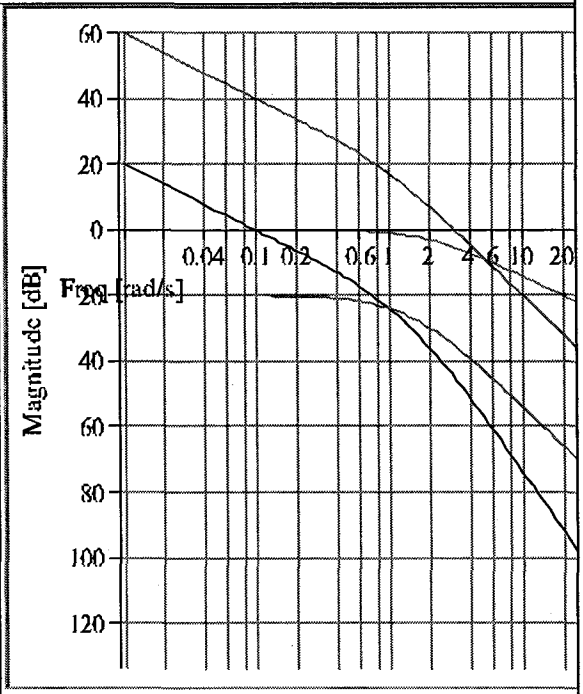
Bode Plot (Magnitude)

Bode Plot (Phase)

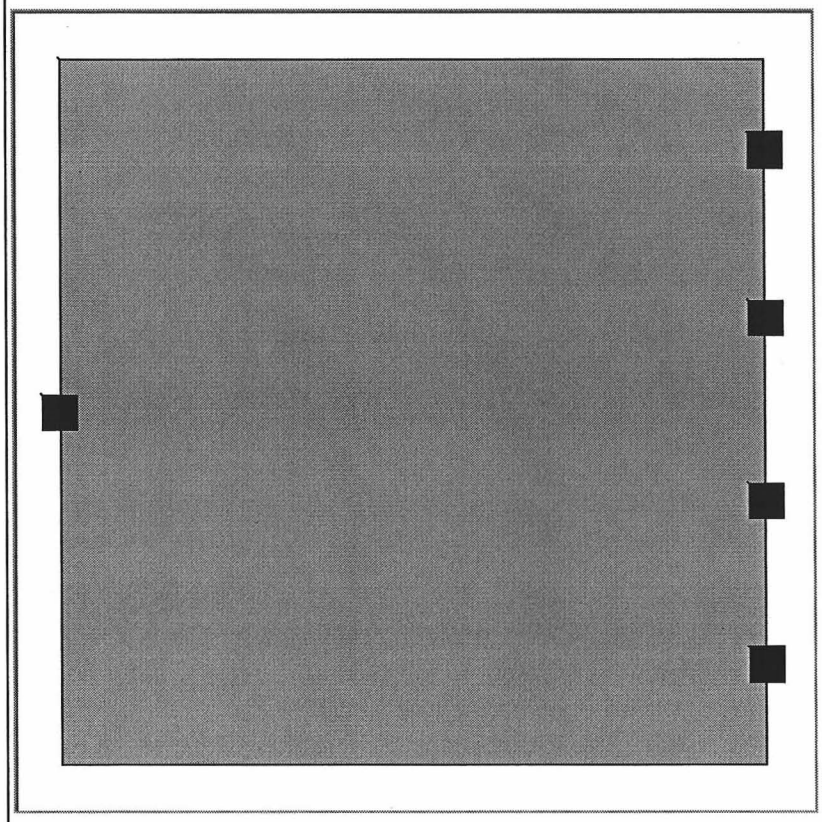
Root Locus Plot K from 0 to 100

Simulate for 10 s using input signal: Step

Note: The signal is only applied to the first input. Use Dynamic Systems for full simulation control.



Component Ports



Add Port | Delete Selected Port

Clear All Ports

Port Type: Signal Input

Port Name: uin

Port Components:

value	u(t)	Choose...
		Choose...
		Choose...
		Choose...
		Choose...

Component Generation

Generate MapleSim Comp

Source Details

```
model Furuta

  parameter Real J = 0.1000000000e-2 "J";
  parameter Real r = 0.3000000000e0 "r";
  parameter Real m = 0.5000000000e-1 "m";
  parameter Real c1 = 0.3000000000e-2 "c1";
  parameter Real c2 = 0.2000000000e-2 "c2";
  parameter Real c3 = 0.1000000000e-2 "c3";
  parameter Real c4 = 0.9800000000e-1 "c4";
  parameter Real a = 0.1000000000e-1 "a";
  parameter Real psi0 = 0.0e0 "psi0";
  parameter Real theta0 = 0.0e0 "theta0";

  Real psi (start = psi0, fixed=true);
  Real theta (start = theta0, fixed=true);
  Real vpsi (start = 0);
  Real vtheta (start = 0);
  Real apsi;
  Real atheta;
  Real u;

  annotation(
            Coordsys(
                     extent=[-100, -100; 100, 100],
                     grid=[2, 2],
```

Generate Component from Source

Figure 3.44 shows the MapleSim block diagram for the open loop simulation. The Furuta custom component generated with above Maple worksheet has been used as the Furuta model subsystem. The Probes have been used to obtain state transient responses. Probe is similar to Scope block of Simulink. The Constant block has been used to generate control signal 'u' that is zero in this particular case.

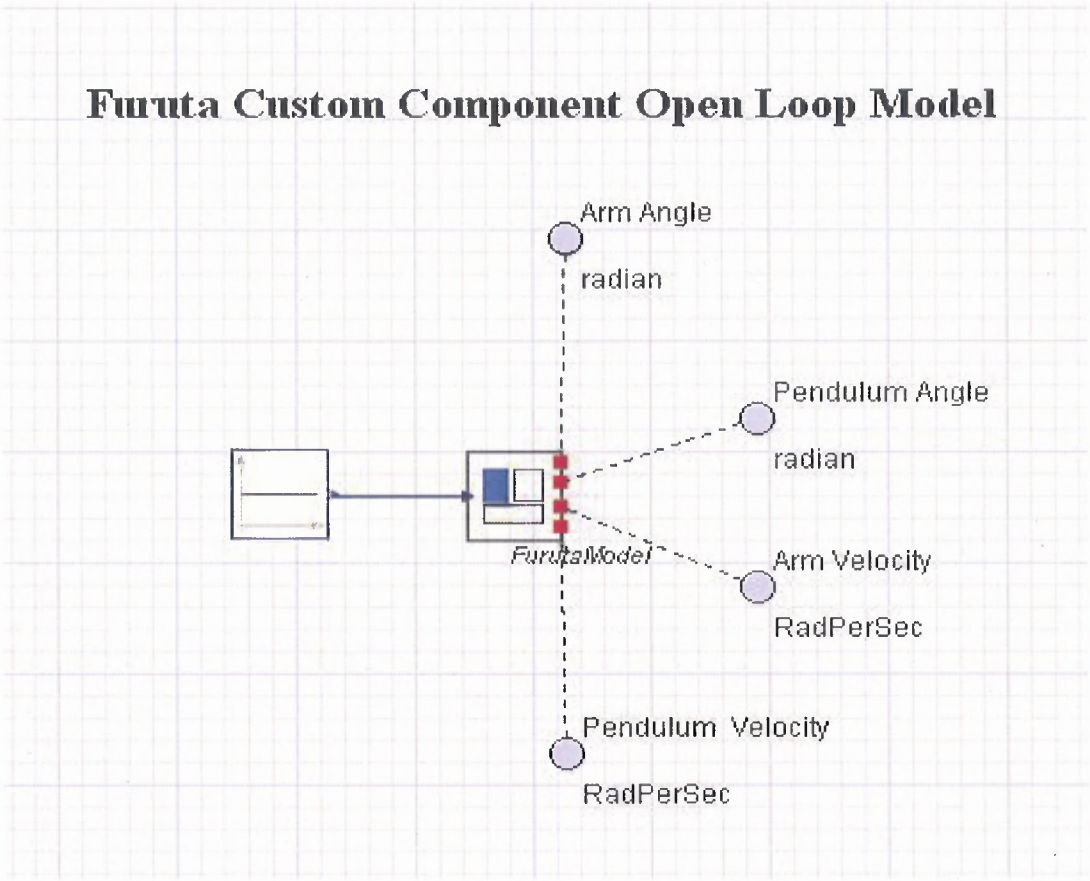


Figure 3.44 The Furuta Pendulum custom component for open loop model.

Linearization:

MapleSim provides pre-built Maple worksheet for linearizing the non linear Furuta Pendulum model to give state space model representation of the system. The linearization process for the Furuta Pendulum with Maple worksheet has been discussed below.

Maple Linearization Worksheet:

```
Description
Use this template to retrieve and linearize the MapleSim
subsystem.  The linearized model can be saved as a linear
systems object in the MapleSim document folder. This
linearized model can then be retrieved and used by other
templates.
```

Note: The ability to linearize the models is currently limited to continuous subsystems with defined signal input (*RealInput*) and signal output (*RealOutput*) ports.

Reset Template

Model Diagram

The Furuta Pendulum model block diagram has been imported to this section.

Model Summary

Use the navigation controls on the toolbar above the model diagram, select your subsystem and then click the System Update button to retrieve the subsystem. This may take some time. Specify the input ports and output ports that will be used to generate the linearized subsystem.

Note: To proceed, you must click the System Update button after selecting your subsystem.

System Update Model name: NonlinearFu

Specified Inputs and Outputs for the System

System Inputs:	System Outputs:	Unknown IOs:
`Main.NonlinearFurutaPendulum1.RI1`(t	`Main.NonlinearFurutaPendulum1.RO1`	none
	`Main.NonlinearFurutaPendulum1.RO2`	
	`Main.NonlinearFurutaPendulum1.RO3`	
	`Main.NonlinearFurutaPendulum1.RO4`	

Linearization

Select whether to calculate the trim point automatically or specify the trim point manually. The manual specification of the trim point is done into the Trim Point Specification subsection below.

○ Automatically Calculate the Trim Point	◉ Specify the Trim Point

Trim Point Specification

Specify the trim point

State Trim

Reset all to Zero

$[\grave{}Main.NonlinearFurutaPendulum1.DFPSubsys1inst.theta\_R1\grave{}\ (t) = 0,$
$\grave{}Main.NonlinearFurutaPendulum1.DFPSubsys1inst.theta\_R2\grave{}\ (t) = 0,$
$\grave{}Main.NonlinearFurutaPendulum1.DFPSubsys1inst.theta\_R1\_dot\grave{}\ (t) = 0,$
$\grave{}Main.NonlinearFurutaPendulum1.DFPSubsys1inst.theta\_R2\_dot\grave{}\ (t) = 0]$

Variable	Value
`Main.NonlinearFurutaPendulum1.DFPSubsys1inst.theta	0

Input Trim

Reset all to Zero

$[\grave{}Main.NonlinearFurutaPendulum1.RI1\grave{}\ (t) = 0]$

Variable	Value
`Main.NonlinearFurutaPendulum1.RI1`(t)	0

Linearize / Generate State Space

Variable Map

This describes the mapping of the variables in the original system to the variables of the linearized model.

☑ State Mapping ☑ Input Mapping ☑ Output Mapping

Linearized Model

This describes the state-space form of the linearized model.

Matrix A | Matrix B | Matrix C | Matrix D | All Matrices

$$\left[\begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & -10. & 147.1500000 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & -15. & 269.7750000 & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 1000. \\ 0. \\ 1500. \end{bmatrix}, \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 1. & 0. & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}\right]$$

Save the Model

In order to save the corresponding linear system object to the document folder, enter the name of the model and the description of the system and then click the Save button.

Name

LinearizedFurutaPendulum

Description

```
Model  :NonlinearFurutaPendulum1

States :
  x[1](t)
  x[2](t)
  x[3](t)
  x[4](t)

Inputs :
  `Main.NonlinearFurutaPendulum1.RI1`(t)
```

Save

3.4.2 Full State Feedback design by Pole Placement:

The Section 3.4.2 presents the full state feedback control design with Pole Placement for the Furuta Pendulum system using Maple worksheet. The Control Design tool box of Maple has been used for Pole Placement design. It also shows the simulation results obtained with MapleSim for Pole Placement control design. The Maple work sheet to calculate the full state feedback gain using Pole Placement design has been discussed below.

Maple Linear Systems Worksheet [15]:

Description

This template allows you to retrieve the linear systems object, stored as an .msys file, from the MapleSim document folder. Then you can construct your own design or analysis document using the power of Maple technical document.

Reset Template

Model Diagram

The Furuta Pendulum model block diagram has been imported to this section.

Model Input

Select a linear system object from the list.

LinearizedFurutaPendulum.msys

Name

LinearizedFurutaPendulum

Description

```
Model  :NonlinearFurutaPendulum1

States :
  x[1](t)
  x[2](t)
  x[3](t)
  x[4](t)

Inputs :
  `Main.NonlinearFurutaPendulum1.RI1`(t)
```

These are the state-space matrices that correspond to the linear system object selected above.

Matrix A Matrix B Matrix C Matrix D All Matrices

$$\left[\begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & -10. & 147.1500000 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & -15. & 269.7750000 & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 1000. \\ 0. \\ 1500. \end{bmatrix}, \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 1. & 0. & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}\right]$$

Enter the name of the desired Dynamic System object below, and then click the button to create the object.

Assign to Variable: sys

Design and Analysis

with(*DynamicSystems*)

[*AlgEquation*, *BodePlot*, *CharacteristicPolynomial*, *Chirp*, *Coefficients*, *ControllabilityMatrix*, *Controllable*, *DiffEquation*, *DiscretePlot*, *FrequencyResponse*, *GainMargin*, *Grammians*, *ImpulseResponse*, *ImpulseResponsePlot*, *IsSystem*, *MagnitudePlot*, *NewSystem*, *ObservabilityMatrix*, *Observable*, *PhaseMargin*, *PhasePlot*, *PrintSystem*, *Ramp*, *ResponsePlot*, *RootContourPlot*, *RootLocusPlot*, *RouthTable*, *SSModelReduction*, *SSTransformation*, *Simulate*, *Sinc*, *Sine*, *Square*, *StateSpace*, *Step*, *StepProperties*, *System*, *SystemOptions*, *ToDiscrete*, *TransferFunction*, *Triangle*, *Verify*, *ZeroPoleGain*, *ZeroPolePlot*]

```
with(ControlDesign)
[Characterize, CohenCoon, DominantPole, FeasibleGains,
    GainPhaseMargin, Kalman, LQR, LQRContinuous,
    LQRDiscrete, LQROutput, ParameterIdentify, RegionPoles,
    StateFeedback, StateObserver, ZNFreq, ZNTimeModified]

DesiredPoles := [-10 + 10·I, -10 - 10·I, -1, -5.56];
[-10 + 10 I, -10 - 10 I, -1, -5.56]

Kmat := StateFeedback[PolePlacement](sys=-a, sys=-b,
    DesiredPoles);
[-0.022670744138634221,-0.039015290519877898
    0.41947049609242331,0.037050193679918574]
```

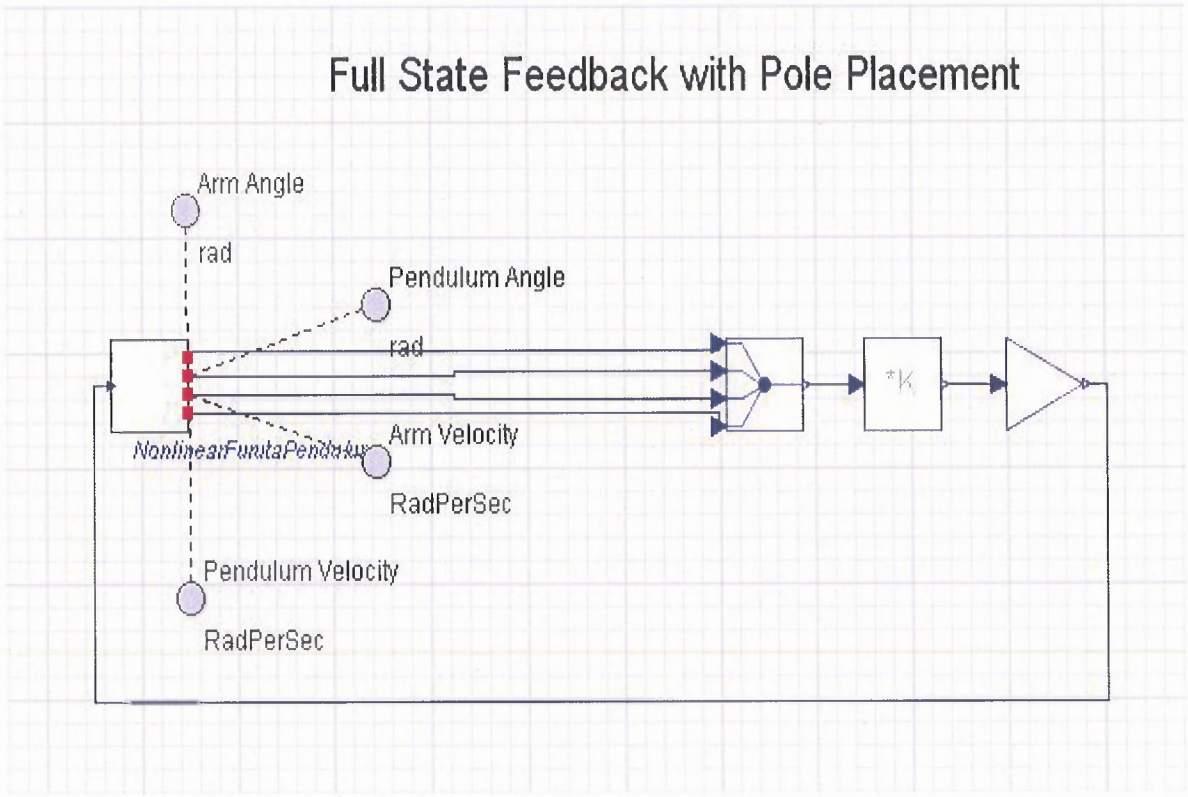


Figure 3.45 MapleSim block diagram for full state feedback with Pole Placement.

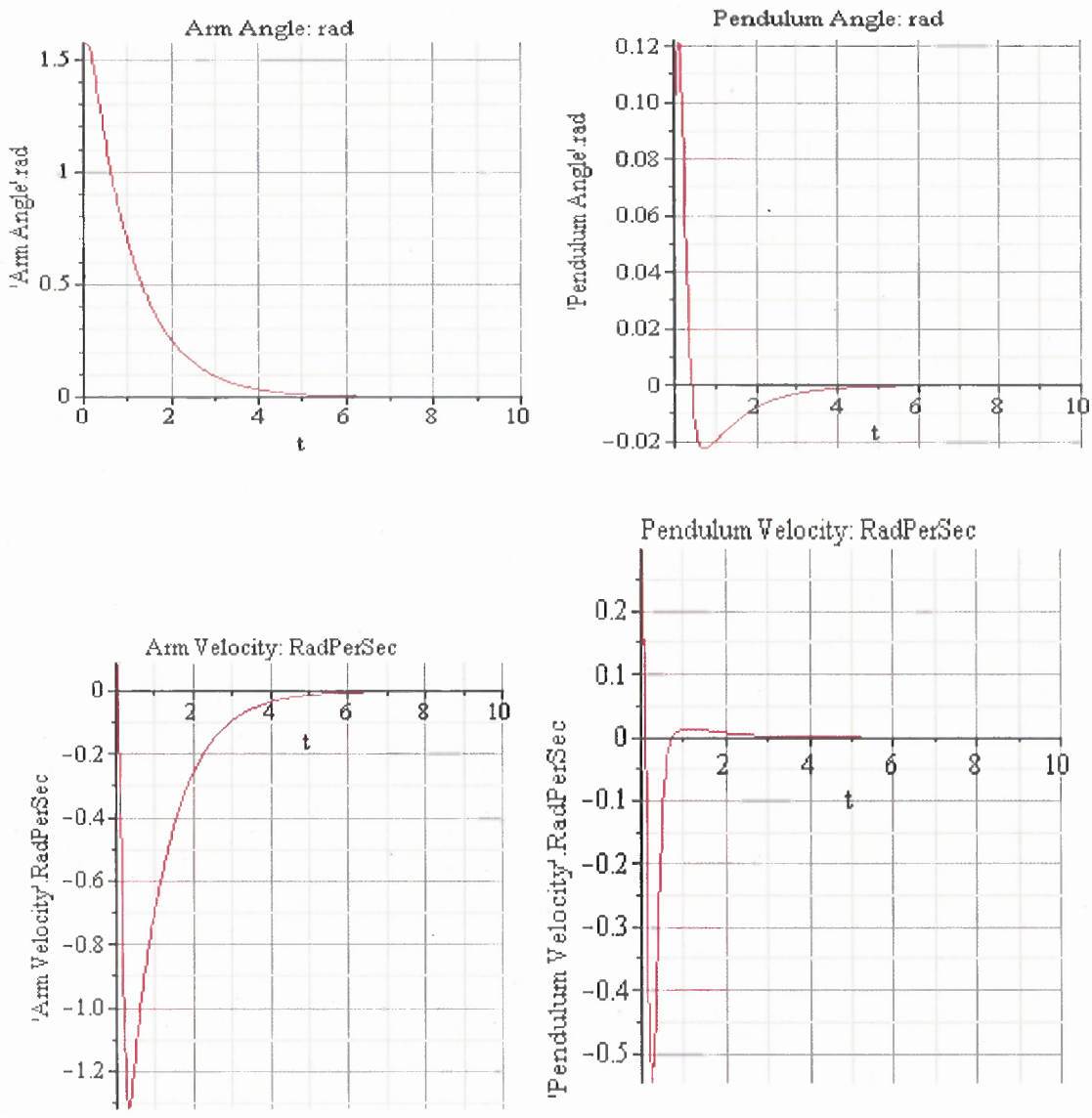


Figure 3.46 MapleSim probe outputs for the Pole Placement design.

Figure 3.45 shows MapleSim block diagram for the Pole Placement design simulation. It consists of the Matrix Gain block which contains the gain vector calculated above using Maple worksheet. It is followed by Gain block which contains gain of '-1' to provide negative feedback. Figure 3.46 shows probe outputs for the position and velocity transient responses. These transients are exactly same as obtained with Simulink.

3.4.3 Full State Feedback design by LQR

The Section 3.4.3 presents the full state feedback control design with LQR for the Furuta Pendulum system using Maple worksheet. The Control Design tool box of Maple has been used for the LQR design. It also shows the simulation results obtained with MapleSim for the LQR control design. The Maple work sheet to calculate the full state feedback gain using LQR design has been discussed below.

Maple Linear System Worksheet [15]:

Description

This template allows you to retrieve the linear systems object, stored as an .msys file, from the MapleSim document folder. Then you can construct your own design or analysis document using the power of Maple technical document.

Reset Template

Model Diagram

The Furuta Pendulum model block diagram has been imported to this section.

Model Input

Select a linear system object from the list.

LinearizedFuruta.msys

Name

LinearizedFuruta

Description

```
Model  :NonlinearFurutaPendulum1

States :
  x[1](t)
  x[2](t)
  x[3](t)
  x[4](t)

Inputs :
  `Main.NonlinearFurutaPendulum1.RI1`(t)
```

These are the state-space matrices that correspond to the linear system object selected above.

Matrix A Matrix B Matrix C Matrix D All Matrices

$$\left[\begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & K\ 10. & 147.1500000 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & K\ 15. & 269.7750000 & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 1000. \\ 0. \\ 1500. \end{bmatrix}, \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 1. & 0. & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}\right]$$

Enter the name of the desired Dynamic System object below, then click the button to create the object.

Assign to Variable: sys

Design and Analysis

with(*DynamicSystems*)

[*AlgEquation*, *BodePlot*, *CharacteristicPolynomial*, *Chirp*,
Coefficients, *ControllabilityMatrix*, *Controllable*, *DiffEquation*,
DiscretePlot, *FrequencyResponse*, *GainMargin*, *Grammians*,
ImpulseResponse, *ImpulseResponsePlot*, *IsSystem*,
MagnitudePlot, *NewSystem*, *ObservabilityMatrix*, *Observable*,
PhaseMargin, *PhasePlot*, *PrintSystem*, *Ramp*, *ResponsePlot*,
RootContourPlot, *RootLocusPlot*, *RouthTable*,
SSModelReduction, *SSTransformation*, *Simulate*, *Sinc*, *Sine*,
Square, *StateSpace*, *Step*, *StepProperties*, *System*,
SystemOptions, *ToDiscrete*, *TransferFunction*, *Triangle*, *Verify*,
ZeroPoleGain, *ZeroPolePlot*]

$DynamicSystems[PrintSystem](sys)$

State Space

continuous

4 output(s); 1 input(s); 4 state(s)

$$\text{inputvariable} = [u_1(t)]$$

$$\text{outputvariable} = [y_1(t), y_2(t), y_3(t), y_4(t)]$$

$$\text{statevariable} = [x_1(t), x_2(t), x_3(t), x_4(t)]$$

$$a = \begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & -10. & 147.1500000 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & -15. & 269.7750000 & 0. \end{bmatrix}$$

$$b = \begin{bmatrix} 0. \\ 1000. \\ 0. \\ 1500. \end{bmatrix}$$

$$c = \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

$$d = \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}$$

State Mapping/Transformation

The output vector of the original nonlinear system is defined as:

$$vecOutput := \langle \text{Psi}(t), \text{theta}(t), PsiRate(t), thetaRate(t) \rangle$$

$$\begin{bmatrix} \Psi(t) \\ \theta(t) \\ PsiRate(t) \\ thetaRate(t) \end{bmatrix}$$

Given the output matrix C from the linear system, we know that:

$$\langle x1(t), x2(t), x3(t), x4(t)\rangle = sys\text{:-}c^{\%T} \;.\; vecOutput$$

$$\begin{bmatrix} x1(t) \\ x2(t) \\ x3(t) \\ x4(t) \end{bmatrix} = \begin{bmatrix} 1.\,\Psi(t) \\ 1.\,PsiRate(t) \\ 1.\,\theta(t) \\ 1.\,thetaRate(t) \end{bmatrix}$$

So we can use this relationship to define the appropriate Q and R matrices, as well as do the inverse mapping for the gain block.

As an example, applying the similarity transform:

$$matTinv := sys\text{:-}c : matT := sys\text{:-}c^{\%T} :$$

$$newA := matTinv \;.\; sys\text{:-}a \;.\; matT;$$

$$\begin{bmatrix} 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 147.150000000000006 & -10. & 0. \\ 0. & 269.774999999999977 & -15. & 0. \end{bmatrix}$$

$$newB := matTinv \;.\; sys\text{:-}b;$$

$$\begin{bmatrix} 0. \\ 0. \\ 1000. \\ 1500. \end{bmatrix}$$

$$newC := sys\text{:-}c \;.\; matT$$

$$\begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

We get the same state space system as MATLAB.

LQR Design

The Q and R matrices for the LQR controller are defined as follows:

$$Qmat := \begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 100000 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 1000 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 \\ 0 & 0 & 100000 & 0 \\ 0 & 0 & 0 & 0 \end{bmatrix}$$

$Rmat := [1]$

$[1]$

The gain value, K for the LQR controller is defined as follows

$K := ControlDesign[LQR](sys, Qmat, Rmat)$

$\{-31.6227763253264129, -25.6434845313779078, 339.909137435868615, 17.7410230347595643\}$

The C matrix of the linear model defined in variable $sys:-c$

$sys:-c$

$$\begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 1. & 0. & 0. \\ 0. & 0. & 0. & 1. \end{bmatrix}$$

The K matrix after the re-mapping process is shown below.

$Kctrl := K.sys:-c$

$\{-31.6227763253264129, 339.909137435868615, -25.6434845313779078, 17.7410230347595643\}$

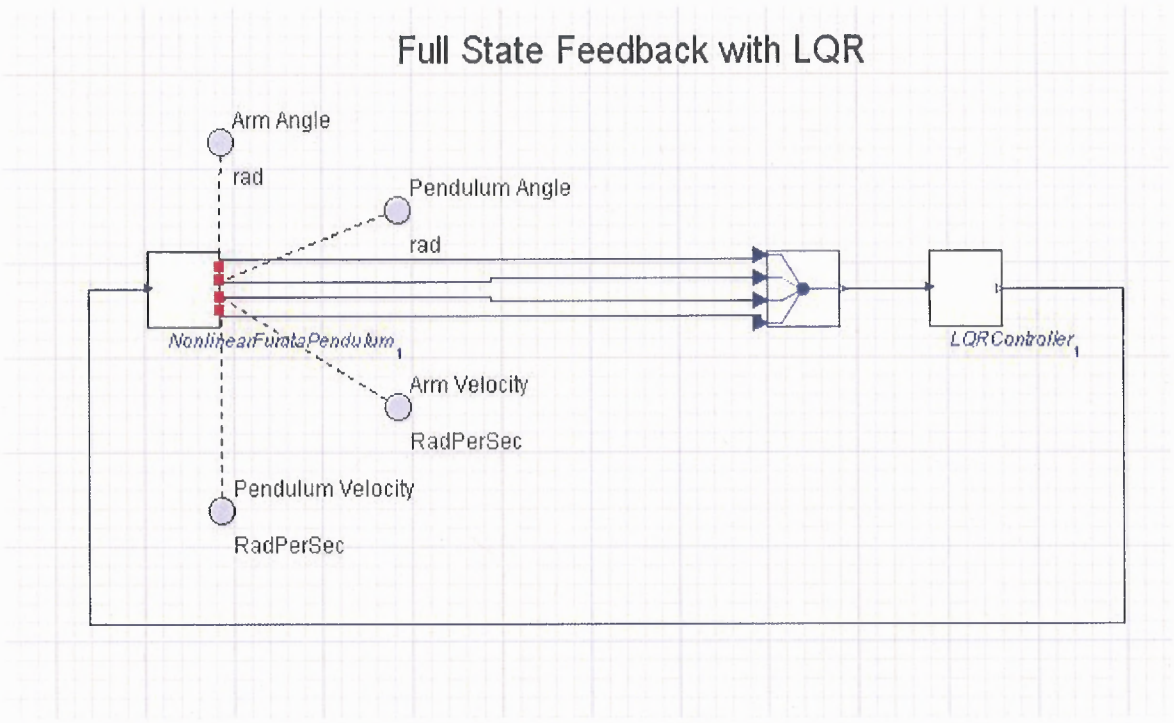


Figure 3.47 MapleSim block diagram for full state feedback with LQR.

Figure 3.47 shows the MapleSim block diagram for the LQR design simulation. It consists of the Matrix Gain block which contains the gain vector calculated above using Maple worksheet. It is followed by Gain block which contains gain of '-1' to provide negative feedback. The LQR Controller subsystem contains both of these gain blocks. Figure 3.48 shows the probe outputs for the position and velocity transient responses. These transients are exactly same as obtained with Simulink.

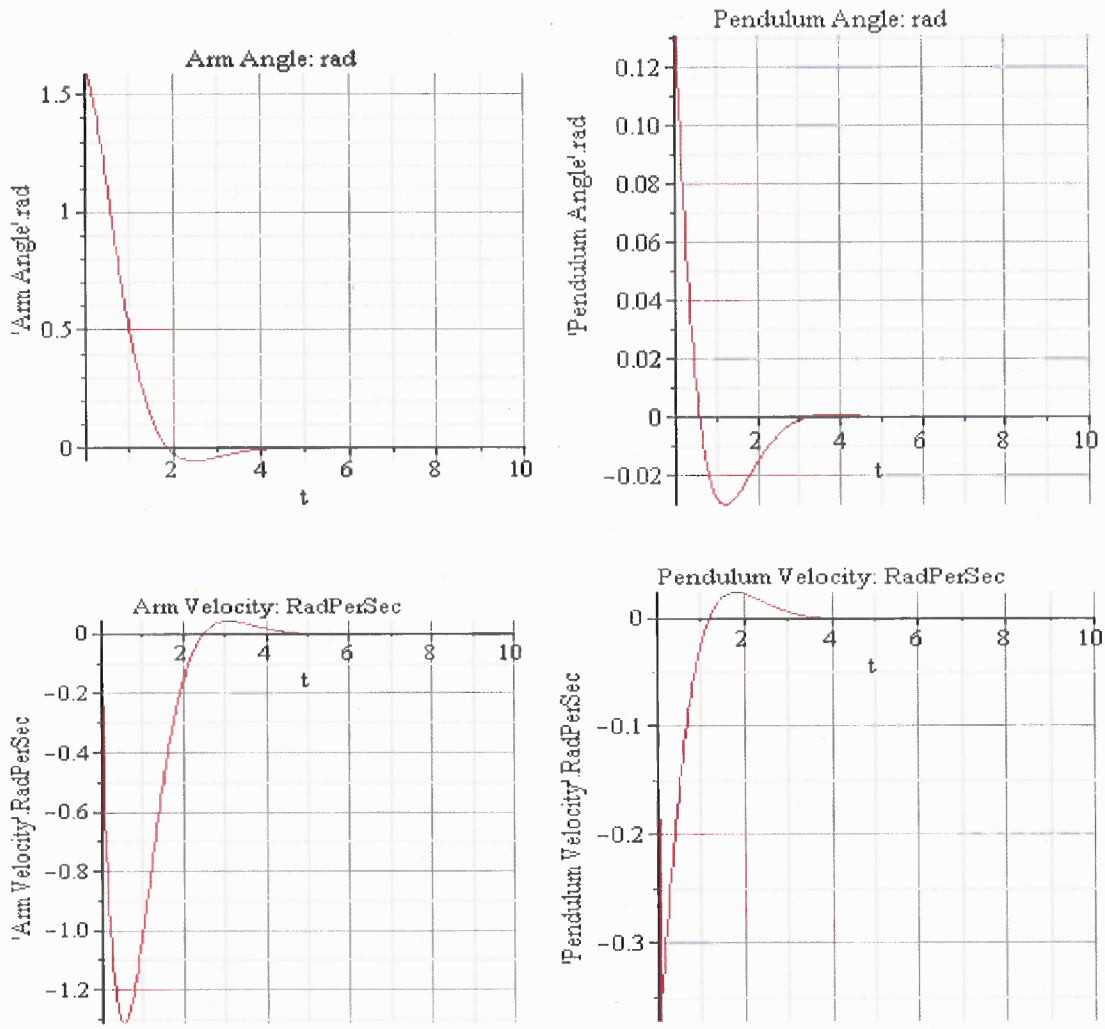


Figure 3.48 MapleSim probe outputs for the LQR design.

3.4.4 Full Order Observer Design using Kalman Filter

The Section 3.4.4 presents the full order observer design with Kalman Filter for the Furuta Pendulum system using Maple worksheet. The Control Design tool box of Maple has been used for the Kalman Observer design. It also shows the simulation results obtained with MapleSim for the Kalman Observer design. The Maple worksheet to calculate the Kalman observer gain has been discussed below.

Maple Linear System Worksheet [15]:

Description

This template allows you to retrieve the linear systems object, stored as an .msys file, from the MapleSim document folder. Then you can construct your own design or analysis document using the power of Maple technical document.

Reset Template

Model Diagram

The Furuta Pendulum model block diagram has been imported to this section.

Model Input

Select a linear system object from the list.

LinearizedFuruta.msys

Name

LinearizedFuruta

Description

```
Model  :NonlinearFurutaPendulum1

States :
  x[1](t)
  x[2](t)
  x[3](t)
  x[4](t)

Inputs :
  `Main.NonlinearFurutaPendulum1.RI1`(t)
```

These are the state-space matrices that correspond to the linear system object selected above.

Matrix A | Matrix B | Matrix C | Matrix D | All Matrices

$$\left[\begin{bmatrix} 0. & 1. & 0. & 0. \\ 0. & K\ 10. & 147.1500000 & 0. \\ 0. & 0. & 0. & 1. \\ 0. & K\ 15. & 269.7750000 & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 1000. \\ 0. \\ 1500. \end{bmatrix}, \begin{bmatrix} 1. & 0. & 0. & 0. \\ 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 1. & 0. & 0. \end{bmatrix}, \begin{bmatrix} 0. \\ 0. \\ 0. \\ 0. \end{bmatrix}\right]$$

Enter the name of the desired Dynamic System object below, then click the button to create the object.

Assign to Variable: sys

Design and Analysis

with(*DynamicSystems*)
[*AlgEquation*, *BodePlot*, *CharacteristicPolynomial*, *Chirp*,
Coefficients, *ControllabilityMatrix*, *Controllable*, *DiffEquation*,
DiscretePlot, *FrequencyResponse*, *GainMargin*, *Grammians*,
ImpulseResponse, *ImpulseResponsePlot*, *IsSystem*,
MagnitudePlot, *NewSystem*, *ObservabilityMatrix*, *Observable*,
PhaseMargin, *PhasePlot*, *PrintSystem*, *Ramp*, *ResponsePlot*,
RootContourPlot, *RootLocusPlot*, *RouthTable*,
SSModelReduction, *SSTransformation*, *Simulate*, *Sinc*, *Sine*,
Square, *StateSpace*, *Step*, *StepProperties*, *System*,
SystemOptions, *ToDiscrete*, *TransferFunction*, *Triangle*, *Verify*,
ZeroPoleGain, *ZeroPolePlot*]

with(*ControlDesign*);
[*Characterize*, *CohenCoon*, *DominantPole*, *FeasibleGains*,
GainPhaseMargin, *Kalman*, *LQR*, *LQRContinuous*,
LQRDiscrete, *LQROutput*, *ParameterIdentify*, *RegionPoles*,
StateFeedback, *StateObserver*, *ZNFreq*, *ZNTimeModified*]

Designing the Kalman filter

Formulating the new system matrices
Coordinate transformation matrix:

$matTinv := sys:\text{-}c : matT := sys:\text{-}c^{\%T}$:

New system A matrix with ordering corrected

$newA := matTinv \,.\, sys:\text{-}a \,.\, matT;$

$$\begin{bmatrix} 0. & 0. & 1. & 0. \\ 0. & 0. & 0. & 1. \\ 0. & 147.150000000000006 & -10. & 0. \\ 0. & 269.774999999999977 & -15. & 0. \end{bmatrix}$$

New input B matrix:

$newB := matTinv \,.\, sys:\text{-}b;$

$$\begin{bmatrix} 0. \\ 0. \\ 1000. \\ 1500. \end{bmatrix}$$

Formulate the measurement C matrix

$newC := \langle\langle 1|0|0|0\rangle, \langle 0|1|0|0\rangle\rangle;$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

Formulating the new system

$newSys := StateSpace(\, newA, newB, newC \,);$

State Space

continuous

2 output(s); 1 input(s); 4 state(s)

inputvariable = $[u1(t)]$

outputvariable = $[y1(t), y2(t)]$

statevariable = $[x1(t), x2(t), x3(t), x4(t)]$

Constructing the covariance matrices

$matQ := 0.02 \cdot LinearAlgebra:\text{-}IdentityMatrix(1, 1)$:

$matR := LinearAlgebra:\text{-}IdentityMatrix(2, 2)$:

Calling Kalman filter design command

$K := Kalman(\, newSys, newB, LinearAlgebra:\text{-}ZeroMatrix(2, 1),$
$matQ, matR);$

$$\begin{bmatrix} 8.934643507788997858.84931841426397092 \\ 8.849318414263970925.2516072797180193 \\ 79.069145506130254873.936105904655278 \\ 128.588912711531520357.977053312191117 \end{bmatrix}, \begin{bmatrix} [8.934643507788997858.84931841426397092 \\ 79.069145506130254128.588912711531520], \\ [8.849318414263970925.2516072797180193 \\ 173.936105904655278357.977053312191117], \\ [79.069145506130254173.936105904655278 \\ 1734.184862914008143115.46133137450806], \\ [128.588912711531520357.977053312191117 \\ 3115.461331374508065974.20943343537419]] \end{bmatrix}$$

Constructing the state space observer from the observer gain:

Kfilter := *StateObserver*[*Observer*](*newSys*, *K*[1]);

State Space

continuous

6 output(s); 3 input(s); 4 state(s)

inputvariable = $[u1(t), u2(t), u3(t)]$

outputvariable = $[y1(t), y2(t), y3(t), y4(t), y5(t), y6(t)]$

statevariable = $[x1(t), x2(t), x3(t), x4(t)]$

Kfilter:-*a*; *Kfilter*:-*b*; *Kfilter*:-*c*; *Kfilter*:-*d*;

$$\begin{bmatrix} -8.93464350778899785 & -8.84931841426397092 & 1. & 0. \\ -8.84931841426397092 & -25.2516072797180193 & 0. & 1. \\ -79.0691455061302548 & -26.7861059046552725 & -10. & 0. \\ -128.588912711531520 & -88.2020533121911399 & -15. & 0. \end{bmatrix}$$

$$\begin{bmatrix} 0. & 8.934643507788997858.84931841426397092 \\ 0. & 8.849318414263970925.2516072797180193 \\ 1000. & 79.069145506130254873.936105904655278 \\ 1500. & 128.588912711531520357.977053312191117 \end{bmatrix}$$

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix}$$

$$\begin{bmatrix} 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix}$$

Estimator pole location:

LinearAlgebra:-*Eigenvalues*(*Kfilter*:-*a*)

$$\begin{bmatrix} -3.59398275262242839 + 2.39549761273298323\,I \\ -3.59398275262242839 - 2.39549761273298323\,I \\ -18.4991426411310869 + 5.44257123807761367\,I \\ -18.4991426411310869 - 5.44257123807761367\,I \end{bmatrix}$$

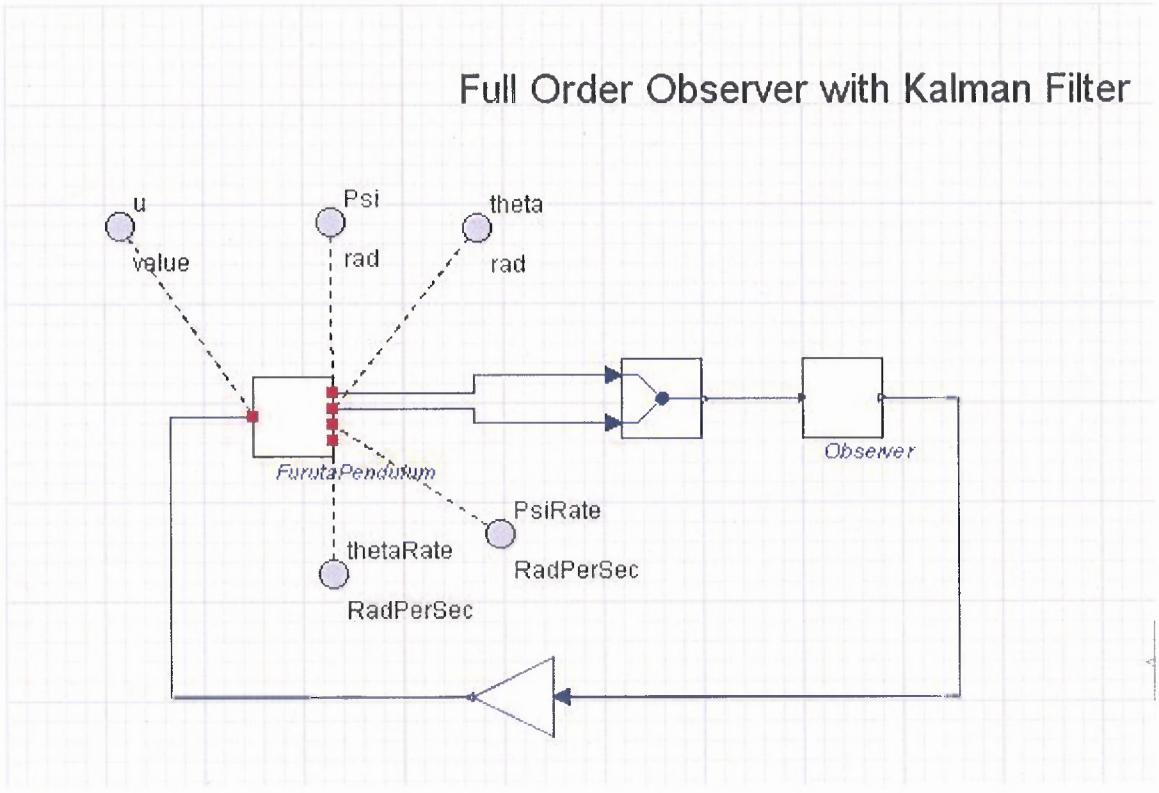


Figure 3.49 MapleSim block diagram for full order observer using Kalman Filter.

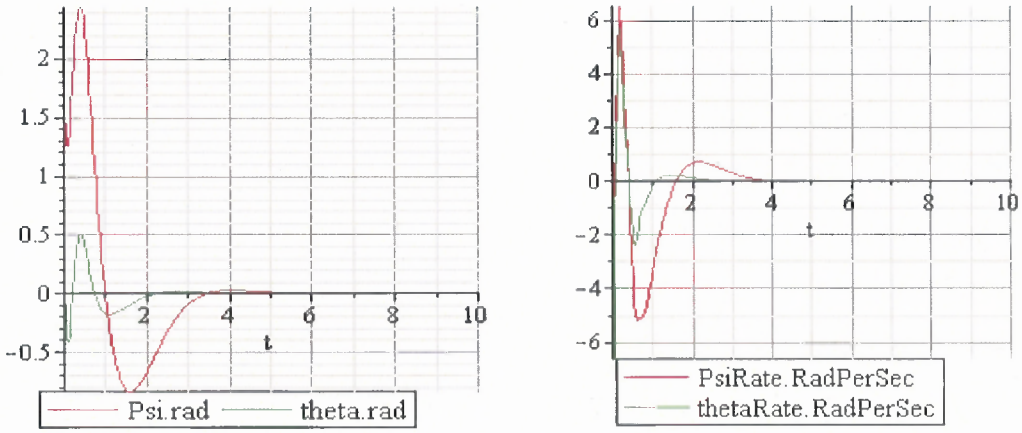


Figure 3.50 MapleSim probe outputs for the Kalman Observer design.

Figure 3.49 shows the MapleSim block diagram for the Kalman Observer design simulation. It consists of the State space model block which contains the observer state space model obtained using above Maple worksheet. It is followed by Gain block which contains gain of '-1' to provide negative feedback. The Observer subsystem contains the Observer dynamics. Figure 3.50 shows the probe outputs for the position and velocity transient responses. These transients are exactly same as obtained with Simulink.

3.5 VisSim

VisSim is a block diagram language for creating complex nonlinear dynamic systems from Visual Solutions. VisSim has highly tuned math engine that executes dynamic model directly with no compilation delay. In addition to accelerating development with rapid turnaround for changes, VisSim's fast execution speed is perfect for model based operator training, off-line controller tuning and hardware-in-the-loop testing. Its efficient C code generator makes it an ideal platform for model-based embedded system development.

3.5.1 Open Loop Analysis

The Section 3.5.1 presents modeling of the Furuta Pendulum with VisSim for the open loop analysis. By combining the simplicity and clarity of a block diagram interface with a high-performance mathematical engine, VisSim provides fast and accurate solutions for linear, nonlinear, continuous time, discrete time, SISO, MIMO, multi-rate, and hybrid systems. With VisSim's wide selection of block operations and expression handling, complex systems can be quickly entered into VisSim.

Figure 3.5.1 shows the non linear Furuta Pendulum model which has been built using various VisSim blocks. VisSim doesn't support implementation of mathematical functions like MATLAB Fcn block. It has expression block that allows entering a C expression or matrix data that VisSim parses and acts upon. But for non linear dynamic systems like Furuta Pendulum having complicated equations of motion it is not possible to implement it with expression block. Hence, various VisSim arithmetic blocks have been used to implement the non-linear Furuta pendulum model. [17]

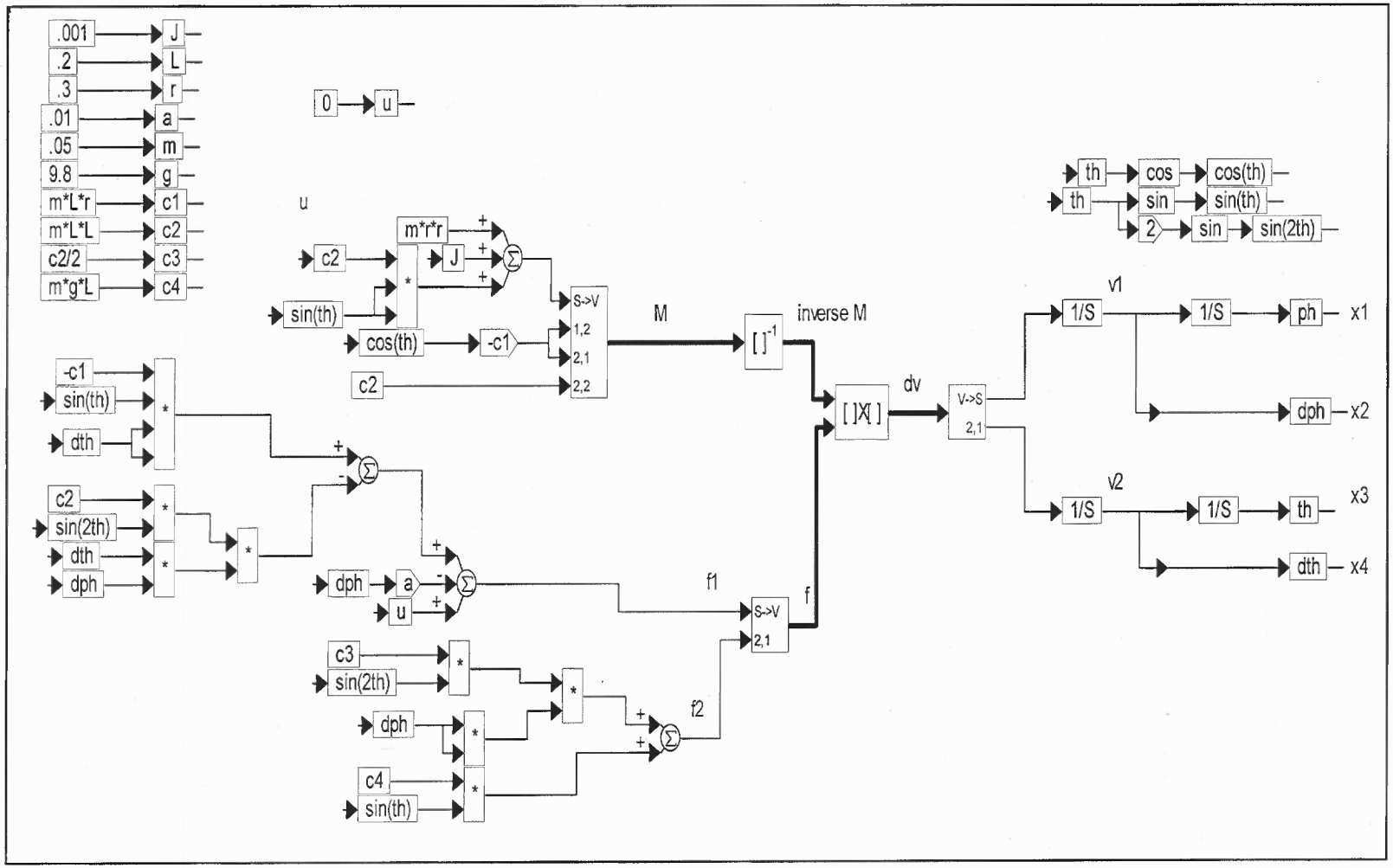


Figure 3.50 VisSim open loop block diagram for the Furuta pendulum.

The variable blocks have been used to define various system parameters within VisSim work space. The variable block lets defining and transmitting a signal throughout diagram without the use of wires. The variables of same name share same signal throughout VisSim diagram. It also accepts arithmetic and trigonometric functions. The * block produces the product of the input signals. Inputs can be scalars or vectors. The const block generates a constant signal. The const block accepts alphanumeric text strings and matrix data. The `scalarToVec` block reduces wiring clutter by combining input signals into a single vector wire. This has been used usually as prerequisite for performing vector and matrix algebra. Similarly, the `vecToScalar` block separates a single vector wire into individual output signals. The invert block inverts a square matrix using singular value decomposition. The invert block accepts one vector input and produces one vector output. The multiply block performs a matrix multiplication. The multiply block accepts two vector inputs and produces one vector output. The integrator block performs numerical integration on the input signal using the integration algorithm. It doesn't support vector input hence four different integrators have been used in cascaded form to extract position and velocity components. The slider block allows mouse input to dynamically modify a signal value during a simulation, between a lower and upper bound in 1% and 10% increments. The slider block displays the current value applied to the signal. It is very useful for varying different system parameters during simulation. As shown in Figure 3.50 these various blocks have been connected logically as per equations of motion to give position and velocity states of the Furuta pendulum.[17]

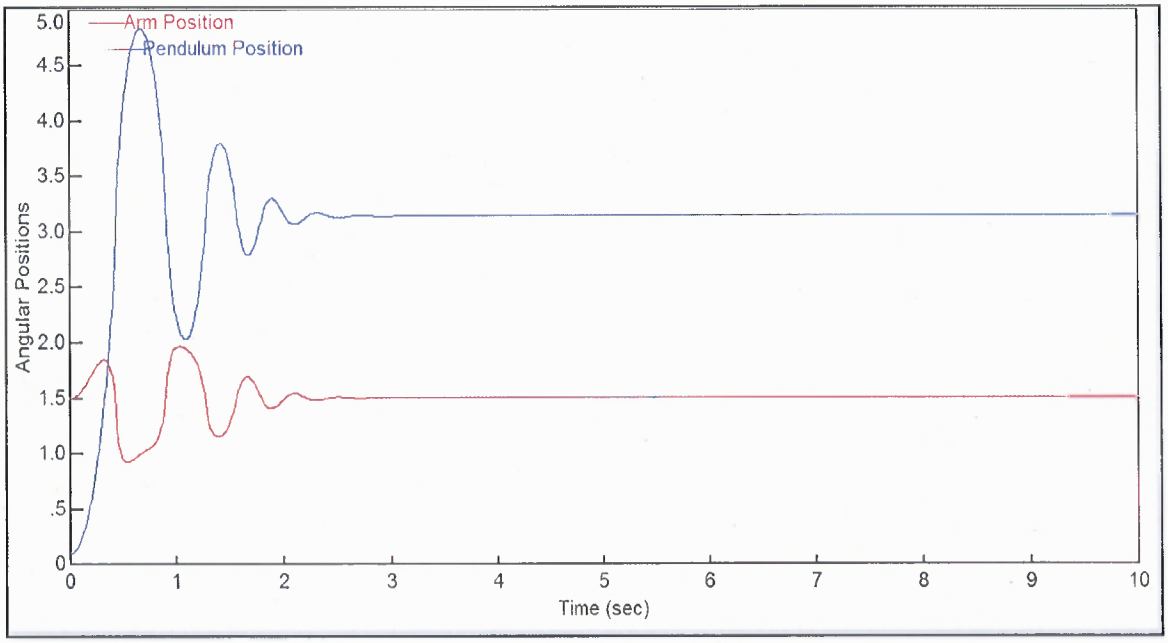


Figure 3.51 VisSim open loop output for the Arm angle and the Pendulum angle with initial conditions [1.57, 0.1] on angles.

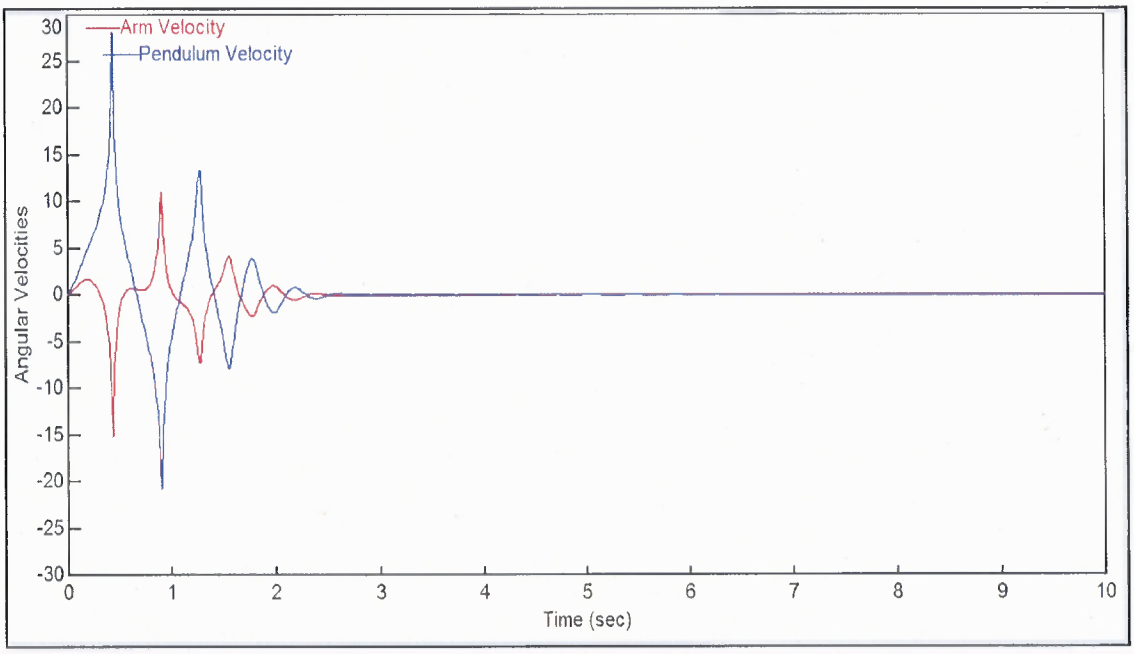


Figure 3.52 VisSim open loop output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0.1] on angles.

Figure 3.51 shows the transient responses for positions of the arm and the pendulum. As per initial condition the arm position takes off from 1.57 radians and after few oscillations it settles down to its initial position. The pendulum rod starts with 0.1 radians as per given initial condition and it moves through angle of 3.14 radians and it settles there after tumbling down from initial position. This position behavior of the Furuta pendulum is exactly same as obtained in Simulink open loop simulation.

Figure 3.52 shows transient responses for the arm and the pendulum velocities which have been observed after running simulation for 10 seconds. The velocity behavior of the Furuta pendulum in VisSim is exactly same as obtained with Simulink open-loop simulation. Both position and velocity open loop behaviors of the Furuta system in VisSim are exactly same as Simulink open loop outputs.

3.5.2 Full State Feedback design by Pole Placement:

The Section 3.5.2 presents Full State Feedback control design with Pole Placement for the Furuta Pendulum system. VisSim doesn't have dedicated control tool box for designing state space control algorithms thus the Control System toolbox of MATLAB has been used for pole placement design and then VisSim engine has been used for simulations. This Section also shows simulation results obtained with VisSim for Pole Placement control design.

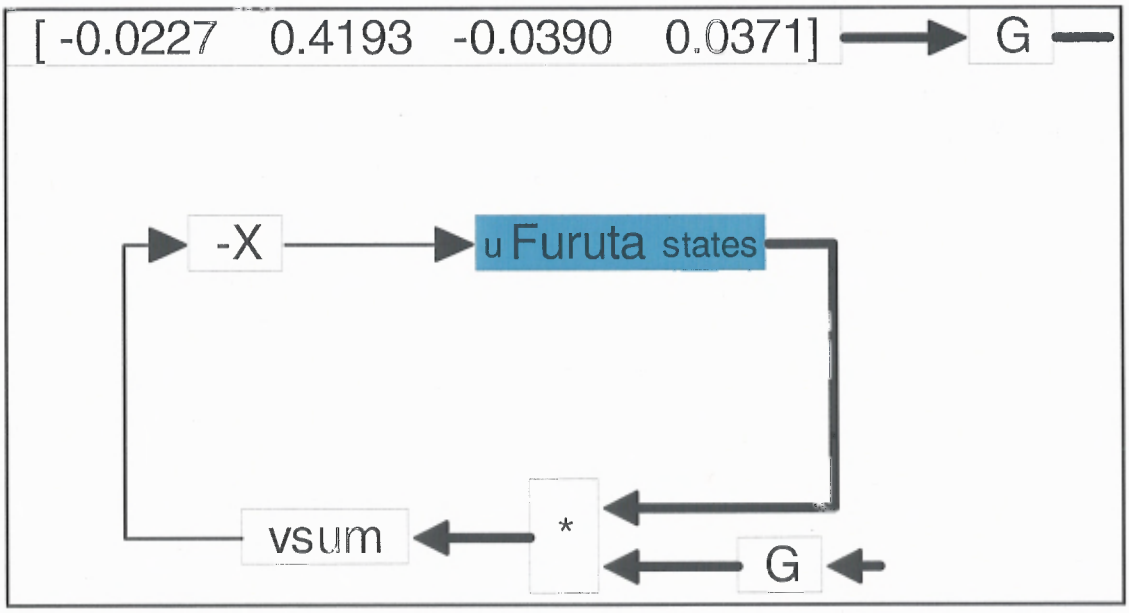


Figure 3.53 VisSim block diagram for full state feedback design by Pole Placement.

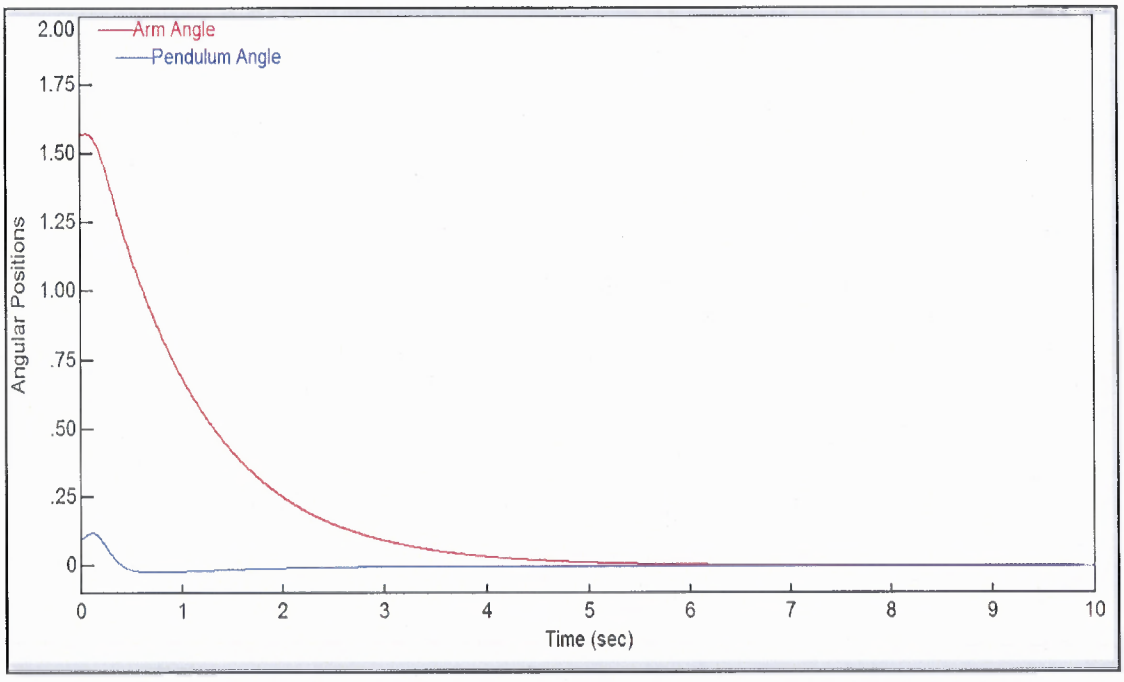


Figure 3.54 VisSim Pole Placement output for the arm angle and the pendulum angle with initial conditions [1.57, 0] on angles.

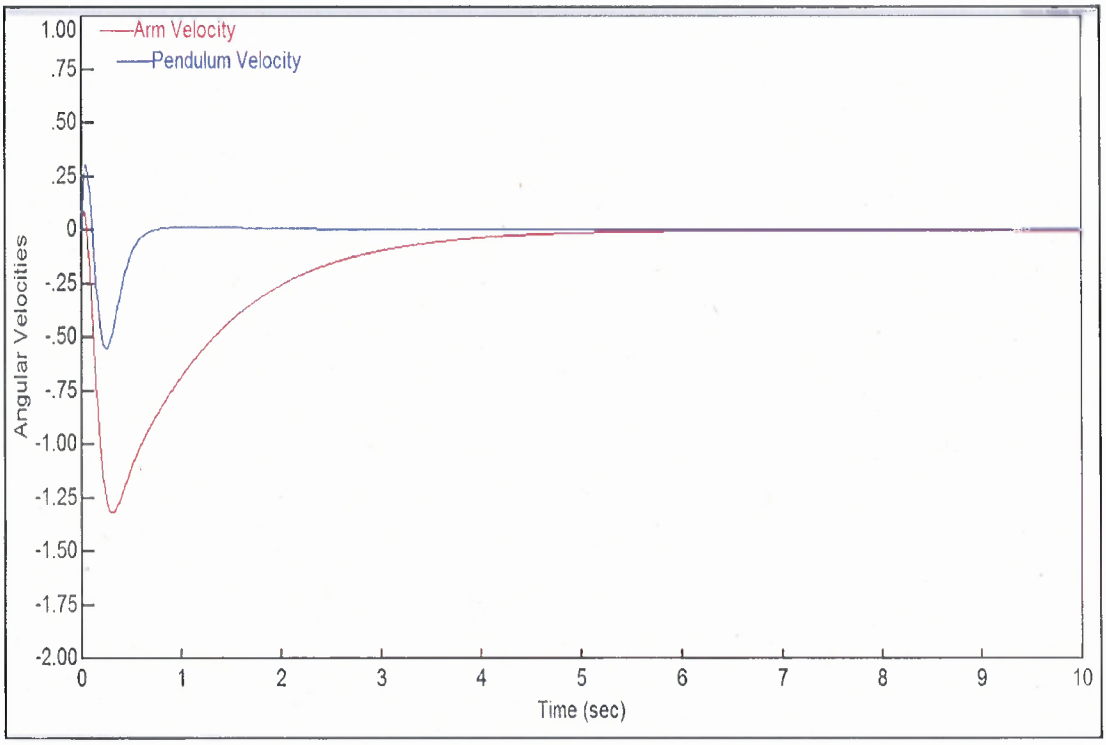


Figure 3.55 VisSim Pole Placement output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0] on angles.

Figure 3.53 shows the block diagram of Pole Placement control for the Furuta Pendulum system. It is similar to the Simulink Pole Placement block diagram. The Gain block multiplies the input state vector by matrix gain. After remapping of gain matrix, G = [-0.0227 0.4193 -0.0390 0.0371] has been used to implement state feedback control law u= -Gx. The vsum block produces a single value summation of all the elements in the matrix. The vsum block accepts one vector input and produces one scalar output. Further, the -X block negates the input signal. Input can be scalar, vector, or matrix. Figure 3.54 shows transient responses of the arm position and the pendulum position. The arm position starts with initial 1.57 radians and it settles down to 0 after smooth transition. Also, the pendulum rod starts with 0.1 radian position and it settles down to 0. It validates the fact that the inverted pendulum has been balanced on the arm. The

position behavior of the Scicos transients is exactly same as that of Simulink. Figure 3.55 shows velocity transients of the Furuta Pendulum system for the designed Pole Placement control. The arm and the pendulum velocities settle to 0 in accordance with angular positions. All these transient responses are same as compared to Simulink. [17]

3.5.3 Full State Feedback design by LQR:

The Section 3.5.3 presents Full State Feedback control design with LQR for the Furuta Pendulum system. VisSim does not have dedicated control tool box for designing state space control algorithms thus the Control System toolbox of MATLAB has been used for LQR design and then VisSim engine has been used for simulations. This Section also shows simulation results obtained with VisSim for LQR control design.

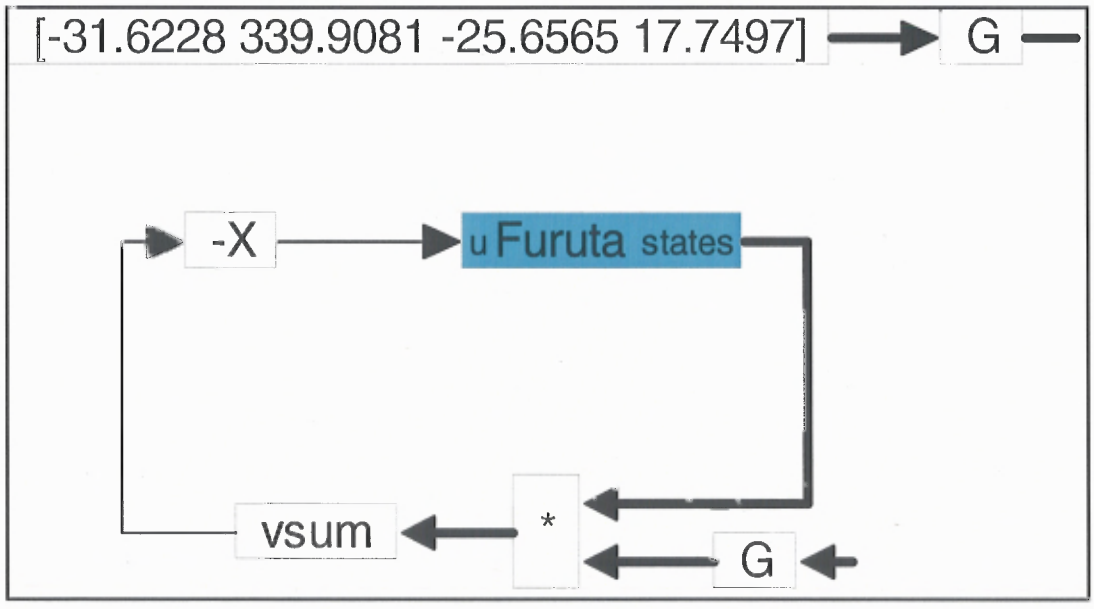


Figure 3.56 VisSim block diagram for full state feedback design by LQR.

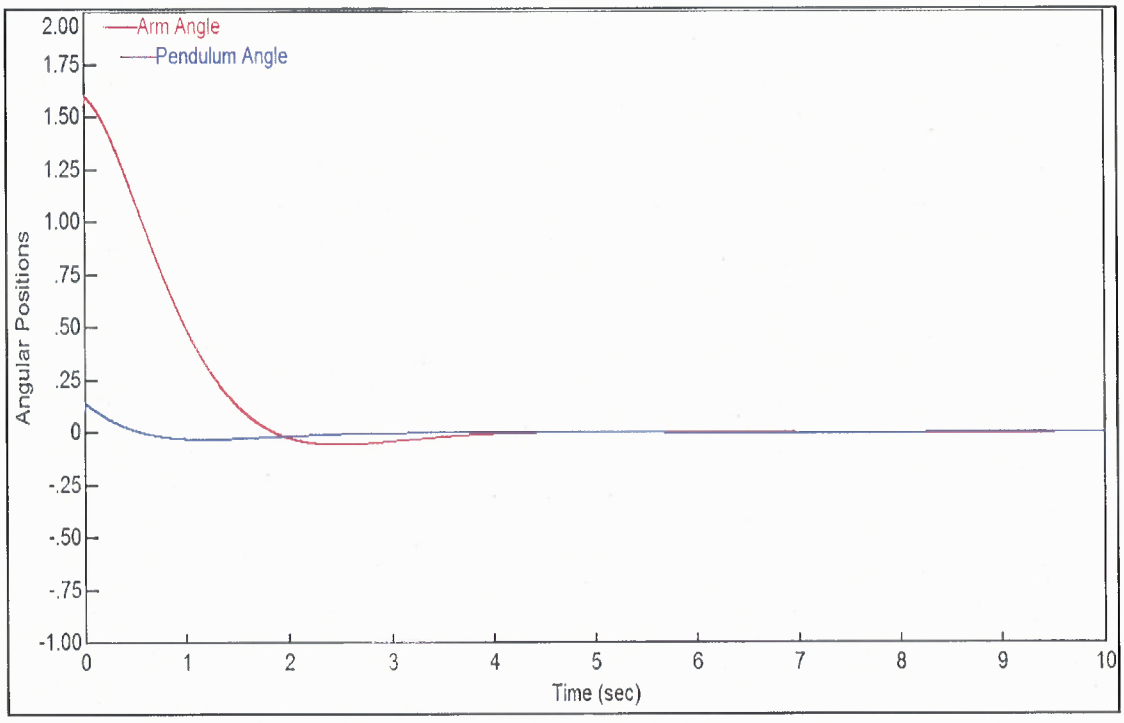


Figure 3.57 VisSim LQR output for the arm angle and the pendulum angle with initial conditions [1.57, 0] on angles.

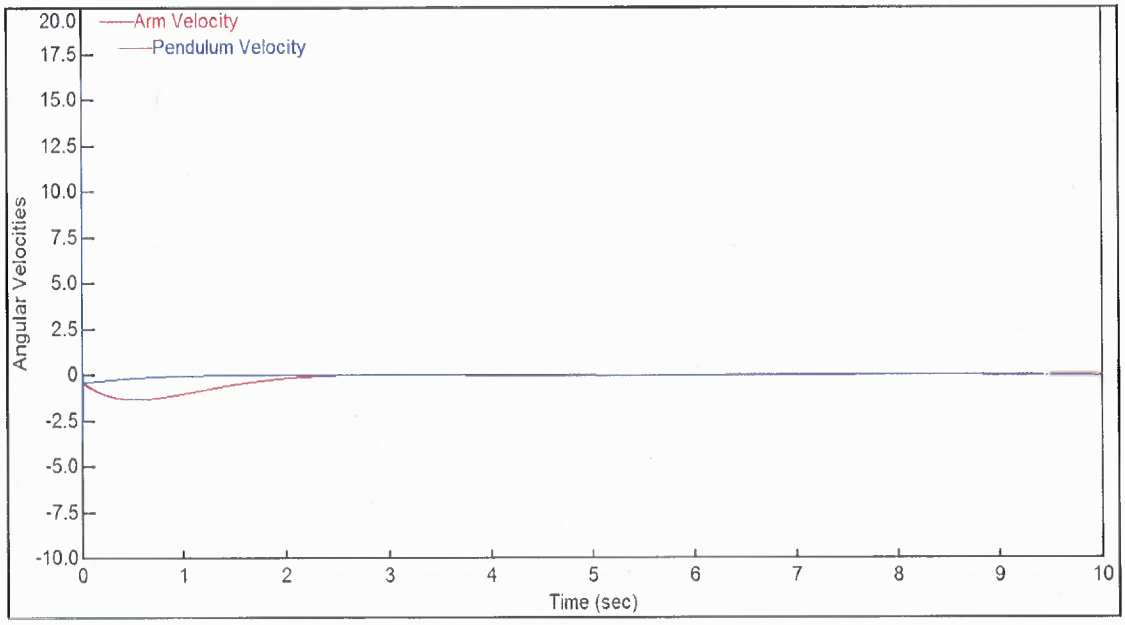


Figure 3.58 VisSim LQR output for the arm velocity and the pendulum velocity with initial conditions [1.57, 0] on angles.

Figure 3.56 shows the block diagram of LQR control for the Furuta pendulum system. It is similar to the VisSim Pole Placement block diagram. The Gain block multiplies the input state vector by matrix gain. After remapping of gain matrix, G = [-31.6228 339.9081 -25.6565 17.7497] has been used to implement state feedback control law u= -Gx. Figure 3.57 shows transient responses of the arm position and the pendulum position for the LQR control. The position behavior of the Scicos LQR transients is exactly same as that of Simulink. Figure 3.58 shows velocity transients of the Furuta pendulum for the LQR control. The velocity behavior of the Scicos LQR is exactly same as that of Simulink. [17]

CHAPTER 4

COMPARISON ANALYSIS

This chapter presents analysis of the comparison study of the various software engines. MATLAB-Simulink has been considered as reference for this comparison study for simulation of closed-loop control systems. This comparison study considers various performance parameters such as control design tools, learning curve, flexibility, tech support and documentation. It also gives information about interoperability of these various software engines. Different distinguishing features of these software engines have been discussed in this chapter.

Control Design Tools

This section presents comparison analysis of control design tools available with various software engines.

- MATLAB-Simulink: MATLAB has a dedicated Control System toolbox to design state space control for non-linear dynamic systems.
- Scilab-Scicos: Scilab-Scicos is an open source work-alike of MATLAB-Simulink. Scilab has a Control System toolbox similar to MATLAB. Most of the Scilab commands are similar to MATLAB with some differences in syntax. Palette blocks in Scicos are very much similar to Simulink library components.
- Maple-MapleSim: Maple has a Control System toolbox with predefined Maple worksheet templates. Maple worksheets are equipped with all tools necessary

- tools to design state space control. Maple Control System toolbox has tight integration with Maple documentation tools to organize control design.
- LabVIEW: LabVIEW has a separate Control Design and Simulation toolbox. LabVIEW contains dedicated Virtual Instruments (VIs) for state space control design and simulation. LabVIEW has MathScript node which is capable of handling M-file scripts from MATLAB.
- VisSim: VisSim does not have a dedicated control design toolbox for state space control design. However, it does have frequency domain control design analysis.

Learning Curve

MATLAB-Simulink being widely known software engine has best learning curve amongst all. Since, MATLAB-Simulink is known software engine it is considered as reference to decide learning curve for other software engines.

- Scilab-Scicos: Scilab-Scicos, being an open source work-alike for MATLAB-Simulink, has a very good learning curve. With available documentation and previous knowledge of MATLAB-Simulink, it is relatively easy to learn Scicos-Scilab.
- Maple-MapleSim: MapleSim has a good learning curve as compared to MATLAB-Simulink. Due to the unique unconventional multi-body modeling approach of Maple-MapleSim, it takes some time to get acquainted with MapleSim physical components.
- LabVIEW: LabVIEW has the steepest learning curve amongst all. There are numerous VIs in control design toolbox of LabVIEW for state space control

design. With LabVIEW, understanding and implementation of those VIs took considerably longer time than that of Simulink.

- VisSim: VisSim has easy learning curve as compared to Simulink. VisSim has complete block diagram approach for dynamic system modeling. This discrete approach makes VisSim easier to learn as compared to the others.

Tech Support

MathWorks, National Instruments, MapleSoft and INRIA have special tech support teams dedicated to serve technical questions of the user community. All of them have structured multi tier tech support system. Technical questions regarding control system domain are handled by control system domain experts. VisSim also has a separate tech support team, but it does not have structured multitier tech support like all the others. MathWorks, National Instruments and MapleSoft have telephone tech support facilities which are the most efficient way to debug and learn more about these software engines. Being a thesis student, MapleSoft and VisSim provided me with exclusive technical support. In case of ordinary users it will be matter of investigation.

Documentation

This section compares quality of documentation available with these software engines to gain required expertise. Different types of documentation such as books, on-line documentation and in-line help documentation have been referred for each software engine.

- MATLAB-Simulink: MATLAB and Simulink both have thorough documentation in their help menu. The Control System toolbox help documentation of MATLAB has extensive examples explaining use of different commands for the state space control design techniques. There are numerous books available which illustrate control design with MATLAB-Simulink.
- Scilab-Scicos: Scilab-Scicos has a standard book available to get started with fundamentals [10]. Scilab has good help documentation to get acquainted with Control System toolbox. However, Scilab-Scicos documentation lack appropriate illustrations with examples for implementing Scilab Control System toolbox commands for user defined applications.
- Maple-MapleSim: Maple has a good help documentation to get started with control design toolbox and multi body modeling approach. Again, depth of contents could be more precise to give better insight into unique physical modeling with MapleSim. There is no standard book available on Maple-MapleSim. The Maple control design worksheet help documentation is good as compared to MATLAB control design toolbox.
- LabVIEW: In order to design state space control in LabVIEW, it has in detail help and on-line documentation available for Virtual Instruments (VIs). Due to intricacy in documentation content it is difficult to understand implementation of these VIs as compared to implementation of Simulink blocks. There are a few standard books available which could be useful to know more about LabVIEW. Extensive on-line documentation is available to learn basics of LabVIEW.

- VisSim: VisSim has brief in-line help documentation of different blocks as compared to Simulink documentation. VisSim help documentation does not illustrate much about implementation part of these available VisSim blocks. There is no standard book available illustrating use of VisSim for control theory.

Flexibility

This is the most crucial criterion to compare different software engines with reference to MATLAB-Simulink.

- MATLAB-Simulink: This software engine has been known for its flexibility and tight integration between MATLAB and Simulink.
- Maple-MapleSim: MapleSim has an edge over Simulink in terms of user friendliness and flexibility in implementing dynamic system models and its state space control. Because of this reason, it has been weighted with an asterisk. The unique ability of MapleSim to model dynamic systems with physical components distinguishes it from Simulink. MapleSim modeling is much efficient and time saving than that of Simulink due to its intuitive nature. Acausal modeling adds great deal of flexibility with availability of multi body physical component library, Parameter Box, Maple Worksheet templates as attachments to MapleSim model and 3-D modeling animation.
- Scilab-Scicos: Scilab-Scicos software engine is an open source work-alike of MATLAB-Simulink with very good flexibility. Scilab has intricate syntaxes for its commands as compared to MATLAB, which makes it relatively less flexible than MATLAB.

- LabVIEW: LabVIEW has overall fair flexibility as compared to MATLAB-Simulink. For LabVIEW Virtual Instruments it is necessary to define different data types by user which makes it less flexible as compared to Simulink blocks. For modeling of a dynamic system with LabVIEW, it is necessary to scalarize vector operations. Unlike Simulink, it does not support vector operations in modeling. Scalarization hampers flexibility in modeling of non linear dynamic system like the Furuta pendulum as compared to the Simulink modeling approach.
- VisSim: VisSim has very good flexibility in modeling dynamic systems. VisSim has less complicated component library with availability of frequently used block icons on task bar for drag and drop convenience. Thou VisSim does not support vector operations, VisSim arithmetic blocks and variable blocks make it easy to implement system equations.

Table 4.1 depicts the comparison summary of various software engines considered. The magnitude weighing has been used on the scale of 1 to 5.

Table 4.1 The Comparison Summary of Various Software Engines

Simulation Engines	Control Design Tools	Learning Curve	Flexibility	Tech Support	Documentation
MATLAB-Simulink	1	1	1	1	1
Scilab-Scicos	1	2	2	1	3
Maple-MapleSim	1	3	1*	1	3
LabVIEW	1	5	4	1	3
VisSim	5	2	2	2	4

1- Best 2-Very Good 3-Good 4-Fair 5-Limited

Interoperability

Table 4.2 presents the interoperability summary of various software engines considered. Interoperability defines compatibility of software engines with each other. There are different conversion tools available in some of these software engines to integrate advantages of two different software engines.

- Scilab-Scicos: Scilab has built-in MATLAB-to-Scilab conversion application tool to import MATLAB scripts into Scilab. However, there is no gateway to import Simulink models into Scicos so one has to build a new model. It was found that overall Scilab-Scicos software engine is about 70% compatible to MATLAB-Simulink.
- Maple-MapleSim: Maple has an add-on product called 'BlockImporter' to import Simulink models into MapleSim. But the 'BlockImporter' tool does not support the Fcn block of Simulink in which nonlinear dynamics of Furuta pendulum have been implemented. 'BlockImporter' could not convert original Furuta pendulum Simulink model into MapleSim. There is a standard list of supported Simulink blocks given by MapleSoft. Overall, it was found that there is Maple-MapleSim has about 50% compatibility with MATLAB-Simulink. To export MapleSim model into Simulink it has an add-on module known as Simulink Connector Toolbox. As for the internal process, MapleSim first generate the equations while running the simulation. Part of that process is to determine the order in which to solve the equations for the simulation. This results in a set of ordered execution sequence. At this point, for the export, MapleSim convert these sequences of expressions into the appropriate code syntax. Together with the generated code,

MapleSim also generates m-script file that can be executed within Simulink to compile and generate the mdl file that uses the compiled DLL binary in Simulink model diagram. Another supported platform is export to LabVIEW EMI, SIT and VeriStand. There is 100% export compatibility from Maple-MapleSim to MATLAB-Simulink as well as to LabVIEW.

- LabVIEW: Based on conversion of the Simulink model of the Furuta pendulum into an equivalent LabVIEW VI, LabVIEW has been assigned 40% compatibility with MATLAB-Simulink. LabVIEW has a Simulation Model Converter Dialog Box. This dialog box is used to convert a Simulink model (.mdl) file, developed in Simulink simulation environment, into a LabVIEW VI that contains a simulation diagram. As part of the conversion process, the Simulation Model Converter uses MATLAB application software and the Simulink application software to compile .mdl file and execute any of .m files that have been specified in the dialog box. This tool does not support MATLAB Fcn block of Simulink so it could not successfully convert the Furuta pendulum Simulink model into LabVIEW VI.
- VisSim: VisSim has been assigned 40% compatibility with MATLAB-Simulink. VisSim has a Simulink File Import tool for importing Simulink models into VisSim block diagram format. This tool does not support MATLAB Fcn block of Simulink as a result it could not convert original Simulink model of the Furuta pendulum into VisSim block diagram.

Table 4.2 The Interoperability Summary of Various Software Engines

Simulation Engines **Output →** **Input↓**	**MATLAB Simulink**	**Scilab Scicos**	**Maple MapleSim**	**LabVIEW**	**VisSim**
MATLAB Simulink	100%	70%	50%	40%	40%
Scicos Scilab	70%	100%	No	No	No
MapleSim Maple	100%	No	100%	100%	No
LabVIEW	No	No	No	100%	No
VisSim	No	No	No	No	100%

CHAPTER 5

CONCLUSION

The comparison study of software engines for simulation of closed-loop control systems elicited several facts regarding performance of these software engines. The software engines such as Scilab-Scicos, LabVIEW, Maple-MapleSim and VisSim have been compared with reference to MATLAB-Simulink. These software engines have been evaluated with regard to various performance criteria by implementing state space control techniques for non linear Furuta pendulum using each one of them. Scilab-Scicos package is an open source work-alike of MATLAB-Simulink. With available documentation and previous knowledge of MATLAB-Simulink; it is easier than other software engines to design and simulate state space techniques with Scilab-Scicos. Scicos-Scilab is the most cost efficient among all others being free open source software alternative to MATLAB-Simulink.

MapleSim has multi body physical modeling (Acausal) which is unique and efficient in saving time for modeling as compared to Simulink. MapleSim has unique ability of generating equations of motion from the 'Acausal' models of the dynamic systems. Except for MapleSim, all software engines require starting with differential equations. MapleSim could detect discrepancies in original available Furuta pendulum's equations of motion after comparing with generated equations with Maple worksheet. With MapleSim it is possible to simulate 3-D animation of a dynamic system model and its control. It gives better insight into control simulation as compared to other simulation engines. MapleSim has the very efficient documentation and control design interface as

compared to others. MapleSim has most convenient structure to organize control design documentation. LabVIEW has a most intuitive user interface known as 'Front panel' as compared to others. It is also equipped with a wide array of control design and simulation tools. It is experienced that the LabVIEW learning curve is the steepest as compared to other software engines. VisSim has great flexibility and ease of learning as compared to Simulink. Unlike MATLAB, it does not have control tool box for state space control design. Above all, Maple-MapleSim software engine is equipped with some unique features for simulation of closed- loop control systems.

REFERENCES

[1] Wikipedia. "Furuta Pendulum."Internet:http://en.wikipedia.org/wiki/Furuta_pendulum, July 2008

[2] Cazzolato, B.S and Prime, Z (2008) "The Dynamics of the Furuta Pendulum", Technical Report, The University of Adelaide. Internet:http://www.mecheng.adelaide.edu.au/robotics_novell/projects/2004/Pendulum/Furuta Pedulum_Internal_report.pdf

[3] MathWorks. "Simulink User's Guide." Internet: http://www.mathworks.com/access/helpdesk/help/toolbox/simulink/ug/bsejk7m-1.html, March 2010.

[4] MathWorks. " Control System Toolbox™ 8 Reference." Internet: http://www.mathworks.com/access/helpdesk/help/pdf_doc/control/reference.pdf, March 2010.

[5] MathWorks. " Control System Toolbox™ 8 User's Guide." Internet: http://www.mathworks.com/access/helpdesk/help/pdfdoc/control/usingcontrol.pdf , March 2010.

[6] Bernard Friedland.*Control system design: an introduction to state-space methods.* Mineola, NY : Dover Publications, 2005, c1986.

[7] National Instruments. *LabVIEW Tutorial Manual.* Austin, TX, January 1996.

[8] National Instruments. *LabVIEW Control Design Toolkit User Manual.* Austin, TX, February 2006.

[9] National Instruments. "Introduction to LabVIEW in 3 Hours for Control Design and Simulation." Internet: http://zone.ni.com/devzone/cda/tut/p/id/5855

[10] Stephen L. Campbell, Jean-Philippe Chancelier, and Ramine Nikoukhah. *Modeling and Simulation in Scilab/Scicos*. New York : Springer Science+Business Media, 2006.

[11] Finn Haugen. "Master Scilab." Internet: http://home.hit.no/~finnh/scilab_scicos/scilab/, April 2008.

[12] Finn Haugen. "Master Scicos." Internet: http://home.hit.no/~finnh/scilab_scicos/scicos/index.htm, April 2008.

[13] Ramine Nikoukhah, Serge Steer. "*SCICOS- A Dynamic System Builder and Simulator User's Guide - Version 1.0."* INRIA, France, June 1997.

[14] INRIA. " *ScilabManual.*" Internet: http://www.scilab.org/download/5.2.2/manual_scilab-5.2.2_en_US.pdf.

[15] Maplesoft. *Maple™ 13- The Essential Tool for Mathematics and Modeling User Manual.* Waterloo, Canada, 2009.

[16] Maplesoft. *Maplesim™ 3- High-Performance Multi-Domain Modeling and Simulation User's Guide.* Waterloo, Canada, 2009.

[17] Visual Solutions, Inc. *VisSim User's Guide*. Westford, Massachusetts, 2009.