EE-479 Digital Control System
Project 3
Rotary Inverted Pendulum

Purpose

The purpose of this project is to design hybrid controllers for balancing the rotary inverted pendulum in the vertical-upright position. The objectives of this project are:

- To obtain a nonlinear model of the rotary inverted pendulum.
- To obtain a linear model in the neighborhood of an equilibrium state.
- To verify the linear and nonlinear models by constructing the nonlinear and the linear models in Simulink and simulating in tandem. Check to see if the linear model is indeed a true model of this system over a specific operating range.
- To design a state feedback controller that manipulates the servomotor angle to keep the pendulum in the upright position. The controller must have good stability margins, and should accommodate model uncertainty. (Sensor noise, friction of motor, etc.). A catch controller must be designed to activate the state feedback controller when the pendulum is placed in a small neighborhood of the inverted equilibrium.
- To design a swing up controller that raises the pendulum to the inverted position in a controlled manner, where the state feedback controller can stabilize it (Self-erecting pendulum).
- To build the compensated system in SIMULINK and simulate offline.
- To build the WinCon application, implement and test the system on the real-time hardware.

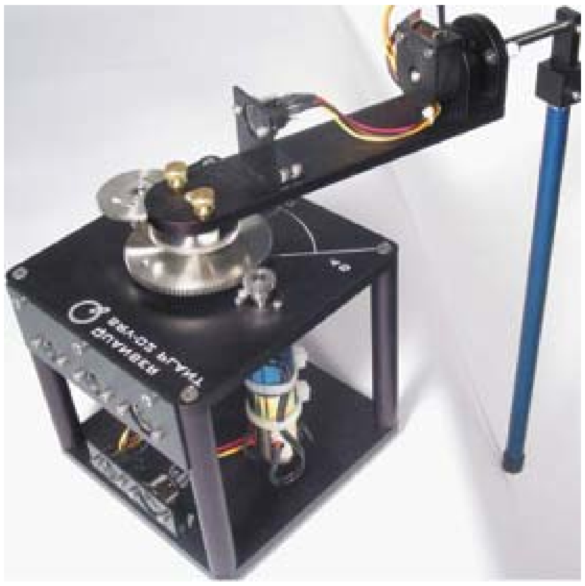

Figure 3.1 Inverted pendulum coupled to SRV02 Servomotor in the high gear ratio

Introduction

The control of an inverted pendulum is a well-known and a challenging problem that serves as a popular benchmark in modern control system studies. The inverted pendulum consists of a rigid link attached to a pivot arm. The arm must move in the horizontal plane in such a way as to keep the pendulum in an upright position. This is similar to the classic pendulum mounted on a cart which must move in such a way as to keep the pendulum in an upright position.

An inverted pendulum is an inherently unstable system by nature. Force must be properly applied to keep the pendulum in an upright position. This is similar to the problem of balancing a broomstick on the palm of your hand. You must constantly adjust the position of your hand to keep the object upright.

The problem of balancing an unstable system occurs in practice in the areas of missile stabilization, control of a space rocket during takeoff and control of robots.

1. Mathematical Modeling

1.1 Servomotor Model

In the position control experiment (EE-371 Lab 5) the servomotor model was developed with the following block diagram as shown in Figure 3.2.

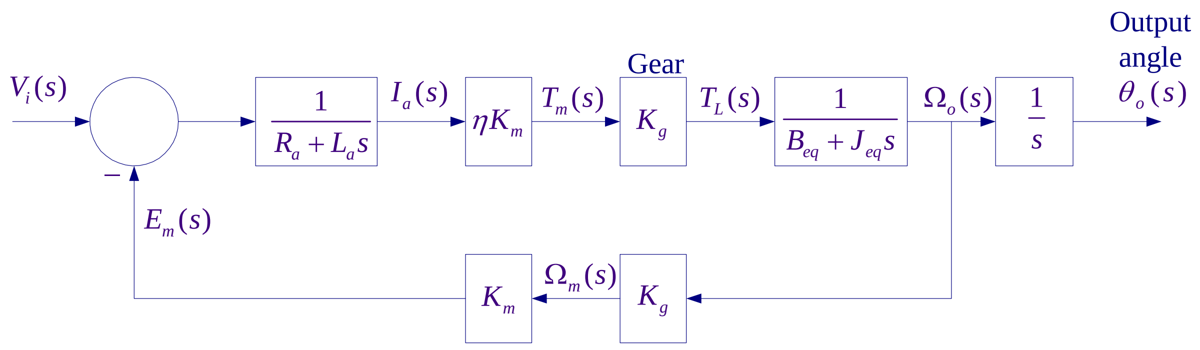

Figure 3.2 Servo plant block diagram

$$\frac{\theta_o(s)}{V_i(s)} = \frac{\dfrac{\eta K_m K_g}{R_a J_{eq}}}{s\left(s + \dfrac{B_{eq}}{J_{eq}} + \dfrac{\eta K_m^2 K_g^2}{R_a J_{eq}}\right)} \tag{3.1}$$

or

$$\frac{\theta_o(s)}{V_i(s)} = \frac{a_m}{s(s+b_m)} \tag{3.2}$$

Where

$$a_m = \frac{\eta K_m K_g}{R_a J_{eq}}, \qquad b_m = \frac{B_{eq}}{J_{eq}} + \frac{\eta K_m^2 K_g^2}{R_a J_{eq}} \tag{3.3}$$

Also from the block diagram the s-domain output torque $T_L(s)$ is

$$T_L(s) = \frac{\eta K_m K_g}{R_a}\left[V_i(s) - K_m K_g \Omega_0(s)\right]$$

Therefore the expression for the output torque in time-domain is

$$T_L(t) = K_1 v_i(t) - K_2 \omega(t) \qquad (3.4)$$

where

$$K_1 = \frac{\eta K_m K_g}{R_a}$$
$$K_2 = \frac{\eta K_m^2 K_g^2}{R_a} \qquad (3.5)$$

In this project the servomotor is arranged for the high gear ratio as shown in Figure 3.3. For this configuration the gear ratio is $K_g = (14)(5)$.

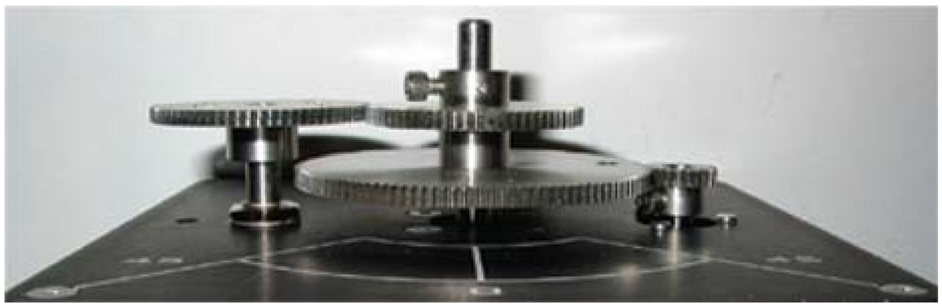

Figure 3.3 high gear ratio configurations

The system parameters are as follows:
Armature resistance, $R_a = 2.6\ \Omega$
Motor voltage constant, $K_m = 0.00767$ V-s/rad
Motor torque constant, $K_\tau = 0.00767$ N-m/A
Armature inertia, $J_m = 3.87 \times 10^{-7}$ Kg m^2
Tachometer inertia, $J_{tach} = 0.7 \times 10^{-7}$ Kg m^2
High gear ratio, $K_g = (14)(5)$
Equivalent viscous friction referred to the secondary gear $B_{eq} = K_g^2 B_m + B_L \simeq 4 \times 10^{-3}$ Nm/(rad/s)
Motor efficiency due to rotational loss $\eta_{mr} \simeq 0.87$
Gearbox efficiency, $\eta_{gb} \simeq 0.85$
$\eta = \eta_{mr}\eta_{gb} = (0.87)(0.85) = 0.7395$

Gear inertia:

$$J_{120} = 4.1835\times10^{-5} \text{ Kg m}^2$$
$$J_{72} = 5.4435\times10^{-6} \text{ Kg m}^2$$
$$J_{24} = 1.0081\times10^{-7} \text{ Kg m}^2$$

Load inertia, $J_L = J_{120} + 2(J_{72}) + J_{24} = 5.2823\times10^{-5} \text{ Kg m}^2$

$$J_{eq} = (J_m + J_{tach})Kg^2 + J_l = (3.87+0.7)10^{-7}(14\times5)^2 + 5.2823\times10^{-5} = 0.0023 \text{ Kg m}^2$$

1.2 The Inverted Pendulum Model

A schematic picture of the rotary inverted pendulum of length l and mass m supported by a pivot arm is shown in Figure 3.4. The pivot arm of radius r rotates in a horizontal plane by the servomotor. The arm must be moved in such a way that the pendulum is in the upright position. One encoder is used to measure the angular position and velocity of the servomotor and another encoder is used to measure the pendulum angle and velocity.

The angle of the pendulum α is defined to be zero when in upright position and positive when the pendulum is moving clockwise. The angle of the arm θ is positive when the arm is moving in clockwise direction. The parameters of the pendulum are defined as follows:

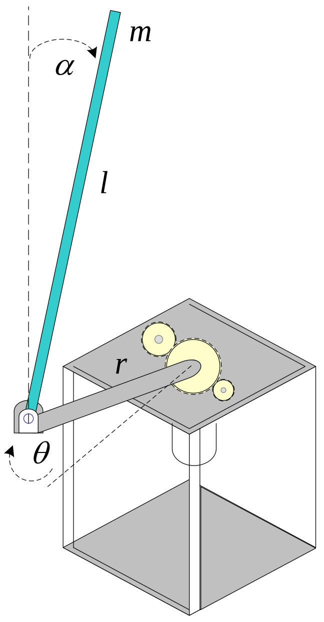

Figure 3.4 Simplified Inverted Pendulum schematic

r	Arm radius ($r = 21.5$ cm)
θ	Servo gear angular displacement
ω	Servo gear angular velocity
α	Pendulum angular deflection
υ	Pendulum angular velocity

L	Pendulum length ($L = 16.75$ cm)
m	Pendulum mass ($m = 0.125$ Kg)
h	Distance of Pendulum Center of mass from ground
J_{cm}	Pendulum Inertia about its center of mass
v_x	Velocity of Pendulum Center of mass in the x-direction
v_x	Velocity of Pendulum Center of mass in the y-direction
g	Gravitational acceleration ($g = 9.8$ m/s^2)

From the above definition

$$\begin{aligned}\dot{\theta} &= \omega \\ \dot{\alpha} &= \upsilon\end{aligned} \qquad (3.6)$$

The governing differential equations of the system are as follows

$$(J_{eq} + mr^2)\dot{\omega} - \frac{1}{2}mLr\cos\alpha\ \dot{\upsilon} + \frac{1}{2}mLr\sin\alpha\ \upsilon^2 + B_{eq}\omega = T_L \qquad (3.7)$$

$$\frac{1}{3}mL^2\dot{\upsilon} - \frac{1}{2}mLr\cos\alpha\ \dot{\omega} - \frac{1}{2}mgL\sin\alpha = 0 \qquad (3.8)$$

or

$$a\dot{\omega} - b\cos\alpha\ \dot{\upsilon} + b\sin\alpha\ \upsilon^2 + B_{eq}\omega = K_1 v_i(t) - K_2\omega \qquad (3.9)$$

$$c\dot{\upsilon} - b\cos\alpha\ \dot{\omega} - d\sin\alpha = 0 \qquad (3.10)$$

where

$$\begin{aligned}a &= J_{eq} + mr^2 \\ b &= \frac{1}{2}mLr \\ c &= \frac{1}{3}mL^2 \\ d &= \frac{1}{2}mgL\end{aligned} \qquad (3.11)$$

Solving for $\dot{\omega}$ from (3.10), we have

$$\dot{\omega} = \frac{c}{b\cos\alpha}\dot{\upsilon} - \frac{d}{b\cos\alpha}\sin\alpha \qquad (3.12)$$

Substituting for $\dot{\omega}$ in (3.9) from (3.12) and solving for $\dot{\upsilon}$ and $\dot{\omega}$, results in the following nonlinear equations

$$\dot{\upsilon} = \frac{1}{f(u)}[ad\sin\alpha - (pb\cos\alpha)\omega - (b^2\cos\alpha\sin\alpha)\upsilon^2 + (bK_1\cos\alpha)v_i(t)] \qquad (3.13)$$

$$\dot{\omega} = \frac{1}{f(u)}[db\cos\alpha\sin\alpha - (cp)\omega - (cb\sin\alpha)\upsilon^2 + (cK_1)v_i(t)] \qquad (3.14)$$

where

$$p = B_{eq} + K_2$$
$$f(u) = ac - b^2 \cos^2 \alpha \tag{3.15}$$

The above equations describing the dynamics of the model are highly nonlinear. The servomotor viscous friction has been included, but the friction between the pendulum and the rotating arm has been neglected. Nonlinearity also arises due to the Coulomb friction which gives rise to limit cycles. The Coulomb friction compensation can be considered by amending the control law with a step function.

1.3 Linearized Model

In order to design a linear regulator state feedback, we need to linearize the model. Equations (3-13) and (3-14) can be linearized by considering the equilibrium state of the system. If we assume α is small (i.e., when the Inverted Pendulum is near its equilibrium point), we can linearize these equations. For small α, $\sin\alpha \approx \alpha$ and $\cos\alpha \approx 1$. Also, for small α, υ^2 is negligible, and we get the following linearized equations

$$a\dot{\omega} - b\dot{\upsilon} + B_{eq}\omega = K_1 v_i(t) - K_2\omega \tag{3.16}$$

$$c\dot{\upsilon} - b\dot{\omega} - d\alpha = 0 \tag{3.17}$$

or

$$\dot{\upsilon} = \frac{ad}{e}\alpha - \frac{pb}{e}\omega + \frac{bK_1}{e}v_i(t) \tag{3.18}$$

$$\dot{\omega} = \frac{bd}{e}\alpha - \frac{pc}{e}\omega + \frac{cK_1}{e}v_i(t) \tag{3.19}$$

where

$$e = ac - b^2$$
$$p = B_{eq} + K_2 \tag{3.20}$$

The linearized model is used to design the stabilizing controller. We now obtain an state-space model

$$\dot{\boldsymbol{x}}(t) = \boldsymbol{A}\,\boldsymbol{x}(t) + \boldsymbol{B}\,u(t) \tag{3.21}$$

$$y(t) = \boldsymbol{C}\,\boldsymbol{x}(t) \tag{3.22}$$

For the combined servomotor and the Inverted Pendulum module, we choose the state variables $\boldsymbol{x}(t)$ as

$$\boldsymbol{x}(t) = \begin{bmatrix} \theta & \alpha & \omega & \upsilon \end{bmatrix}^T$$

By combining (3.6), (3.19) and (3.18) we obtain the following linear state space model of the inverted pendulum.

$$\begin{bmatrix} \dot{\theta} \\ \dot{\alpha} \\ \dot{\omega} \\ \dot{\upsilon} \end{bmatrix} = \begin{bmatrix} 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ 0 & \frac{bd}{e} & -\frac{pc}{e} & 0 \\ 0 & \frac{ad}{e} & \frac{-pb}{e} & 0 \end{bmatrix} \begin{bmatrix} \theta \\ \alpha \\ \omega \\ \upsilon \end{bmatrix} + \begin{bmatrix} 0 \\ 0 \\ \frac{cK_1}{e} \\ \frac{bK_1}{e} \end{bmatrix} v_i(t) \tag{3.23}$$

2. Control Strategy

The controller consists of three parts: a *swing-up controller*, a *catch controller*, and a state *feedback stabilizing controller*.

Manual Mode – For this mode, the pendulum is raised manually by hand towards the upright position (no swing up controller) until it is captured by the catch-controller which turns on the state feedback stabilizing controller.

Self-erecting pendulum - Initially, the pendulum is in the downward position and the swing-up controller is used to bring it to the top position. Once it is sufficiently close to the top equilibrium position, it is caught and switched to the state feedback stabilization controller.

2.1 Stabilizing Controller Design

(a) Pole-placement

State-space feedback is the most important aspect of modern control system. By proper state feedback, unstable systems can be stabilized or damping of oscillatory systems can be improved. One basic approach is known as the *pole-placement design*. Pole placement design allows the placement of poles at specified locations, provided the system is controllable**.** This results in a regulator with constant gain vector $\boldsymbol{K}$. However, the basic pole placement method is strongly dependent on the availability of an accurate model of the system. Other approach to the design of regulator systems is the optimal control problem where a specified mathematical performance criterion is minimized. In optimal regulator design random disturbances can be rejected and the closed-loop system can be made insensitive to changes of plant dynamics.

In pole-placement design the control is achieved by feeding back the state variables through a regulator with constant gains. Consider a linear continuous-time controllable system, modeled in state-space form as given by (3.21) - (3.22). The block diagram of the system with the following state feedback control is shown in Figure 3.5.

$$u(t) = -\ \boldsymbol{Kx}\ (t) \tag{3.24}$$

where $\boldsymbol{K}$ is a $1 \times n$ matrix of constant feedback gain. The control system input is assumed to be zero. The purpose of this system is to return all state variables to values of zero when the states have been perturbed.

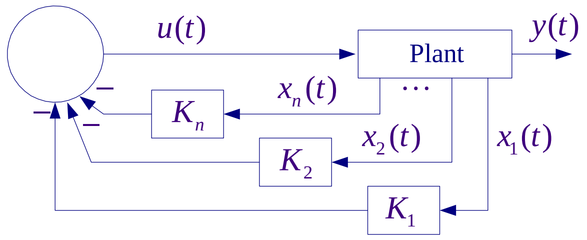

Figure 3.5 Control system design via pole placement.

Substituting (3.24) into (3.21), the closed-loop system state-variable representation become

$$\dot{\boldsymbol{x}}(t) = [\boldsymbol{A} - \boldsymbol{BK}]\,\boldsymbol{x}(t) = \boldsymbol{A}_f\,\boldsymbol{x}(t) \qquad (3.25)$$

The design objective is to find the gain matrix $\boldsymbol{K}$ such that the characteristic equation for the controlled system is identical to the desired characteristic equation. The derivation when matrix $\boldsymbol{A}$ is in phase variable control canonical form is straightforward. When the system is not in phase variable form, first the system is transformed into the phase variable control canonical form; refer to your textbook and lecture notes or to "Computational Aids in Control Systems using MATLAB, Hadi Saadat, McGraw-Hill 1993, Chapter 8, page 170." A custom-made function named **[K, Af] = placepol(A, B, C, P)** is developed for the pole placement design. **A**, **B**, **C** are system matrices and P is a row vector containing the desired closed-loop poles. This function returns the gain vector $\boldsymbol{K}$ and the new system matrix **Af**. Also, the MATLAB Control System Toolbox contains two functions for pole-placement design. Function **K = acker(A, B, P)** is for single input systems, and function **K = place(A, B, p)** suitable for multi-input systems. The plant described by (3.21) with the system matrix having dimension $n \times n$ is completely state controllable if and only if the controllability matrix S has a rank of n. [Refer to Chapter 8 p. 175 in the above reference]. A function **S = cntrable(A, B)** is developed which returns the controllability matrix S and determines whether or not the system is state controllable. Also, the function **ctrb(A, B) in** MATLAB Control system toolbox can be used to determine the controllability matrix. In pole-placement design, it was assumed that all state variables are available for feedback. However, in practice it is impractical to install all the transducers, which would be necessary to measure all of the states. If the state variables are not available an observer or estimator is designed.

(b) Optimal Regulator

Instead of the pole-placement design described above, the stabilization controller can be designed using the Linear Quadratic Regulator (LQR). The object is to determine the optimal controller $u(t) = -\boldsymbol{K}\,\boldsymbol{x}(t)$ such that a given performance index $\boldsymbol{J} = \int (\boldsymbol{x}^T\boldsymbol{Q}\boldsymbol{x} + \boldsymbol{u}^T\boldsymbol{R}\boldsymbol{u})\boldsymbol{dt}$ is minimized. The performance index is selected to give the best performance. The choice of the elements of Q and R allows the relative weighting of individual state variables and individual control inputs. For example, using an identity matrix for Q weights all the states equally. As a starting point you may use a diagnol matrix with values $Q = diag([4 \quad 40 \quad 0.1 \quad 0.5])$ and $R = 1$.The MATLAB Control System Toolbox function **[k, S] = lqr2(A, B, Q, R)** calculates the optimal feedback matrix K such that it minimizes the cost function J subject to the constraint defined by the state equation. Also returned is S, the steady-state solution to the associated algebraic Riccati equation. Refer to your textbook and lecture notes or to "Computational Aids in Control Systems using MATLAB, Hadi Saadat, McGraw-Hill 1993, Chapter 8, page 180

2.2 Closed-loop poles specifications ─ Pole-placement

The state equation obtained in (3.25) results in a fourth order characteristic equation. For the compensated closed-loop system we choose the poles as a desired pair of dominant second-order poles, and select the rest of the poles to have real parts corresponding to sufficiently damped modes so that the system will mimic a second-order response with reasonable control effort. Pick the low-frequency modes and the high-frequency modes as follows:

- A pair of dominant second-order poles with a time constant of $\tau = \frac{1}{3}$ second and a damping ratio of $\zeta = 0.8$.
- Select two real poles with very small time constants $\tau_3 = 0.125$ second and $\tau_4 = 0.025$ second.

3. Pre Laboratory Assignment

3.1 Verification of the liner model

A Simulink diagram is drawn for the nonlinear differential equations (3.13) and (3.14), with parameters defined in (3.11) and (3.15). This simulation diagram which is turned into a subsystem is shown in Figure 3.6

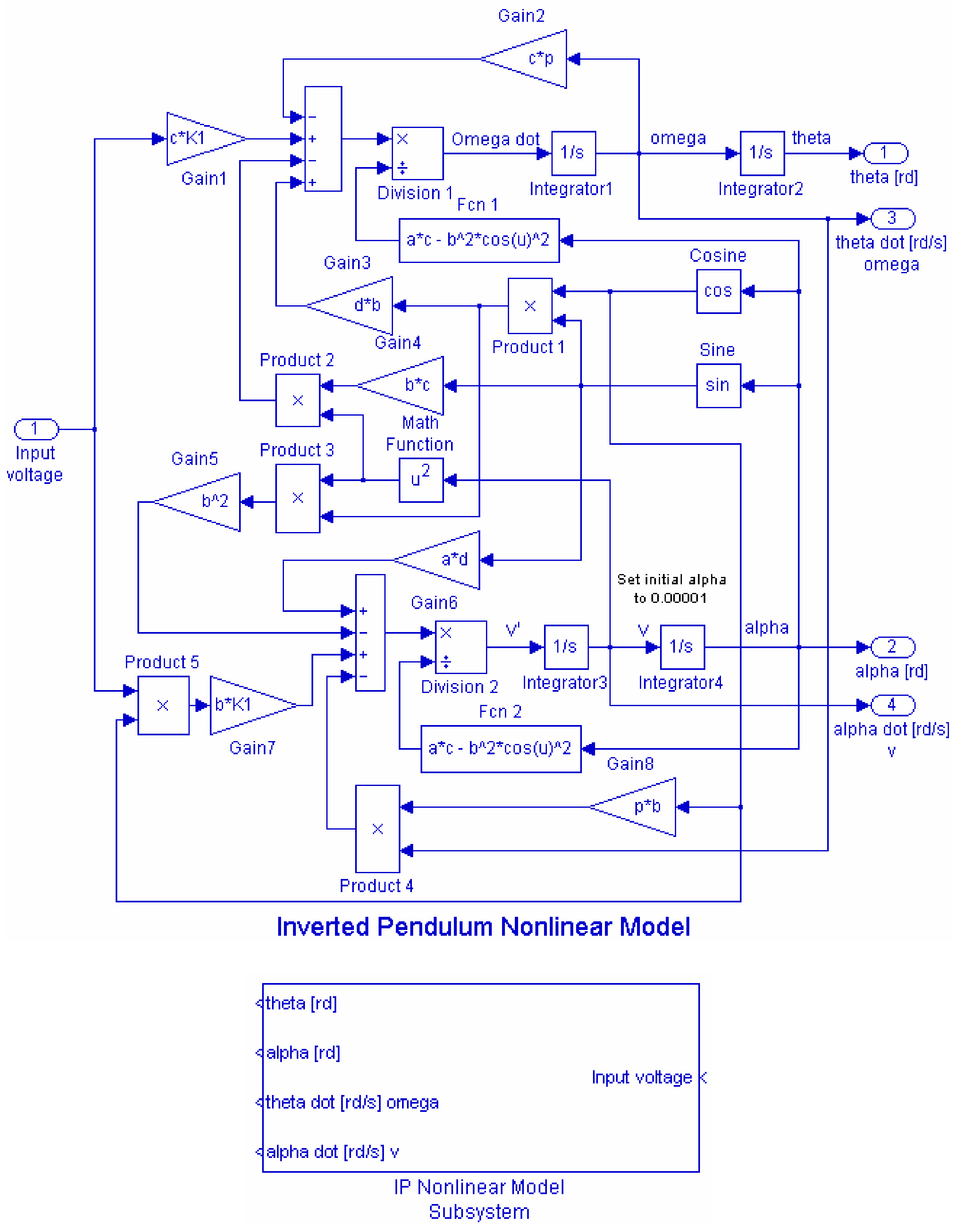

Figure 3.6 Simulink diagram for the Inverted Pendulum Nonlinear model.

The state variable simulation diagram for the linear model is shown in Figure 3.7

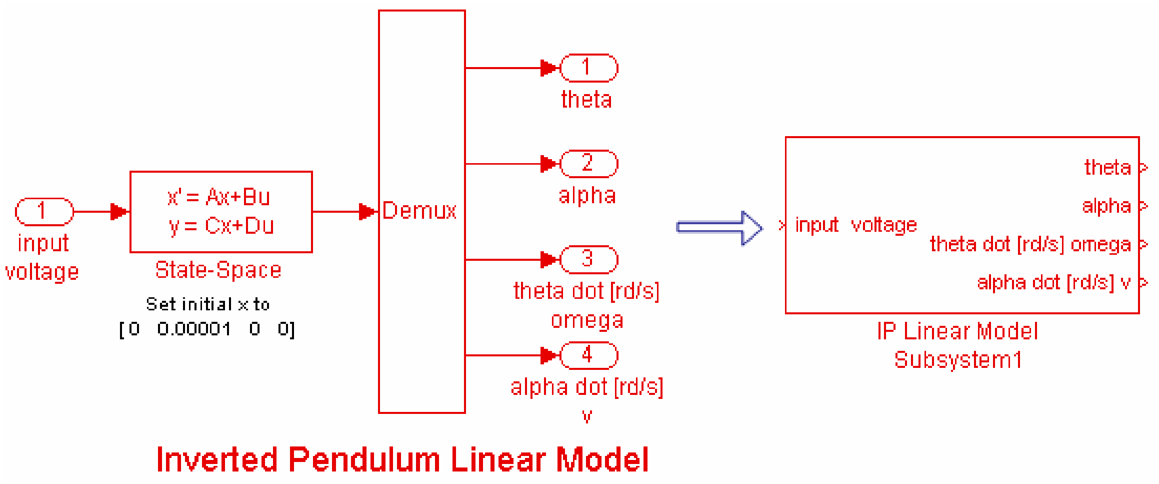

Figure 3.7 Simulink diagram for the Inverted Pendulum linear model.

In the nonlinear model, open the Integrator 4 and set the initial angle (alpha) to about 0.00001Radian. Also open the state model dialog box and set the initial conditions to [0 0.00001 0 0]. Copy the two Subsystems in a new Simulink page, place the necessary scopes for the purpose of comparing the two models shown in Figure 3.8, and save as "Model_Comparison.mdl.".

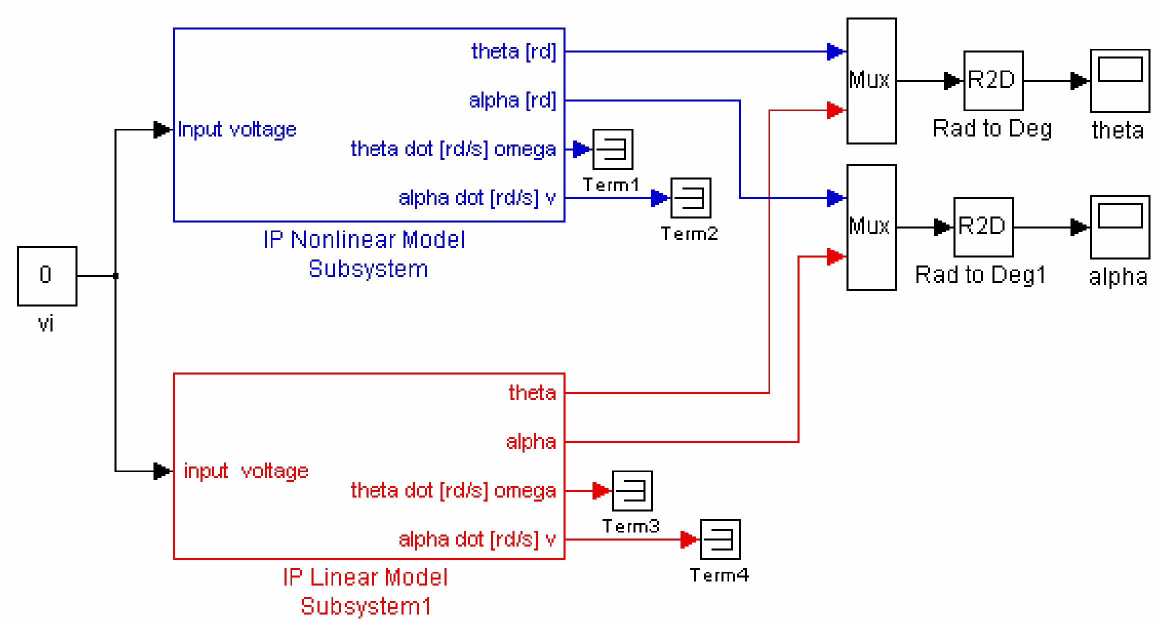

Figure 3.8 Simulink simulation diagram for combined Nonlinear and linear models

Since both angles α and θ increases rapidly even for a small perturbation, set the Simulation Stop Time to about 1.6 seconds to see the comparison for a small range of α. Simulate and examine the plots for α, and θ, and determine the range of α for which the linear model is in close agreement with the nonlinear response. From this observation a threshold angle α_{th} is obtained for the state feedback controller.

3.2 Open-loop Analysis

Define the servomotor and Inverted Pendulum parameters in a script m-file and evaluate the system $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices from (3.23). Use **damp(A)** to obtain the roots, damping ratios and natural frequencies of the uncompensated characteristic equation. Comment on the open-loop system stability and obtain the open-loop step response. Consider a step input of 25°. To find the open-loop step response, following the $\boldsymbol{A}$ and $\boldsymbol{B}$ matrices, define $\boldsymbol{C} = \boldsymbol{eye}(4)$, and $\boldsymbol{D} = \boldsymbol{zeros}(4,1)$, and add

```
[numo, deno] = ss2tf(A, B, C, D, 1)
Gp = tf(numo(1, :), deno)
damp(A)
ltiview('step', 25*Gp)
```

Run the program and show that the system is a non minimum phase system.

Use function **S = cntrable(A, B)** or **S = ctrb(A, B)** to determine the system controllability.

3.3 Regulator Design

Based on the design specifications given in section 2.2 locate the desired closed-loop poles and form the vector **P** containing the four poles. Use function **[K, Af] = placepol(A, B, C, P)** to obtain the gain vector **K** and the new system matrix **Af**. Add the following statements to obtain the compensated transfer function and the step response to a 25° step input

```
[K,  Af] = placepol(A, B, C, P)
[numc, denc] = ss2tf(Af, B, C, D, 1)
Tc = tf(numc(1, :), denc)
ltiview('step', 25*Tc)
damp(Af)
```

Use **damp** function to obtain the compensated system roots and confirm the pole-placement design. Comment on the system response.

Alternatively apply LQR design with $Q = diag([40 \quad 40 \quad 0.1 \quad 0.5])$ and $R = 1$, and determine the state-feedback gain matrix K.

3.4 Digital Simulation

Construct a SIMULINK simulation diagram named "InvPen_sim.mdl" for the compensated system with state feedback as shown in Figure 3.9. In the Simulink block diagram you can replace $\boldsymbol{A},\ \boldsymbol{B},\ \boldsymbol{C},\ \boldsymbol{D},\ \ K(1)$, and the gain matrix $\boldsymbol{K}$ with the computed values. Alternatively, you can place all equations in a script m-file to compute these parameters, which are sent to the MATLAB Workspace, then simulate the Simulink diagram. From the Simulink/Simulation Parameters select the Solver page and for Solver Option Type select Fixed-step and ode4 (Runge-Kutta) and set the fixed step size to 0.001.

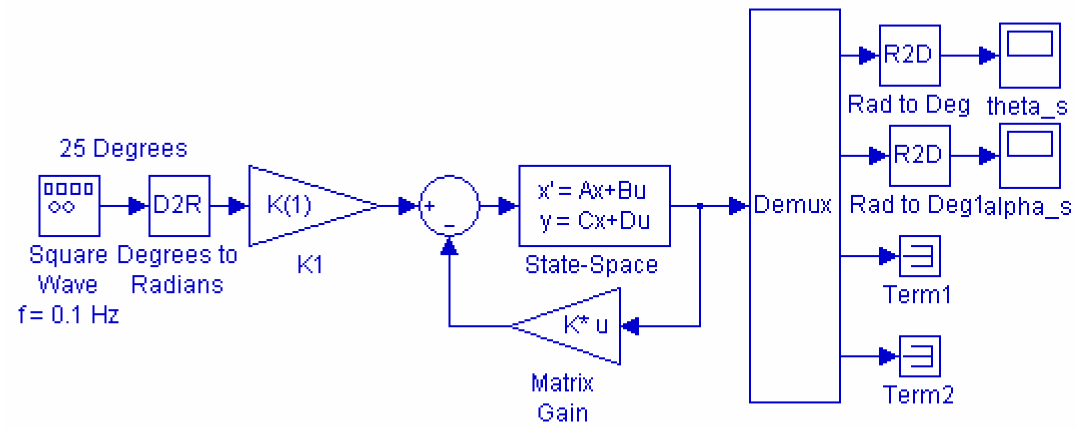

Figure 3.9 Simulink simulation diagram for the Inverted Pendulum

Obtain the response to a square input of amplitude 25 degrees and frequency 0.1 Hz.

3.5 Swing-up Controller

As you have found out the linearized model is accurate when the pendulum angle α is within a small range, $\pm 30^\circ$ (verify this from the simulation results in 3.1). Therefore, the state feedback stabilization controller is accurate when pendulum is within this region. In the manual mode, the pendulum is brought up to the vertical position by hand. Alternatively a Swing-up controller can be designed to swing the pendulum to the upright position from the stable 'rest' position. This controller is responsible for the swinging up the pendulum to a position in which the stabilization mode can takeover. First the servomotor is placed under position control then an algorithm is prescribed for the deriving force.

In this design the pendulum motion is ignored, but an approximate moment of inertia of 0.005 Kg m^2 to account for the rotation of arm and the pendulum mass is considered. That is, the servomotor equivalent inertia J_{eq} is replaced by $J_{eq} + 0.005$. Based on this value, compute a_m and b_m from (3.3), and obtain the servomotor transfer function given in (3.2). As in position control (EE371, Lab 5) we use a rate feedback and a position feedback given by

$$V_i(s) = K_P[\theta_i(s) - \theta_o(s)] - K_D\Omega_o(s) \qquad (3.26)$$

This feedback loop is shown in Figure 3.10. The purpose of this system is to have the output angle, $\theta_o(t)$ follow the desired position $\theta_i(t)$. Design the inner loop for position control to meet the following time-domain specifications:

- Step response damping ratio $\zeta = 0.8$
- Step response peak time $t_p = 0.115$ second

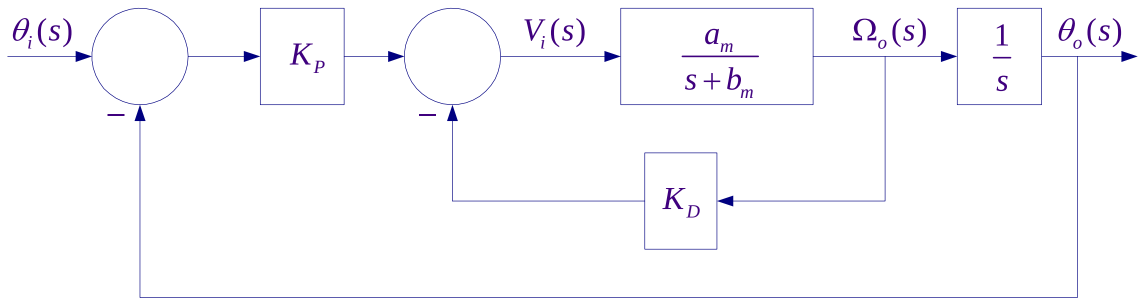

Figure 3.10 Position Control using position and rate feedback.

Applying the Mason's gain formula, the overall transfer function becomes

$$\frac{\theta_o(s)}{\theta_i(s)} = \frac{K_P a_m}{s^2 + \left(K_D a_m + b_m\right)s + K_P a_m} \tag{3.27}$$

The second-order response peak time t_p, is given by

$$t_p = \frac{\pi}{\omega_n\sqrt{1-\zeta^2}} \tag{3.28}$$

The servo motor transfer function has the same form as the standard second-order transfer function

$$\frac{\theta_o(s)}{\theta_i(s)} = \frac{\omega_n^2}{s^2 + 2\zeta\omega_n s + \omega_n^2}$$

From (3.27), we have

$$K_P = \frac{\omega_n^2}{a_m}\ , \qquad K_D = \frac{2\zeta\omega_n - b_m}{a_m} \tag{3.29}$$

From the given specifications compute ω_n, and use (3.29) to determine K_P, and K_D.

Many schemes can be devised to prescribe a driving force for a suitable trajectory in a controlled manner that energy is gradually added to the system to bring the pendulum to the inverted position. You may be asked to design one or more of the following schemes:

(a) Simple Rectangular Reference Input

Construct a subsystem Simulink diagram for the Swing-up Controller with position and rate feedback. In order to safeguard against the excessive rotation of the servomotor angle θ during swing-up, or when subjected to extreme tap disturbance in the stabilization mode, a rational operator is added to stop the simulation when θ exceeds $\pm 2\pi/5$, (i.e., ± 72 degrees) as shown in Figure 3.11.

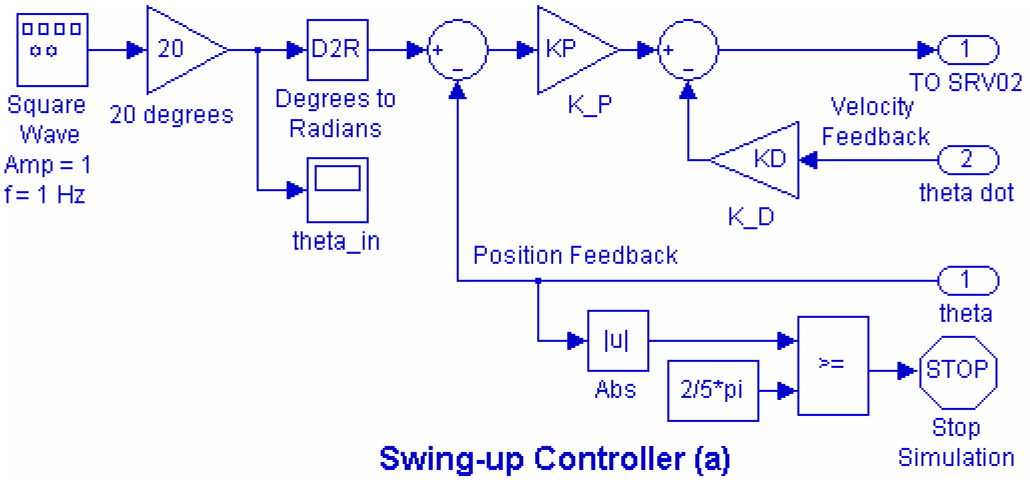

Figure 3.11 Simulink diagram for the Swing-up Controller

A square-wave input of 1 Hz with a gain of 20° was found to be sufficient to bring-up the pendulum to the upright position. You can adjust the magnitude or the frequency to obtain a more satisfactory swing-up. In the implementation part each time you change the frequency you need to build and download the model.

Connect the Swing-up Controller Subsystem (a) to the IP Nonlinear model as shown in Figure 3.12. In the nonlinear model, open the Integrator 4 and set the initial angle (alpha) to $-\pi$ Radians (-pi). For a square wav input of 1Hz, and amplitude 20 obtain the response for θ_o, and α .

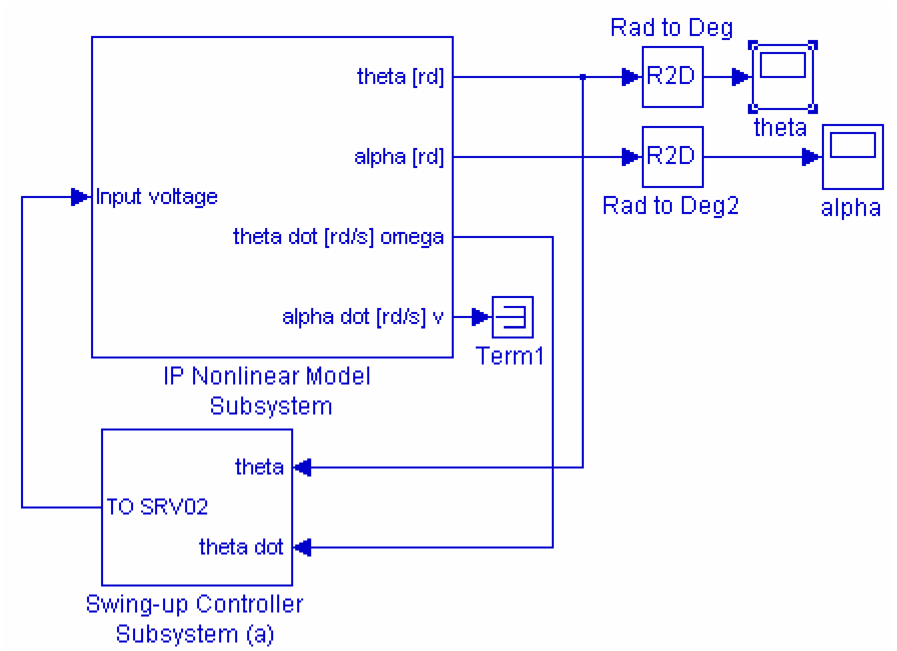

Figure 3.12 Simulink simulation diagram for Swing-up controller design (a).

The pendulum angle α should swing within a bound of 0 and -360° with increasing amplitude. Due to the nonlinearity and the assumption made in designing the Swing-up controller, in the implementation diagram, the Gain K_D may have to be further tuned so that the pendulum is raised to the upright position within $\alpha = \pm 5^{*}$ in a controlled manner.

(b) Swing-up with positive feedback

The desired position $\theta_i(t)$ is determined by the following control law

$$\theta_i(s) = P\alpha(s) + D\upsilon(s) \qquad (3.30)$$

This positive feedback based on the pendulum angle and its velocity creates a trajectory with growing amplitude. The gains P and D are adjusted to bring up the pendulum smoothly. Experimentally obtained suggested values are $P = 0.02$, and $D = 0.015$

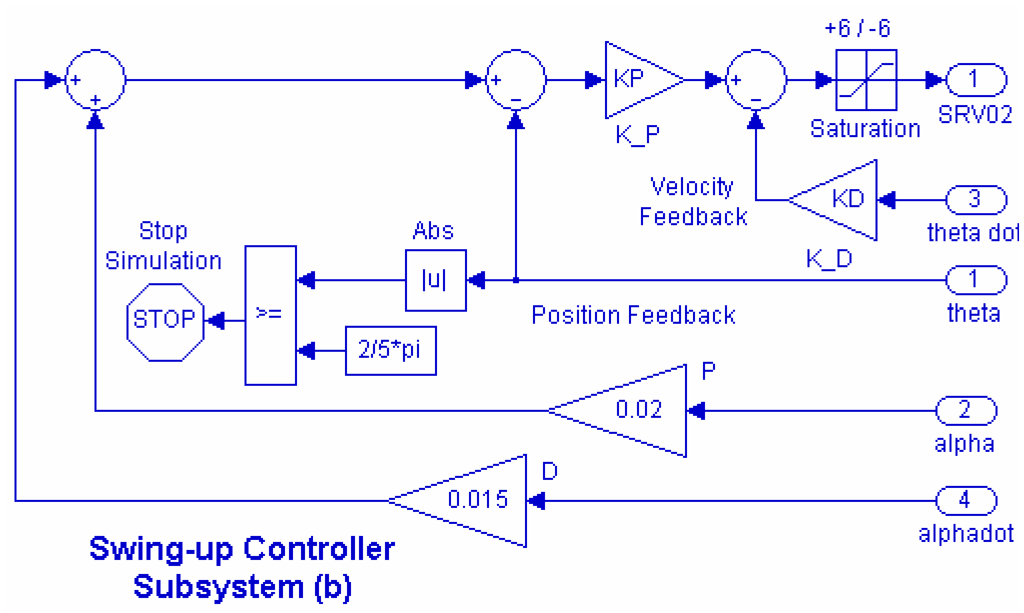

Figure 3.13 Simulink simulation diagram for Swing-up controller design (b).

You may need to adjust these values to obtain a smoothly growing response.

Connect the Swing-up Controller Subsystem (b) to the IP Nonlinear model similar to Figure 3.12. In the nonlinear model, open the Integrator 4 and set the initial angle (alpha) to $-\pi$ Radians (-pi). Obtain the response for θ_o, and α .

(c) Swing-up with positive feedback and adaptive rate gain

A better approach would be to design an adaptive rate gain D that changes with the states of the pendulum. One suitable strategy would be to detect the pendulum maximum and minimum swings α_{max}, $\alpha_{\min}$ and adjust D exponentially to bring up the pendulum more smoothly. The MinMax block from the Math Library, together with a Unit Delay block can be used to obtain the maximum and minimum of each swing as shown in Figure 3.14. The Unit Delay block delays and holds its input signal by one sampling interval. Open the unit Delay dialog box and set the initial condition to "–pi ", and the sample time the same as the simulation sampling time (i.e., 0.001).

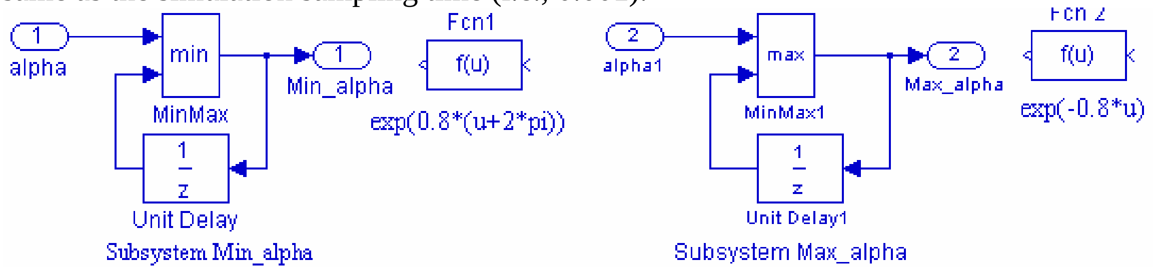

Figure 3.14 Simulink instructional blocks to hold the min and max values of each swing

The Function blocks FCn1 and Fcn2 with a Max block and a Gain block Y is used to multiply the feedback gain D by the $Ye^{a(\alpha_{\min}+2\pi)}$ or $Ye^{a(-\alpha_{\max})}$ as shown in Figure 3.15.

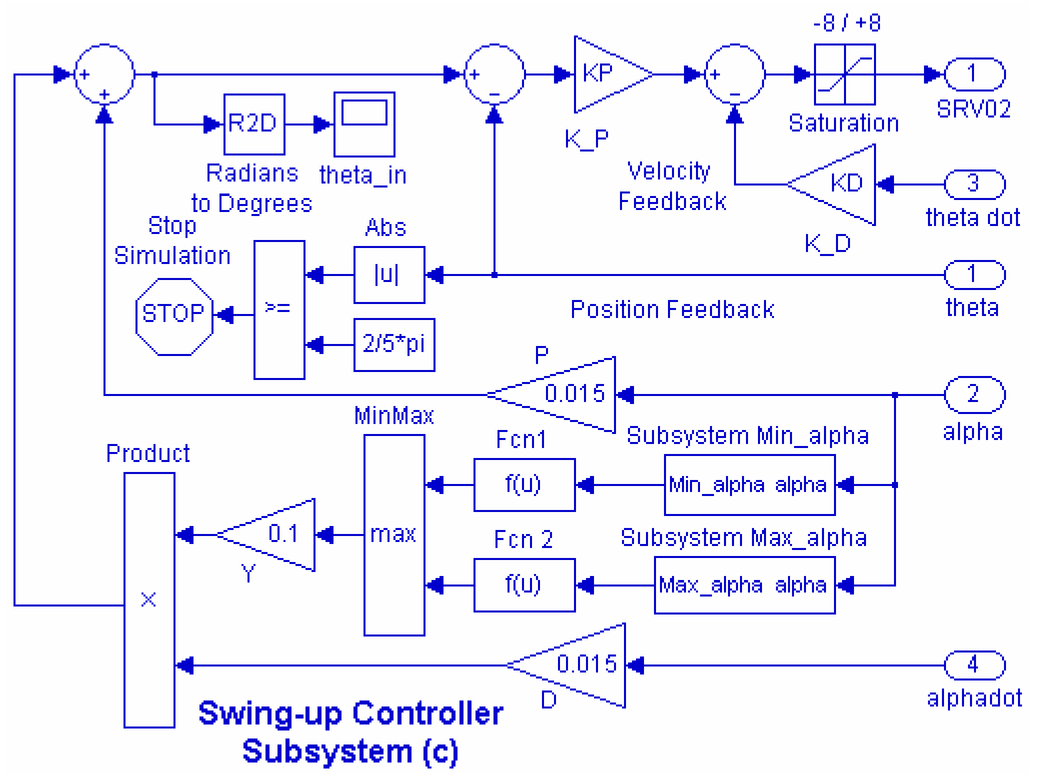

Figure 3.15 Simulink simulation diagram for Swing-up controller design (c).

The values for P, D, a and Y are obtained experimentally, suggested values are $P = 0.015$, and $D = 0.015$, $a = 0.8$ and $Y = 0.1$. You need to experiment with these values and perform a fine tuning to bring up the pendulum to the upright position with a small angular velocity.

Connect the Swing-up Controller Subsystem (c) to the IP Nonlinear model similar to Figure 3.12. In the nonlinear model, open the Integrator 4 and set the initial angle (alpha) to $-\pi$ Radians (-pi). Obtain the response for θ_o, and α.

Swing-up Controller Design

The servomotor equivalent inertia J_{eq} is replaced by $J_{eq} + 0.005$ to account for the pendulum inertia. Evaluating a_m, and b_m from (3.3). From (3.27) – (3.29) evaluate the gains for K_P and K_D.

Construct the simulation diagram in Figure 3.12 and obtain the swing-up simulation results for θ and α for each of three proposed swing-up controllers.

4. Laboratory Procedure

When you have finished testing your model in SIMULINK, it has to be prepared for implementation on the real-time hardware. This means the plant model has to be replaced by the I/O components that form the interfaces to the real plant.

4.1 Creating the Implementation model

Creating the Subsystem Block for the Interface to SRV02 and IP module

Two encoders are used to measure the angular position of the servomotor, θ and the pendulum position α.

If we turn the pendulum stick several times, the encoder will measure multiple revolutions. In order to make measured signal α to roll over after a full turn (modulus of 2π), we can use the Math Function **mod** from the Math Operations library. The command $\operatorname{mod}(\alpha,\ 2\pi)$ results in $\alpha - (n)(2\pi)$, where n is the remainder (integer) of $\dfrac{\alpha}{2\pi}$

For Example the following Simulink instruction with α_i increasing from 0 to 8π produces the output α_o that is always between 0 to 2π as shown in Figure 3.16.

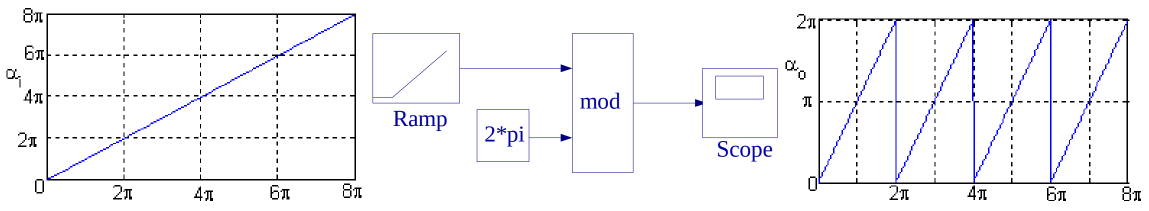

Figure 3.16 An Example of use of Modulus function

We must use two observers to reconstruct the state variables ω, and υ which are not directly measured. These are the derivatives of θ, and α. Remembering that we cannot implement a pure derivative, we use the low-pass band-limited observer $\frac{50s}{s+50}$.This filter is tuned to reduce the noise introduced in the numerical derivative.

Open a Simulink page and get the part Encoder Input. Double-click on the Encoder input block to open its dialog box and set the Channel Use to 0. Since the encoder has 4096 signal periods per revolution, get a gain block and set its value to $-2*pi/4096$. A negative sign is used for the encoder gain, because the negative feedback gain is already implemented in the encoder wiring (i.e. positive voltage $\Rightarrow$ negative counts). Get a Transfer function block for the low-pass band-limited filter. Get another Encoder Input for the pendulum arm encoder with the same gain block as before to convert the encoder count to radian. Add a Math Function from Math Operations library open the block and set the function to **mod.** Apply a Constant block with a value of 2π as its second input. Since at start the pendulum is hanging down, use a summing point to add the initial angle of $-\pi$. Add another low-pass band-limited differentiator to obtain the derivative of α as shown in Figure 3.17.

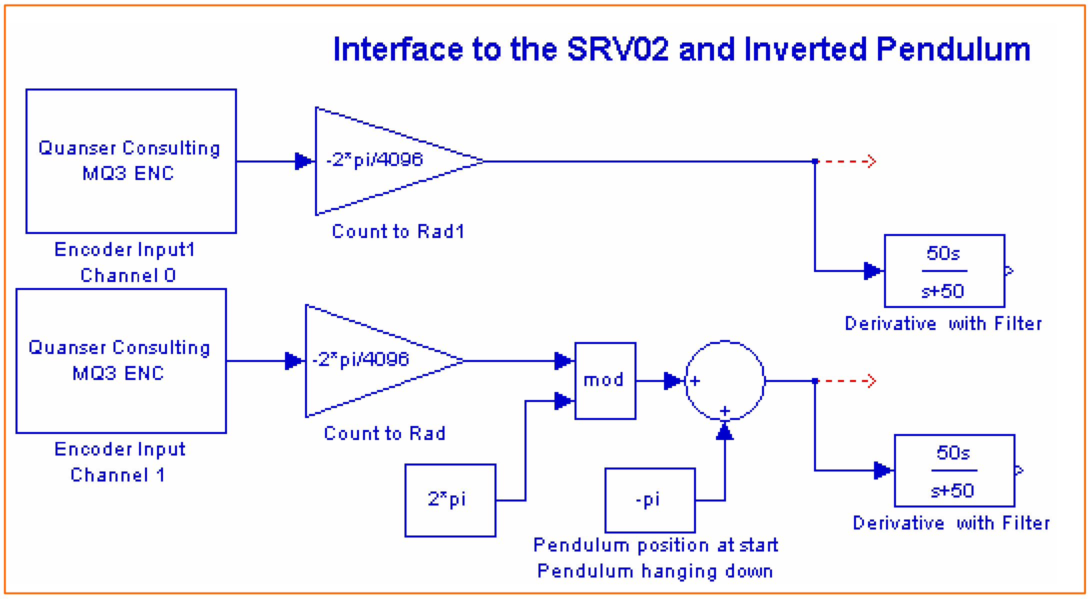

Figure 3.17 Building the Interface

To create a subsystem, enclose the blocks within a bounding box as shown in Figure 3.17 (Do not place any outport blocks). Choose **Create Subsystem** from the **Edit menu**. Simulink replaces the selected blocks with a subsystem block. Double-click to open the subsystem block. Notice that the Simulink automatically adds Outport blocks. If you have added the dashed lines as shown in Figure 3.17, outports would also be added at these locations, and then you can erase the extra dashed lines. If you did not have the dashed lines in Figure 3.17, the subsystem would create only two Outport terminals and you need to open the subsystem and add the Output ports from the Sink Menu manually. Label the Outports appropriately as shown in Figure 3.18. Rename and title subsystem to Interface to Srvo2 & IP, and save it as IP_Interface.mdl. You know have the following subsystem as shown in Figure 3.18.

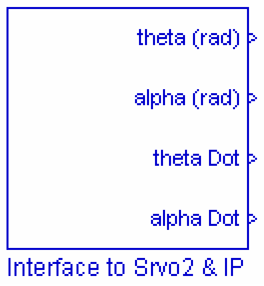

Figure 3.18 Subsystem block for the interface to the SRV02

If you double-click on the above subsystem it would display the underlying system as shown in Figure 3.19.

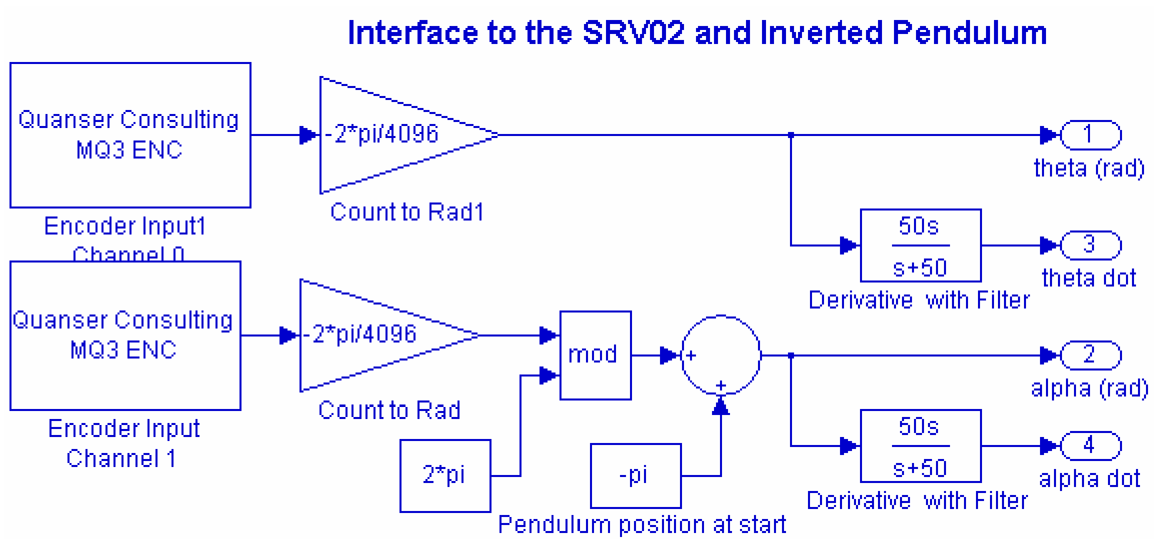

Figure 3.19 Interface to SRV02 and Inverted Pendulum

Catch Controller

The control instruction for the Catch Controller Subsystem is shown in Figure 3.20. Its purpose is to track the pendulum angle α , and facilitate switching between the swing-up and stabilization modes. This controller is to be enabled when α is in the neighborhood of zero, within $\pm 5^\circ$ and for as long as $|\alpha| < 25^\circ$.

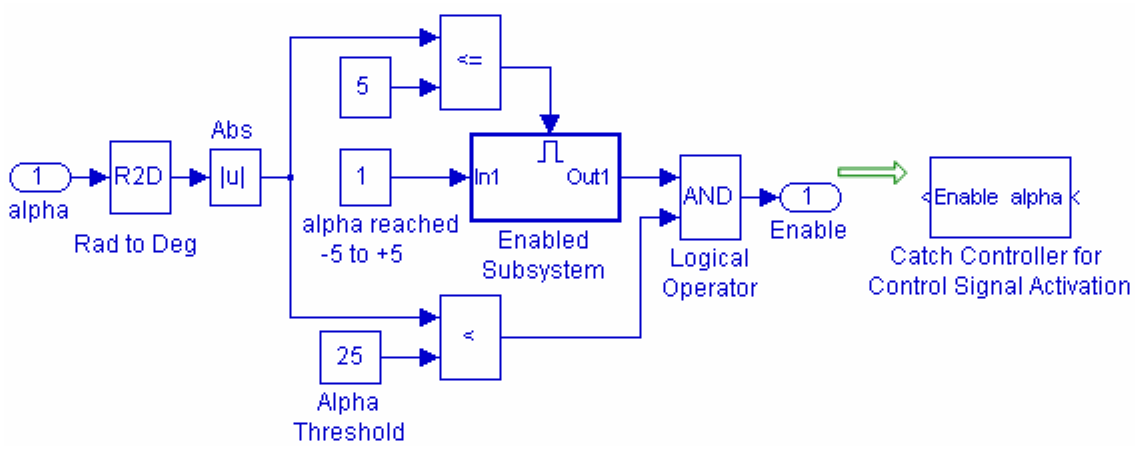

Figure 3.20 Catch Controller for control signal activation

To make this controller, get the Enabled Subsystem (from Port & Subsystem library) and add a constant input of 1 at it's inport. Get a R2D block and the Absolute block and connect to the Enable Subsystem block via a Relational Operator <= and constant block of value 5. This will enable the Subsystem when $-5^{\circ} \leq \alpha \leq 5^{\circ}$. Use a Relational Operator block < to test the absolute value of α in degrees against the threshold angle of 25°. Finally use a Logical Operator AND to enable the Catch Controller when the given constraints are met. Make the Subsystem and label it "Catch Controller for Control Signal Activation" When expanded it should be the same as shown in Figure 3.20.

Switch for activating the stabilization controller

This switch is triggered by the output of the Catch controller. Initially when the pendulum is hanging down, the Switch terminal 2 (See Figure 3.21) is at zero state and the control signal is passed through terminal 3 (lower terminal). If the pendulum is brought to the upright position (either manually or by means of the swing-up controller) as α reaches in the neighborhood of $\pm 5^{\circ}$, the Catch controller is enabled. The Switch control input state becomes 1, the signal is passed through terminal 1 (top terminal) and stabilization controller takes over.

Manual Switch

The manual switch is provided to allow the experiment to be performed with manual swing-up or self-erecting mode, using an appropriate Swing-up controller.

Implementation model

Start constructing the implementation diagram. Copy the IP_Interface.mdl (constructed in part 4.1, Figure 3.18) to the clipboard and paste it on a new Simulink page as shown in Figure 3.21. Get the Analog Output block from the Quanser MultiQ3 library and set the Channel Use to 0. Connect the output terminals of the Interface to Srvo2 & IP via a Mux block to a Matrix Gain block. Connect the output of the Gain Matrix block via a Switch (from Signal Routing library) to the Quanser Analog output.

Copy and paste the Catch Controller subsystem and connect its input to the pendulum angle signal and its output to the Switch control input (terminal 2). Next place one of the Swing-up Controller subsystems constructed in part 3.5 (Figure 3.11, Figure 3.13, or Figure 3.15), and connect it to the lower terminal of the Main Switch via a manual Switch. Connect the Swing-up controller input terminals to the appropriate state variables. When the Manual switch is at the top terminal, the pendulum is in the manual mode and when at the lower terminal, the pendulum is in the self-erecting mode. Place as many Scopes as you like to monitor the phase angles θ and α. Your completed model should be the same as shown in Figure 3.21.

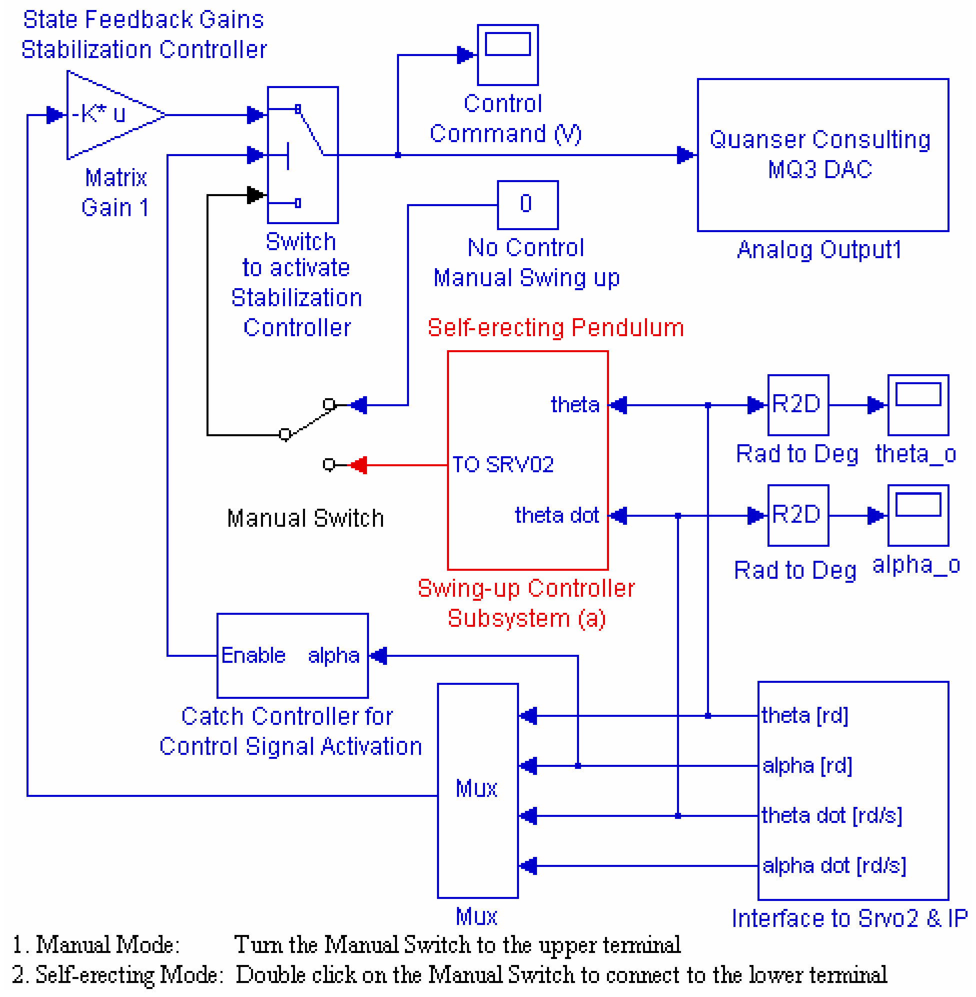

Figure 3.21 Implementation Diagram for the Rotary Inverted Pendulum

The Main switch Threshold is set to 1. The middle terminal is the control input. If the control input is 1 the signal is passed through input 1 (top terminal), and if the control input is 0, signal is passed through input 3 (bottom terminal).

The switch control input is triggered by the signal (0 or 1) coming from the Catch Controller Subsystem.

4.2 Wiring diagram

Attach the Rotpen module to the SRV02 as shown in Figure 3.1. Tighten the two thumb screws very well. Clamp the SRV02 to the table corner so that it does not tip when the pendulum is self erecting

Using the set of leads, universal power module (UPM), SRV-02 DC-motor, and the connecting board of the MultiQ3 data acquisition board, complete the wiring diagram shown in Figure 3.22 as follows:

From	To	Cable
Encoder on SRV02	MultiQ/Encoder 0	5 pin Din to 5 pin Din
Encoder on the pendulum arm	MultiQ/Encoder 1	5 pin Din to 5 pin Din
Motor on SRV02	UPM/To Load	6Pin to 4 Pin Din, **Gain 1 Cable**
D/A #0 on MultiQ	UPM – From D/A	RCA to 5 Pin Din
A/D # 0, 1, 2, 3, on MultiQ	UPM- TO A/D	5 Din to 4xRCA

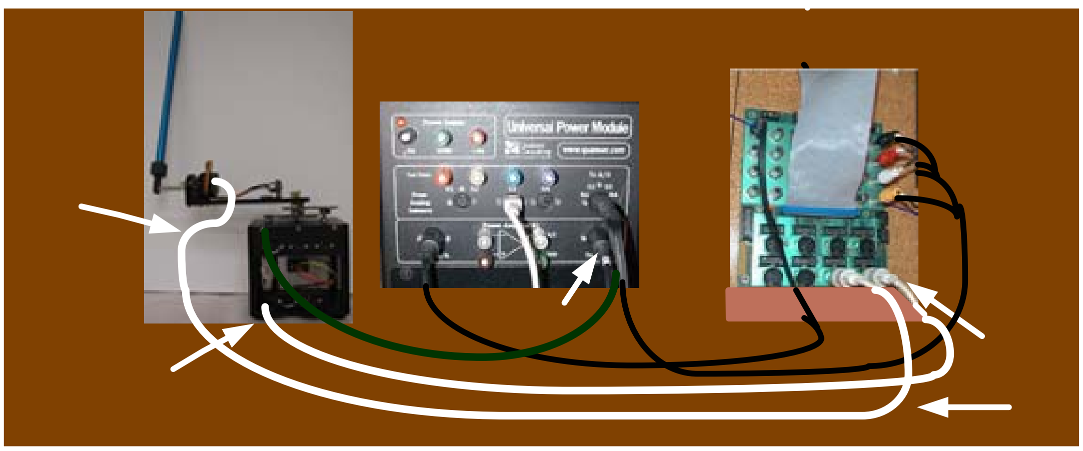

Figure 3.22 Wiring diagram for the IP Project.

4.3 Compiling the model

In order to run the implementation model in real-time, you must first build the code for it.

Turn on the UPM. Start WinCon, Click on the MATLAB icon in WinCon server. This launches MATLAB. In the Command menu set the Current Directory to the path where your model InvPen_Imp.mdl is. Before building the model, you must set the simulation parameters. Pull down the Simulation dialog box and select Parameters. Set the Start time to 0, the Stop time to 10, for Solver Option use Fixed-step and ode4 (Runge-Kutta) method, set the Fixed-step size, i.e., the sampling rate to 0.001. In the Simulation drop down menu set the model to **External**. Set the Matrix gains K to the values found in part (3.3), or run the m-file that returns the values of the matrix K.

Start the WinCon **Server** on your laptop and then use **Client Connect**, in the dialog box type the proper Client workstation IP address. Generate the real-time code corresponding to your diagram by selecting the "**Build**" option of the WinCon menu from the Simulink window. The MATLAB window displays the progress of the code generation task. Wait until the compilation is complete. The following message then appears: "*### Successful completion of Real-Time Workshop build procedure for model: InvPen_Imp*".

4.4 Running the code

Following the code generation, WinCon Server and WinCon Client are automatically started. The generated code is automatically downloaded to the Client and the system is ready to run. Set the Manual switch to its top position (No control Manual Swing-up). To start the controller to run in real-time, click on the **Start** icon from the WinCon Server window shown in Figure 3.23. It will turn red and display STOP. For the manual mode hold the tip of the pendulum and move the pendulum slowly upward towards $\alpha = 0$ until you feel the system is catching and stabilizing the pendulum in upright position.

Clicking on the **Stop** icon will stop the real-time code and return to the green button.

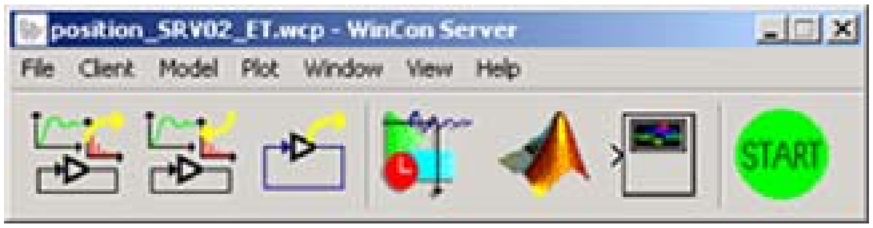
position_SRV02_ET.wcp - WinCon Server
File Client Model Plot Window View Help
START

Figure 3.23 WinCon Server

If the pendulum is going out of control immediately stop the controller. Also, if you hear a whining or buzzing in the motor you are feeding high frequency noise to the motor or motor is subjected to excessive voltage, immediately stop the motor. Ask the instructor to recheck the implementation diagram and the compensator gains before proceeding again.

4.5 Plotting Data

You can now plot in real-time any variables of your diagram by clicking on the "**Plot/New/Scope**" button in the WinCon Server window and selecting the variable you wish to visualize. Select "alpha_o" and click OK. This opens one real-time plot. From Scope pull-down menu, select Buffer and set the Buffer Size to 10. From the File menu you can Save and Print the graph. If you choose Save As M-File … you save the plot as

M-file. Now at the MATLAB prompt if you type the file name you can obtain the MATLAB Figure plot. You can type grid to place a grid on the graph or edit the Figure as you wish. Similarly obtain a plot for theta_o.

The inverted pendulum system allows the user to add disturbances, by tapping on the tip of the pendulum. The disturbance should maintain stability unless the disturbance causes the angle α to exceed 25 degrees in either direction. Tap the tip of pendulum gently to see how it reacts, and if stability is maintained for a small disturbance.

In WinCon Server, use File Save, this saves the compiled controller including all plots as a **.wpc** (WinCon project) file. In case you want to run the experiment again, from WinCon Server use File/Open to reload this **.wcp** file, and run the project in real time independent of MATLAB/Simulink.

To prevent excessive wear to the motor and gearbox run the experiment for a short time.

Self-Erecting Pendulum

Make sure the controller is not running WinCon Server window should display green START. Double-click on the manual switch to connect to the lower position (Self-erecting Swing-up Controller). Click on the **Start** icon from the WinCon Server window it will turn red and display STOP. The pendulum will start swinging, it should come up to the upright position in the first few swings where the Catch Control will activate the State feedback Controller and stabilizes the pendulum. If the pendulum is not catching up, stop and try again. Always start the system with the servomotor arm in the middle ($\theta = 0$), and the pendulum in at standstill in down position. If the pendulum does not quite come to the upright position you may have to do fine tuning for K_D . Obtain a plot of θ, and α , which should be the same as before when the pendulum is under the state feedback stabilization controller. Tap the tip of pendulum gently to see how it reacts, and if stability is maintained for a small disturbance. If the tap is very hard that pushes θ beyond $\pm 25^\circ$ the catch controller is disabled, the Switch state becomes zero, and the pendulum falls and the pendulum will start oscillating again until the pendulum comes up and switch state once again becomes 1. In some situation if the tap is so hard that pushes θ beyond $\pm 2\pi / 5$ (i.e., $\pm 72^\circ$) the Swing-up controller stops the run to prevent the full rotation of the servomotor.

Alternative Swing-up controller
Further independent study or projects for design of the Swing-up controller are as follows.

- Controller design using an S-curve trajectory.
- Designing a Pulse-Width Modulation (PWM), with a variable duty cycle signal.
- Creating a trajectory based on pendulum energy control with minimum time strategy.

- Design of adaptive fuzzy logic controllers.
- Design of neural fuzzy logic controllers with adaptive membership functions.
- Study of the robustness of these techniques.

5. Project Report

Discuss the assumptions and approximations made in modeling the servomotor and the inverted pendulum.

Your report must include the detailed servo plant and the Inverted pendulum block diagrams. Discuss and compare the nonlinear model with the linearized model, and determine the range of α for which the linear model is in close agreement with the nonlinear model. Summarize the state feedback gains and comment on the simulation results due to a step input in part 3.4. Discuss how the implemented stabilization controller performed when the pendulum was brought to the upright position manually. Comment on the system behavior when the pendulum was subjected to external disturbances. State the type of Swing-up controller used and state if it produced a satisfactory start up. If you used more that one Swing-up controller compare their performance and actual positive feedback gains used in the Swing-up controller (b) and (c). Discuss any modification you made in the Swing-up controller that significantly improves the performance of the Swing-up controller. That is, to bring the pendulum to the upright position with minimum number of swings and a very small velocity in the upright position

.