Chapter 2

Background: Robust Adaptive Observer Estimators

Chapter 2
Background: Robust Adaptive Observer Estimators

A device that estimates or observes state variables of a system is called a **state observer**. A state observer utilizes measurements of the system inputs and outputs and a model of the system based on differential or difference equations.

Three main classical observers are: Luenberger observer, adaptive observer and Kalman filter. For deterministic systems without random noise the Luenberger observer and its extensions (Luenberger 1971; Bhattacharyya 1976; Fairman and Gupta 1980; O'Reilley 1983) may be used for time-invariant systems with known parameters. The equation for the Luenberger observer contains error correction terms, which ensures stability and convergence of the observer even when the system being observed is unstable. When the parameters of the system are unknown or time varying, an adaptive observer may be used, which in addition to estimating the system states, estimates the system parameters. Achieving this added requirement, while maintaining stability, has resulted in the development of significantly complex observer structure. Because prediction error can no longer be unambiguously associated with errors in estimating state, a *persistently exciting signal* must be generate to insure the stability of the adaptive observer. Even with this persistently exciting plant signal, the adaptive observer has significant difficulty distinguishing between the effects of inaccurate parameter estimates and measurement disturbances.

Disturbances can be either stochastic or deterministic. While stationary stochastic input processes with a zero-mean Gaussian distribution can be effectively rejected by a Kalman filter when accurate noise statistics are available, fixed non-stochastic disturbances can only be rejected when the observer is augmented with a dynamic model of the disturbance (Lin 1994). Time varying disturbances of either type that can not be modeled as a linear system are difficult if not impossible to reject. Any ability to compensate for either stochastic or deterministic disturbances is called **disturbance rejection**.

Rejection of a single disturbance can be achieved in a straight forward manner with a single observer when the system has *known* parameters. No known disturbance rejection methodologies exist for adaptive observers. Even with a persistently exciting plant input the adaptive observer has difficult distinguishing between the effect of inaccurate parameter estimates and disturbances. Additional difficulties exist when the adaptive observer is utilized in a closed loop regulator because the controller suppresses the persistently exciting input signal needed for the convergence of parameter estimates.

The two main applications of observers are observer-based state feedback control and fault detection. Both applications rely on accurate state estimation and suffer from performance degradation when input disturbances corrupt the observer's state estimates.

Classical observers are partitioned into deterministic and stochastic variants. Deterministic observers require relatively noise free measurements of a system's input and output while stochastic observers can model additive measurement noise and process noise and provide the linear least means square estimates of state

2.1 Deterministic

Deterministic observers for time invariant plants with known parameters are commonly referred to as Luenberger Observers (Luenberger 1971) while observers for plants with unknown parameters are known simply as adaptive observers.

• ***Luenberger Observer***

The Luenberger observer estimates the state variables of linear time-invariant systems with known parameters. Consider a dynamic system of order n described by the following equations

$$\begin{aligned} \dot{x} &= Ax + Bu \\ y &= Cx \end{aligned} \qquad 2.1$$

where $u(t)$ are the p inputs and $y(t)$ the m outputs. A, B and C are $(n \times n)$, $(n \times p)$ and $(m \times n)$ matrices respectively. A Luenberger observer is described by

$$\dot{\hat{x}} = A\hat{x} + L(y - C\hat{x}) + Bu \qquad 2.2$$

where $\hat{x}$ is the estimates of the state x. If the system (**A**, **C**) is observable then the constant **L** can be selected so that (**A** - **LC**) is asymptotically stable and $\hat{x}(t)$ asymptotically approaches x(t). The $L(y - C\hat{x})$ term provides a provides a proportional correction factor that insures stability of the observer even when the system (**A**, **B**) is unstable. A schematic of the observer in this open-loop configuration is shown in Fig 2.1.

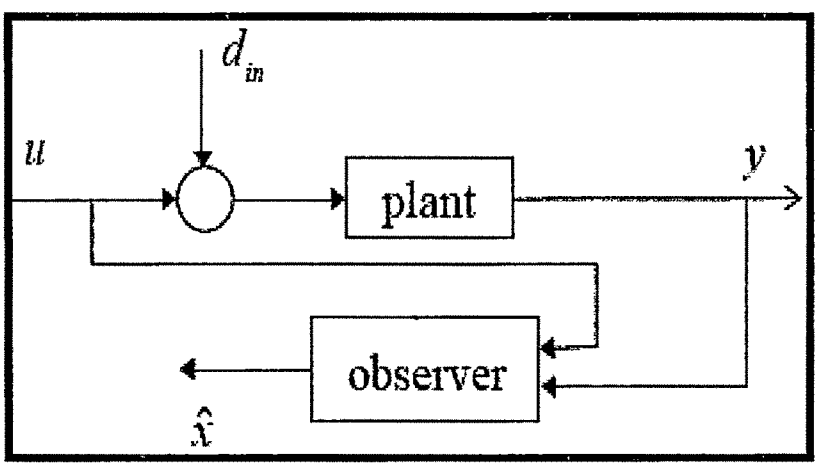


Fig 2.1 Observer Schematics

• ***Robust PI Observer***

The PI observer includes an additional term, the integral of the estimation error, v, in the observer equation

$$\dot{\hat{x}} = A\hat{x} + L(y - \hat{y}) + B(u + v)$$
$$\dot{v} = K_I(C\hat{x} - y) \qquad 2.3$$

where $\mathbf{K_I}$ is a $(n \times n)$ matrix and is selected so the matrix **R**,

$$R = \begin{bmatrix} A - LC & B \\ K_I C & 0 \end{bmatrix} \qquad 2.4$$

is asymptotically stable.

• ***Adaptive Observer***

When the plant has unknown parameters, an observer must be adaptive and capable of estimating the system parameters **A**, **B** and **C**. Parameters estimates are only required for those parameters that are known. Estimation of these system parameters is aided by associating the parameters with the measurements $\mathbf{y}(t)$ and $\boldsymbol{u}(t)$ (Carroll and Lindorff 1973; Kudva and Narendra 1973). This is accomplished by first transforming the system's unknown and known parameters to left companion observable canonical form

$$\dot{x} = \left[-a \middle| \overline{A}\right]x + bu$$
$$y = cx = x_1 \qquad 2.5$$

Where $\mathbf{a}^T = [\, a_1, a_2, \ldots . a_n]$, $\mathbf{b}^T = [\, b_1, b_2, \ldots . b_n]$, $\mathbf{c}^T = [\, 1, 0, \ldots . 0]$ and $\overline{A} = \begin{bmatrix} I_{n-1} \\ 0 \end{bmatrix}$ with I_{n-1} representing an $((n-1) \times (n-1))$ identity matrix. Now only the n parameters in $\mathbf{a}(t)$ are needed to specify $\mathbf{A}(t)$ and the n parameters in $\mathbf{b}(t)$ are needed to specify $(\mathbf{B}(t), \mathbf{C}(t))$. Introduction of the constant **k**, a $(n \times 1)$ vector, allows further algebraic manipulation of equations so that the system parameters $\mathbf{a}(t)$ and $\mathbf{b}(t)$ are associated with the measurements $y(t)$ and $u(t)$ respectively:

$$\dot{x} = Kx + [k - a(t)]y(t) + b(t)u(t)$$
$$y = cx = x_1 \qquad 2.6$$

k is selected so the matrix $\mathbf{K} = [-\mathbf{k} | \overline{A}\,]$ is asymptotically stable. A P adaptive observer can now be described by

$$\dot{\hat{x}} = K\hat{x} + [k - \hat{a}(t)x_1(t) + \hat{b}(t)u(t) + w_1(t) + w_2(t) \qquad 2.7$$

where $\hat{a}(t)$ and $\hat{b}(t)$ are estimates of the parameters $\mathbf{a}(t)$ and $\mathbf{b}(t)$. $w_1(t)$ and $w_2(t)$ are n-dimensional signal that are required to insure stability of the observer[1]. The adaptation laws for the parameters $\mathbf{a}(t)$ and $\mathbf{b}(t)$ are

$$\dot{\hat{a}} = \Gamma_1(c\hat{x} - y)w_1$$
$$\dot{\hat{b}} = \Gamma_2(c\hat{x} - y)w_2 \qquad 2\,8$$

The constant matrices Γ_1 and Γ_2 are proportionality constants that affect the rate of convergence of the parameter estimates.

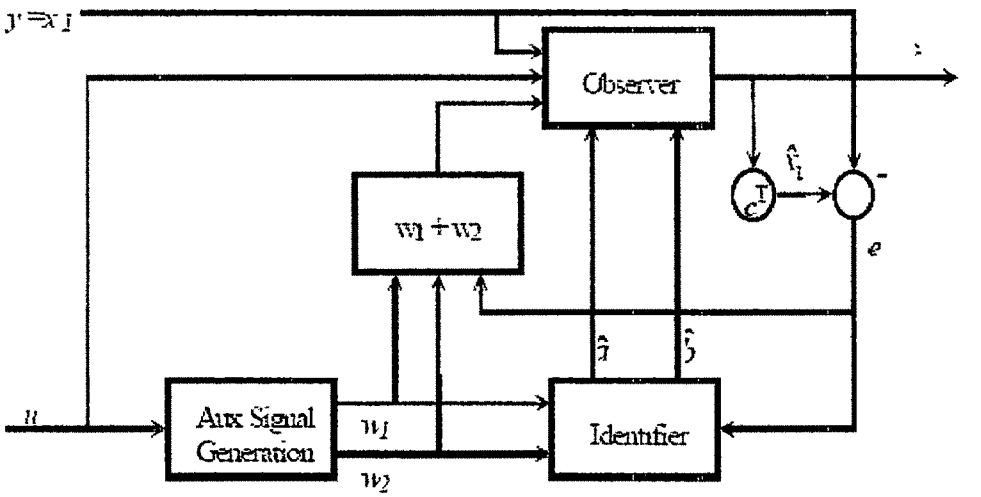


Fig 2.2 Adaptive Observer schematic

Fig 2.2 shows a schematic of the adaptive observer implementation Equation 2.7 is implemented by the Observer block, and Equation 2.8 is implemented by the Identifier block.

2.2 Stochastic

Stochastic observers are commonly referred to as Kalman filters (Kalman 1960). They structure is the same as simple linear observe with a gain optimized to solving the on-line the Linear Least Mean Square (LLMS) estimation problem. The filter dynamically calculates the prediction and estimation error covariance and at each filters iteration and calculates the optimal observer gain When the plant is time invariant and the random noise has a fixed variance the gain of the observer converges to a fixed value, the observer can then be calculated off-line.

- ***Kalman Filter***

Consider a dynamic system of order *n* described by the following discrete linear system model

$$\mathbf{x}(k+1) = \mathbf{A}x(k) + \mathbf{B}u(k) + \mathbf{v}(k)$$
$$\mathbf{y}(k) = \mathbf{C}x(k) + \mathbf{w}(k) \qquad 2.9$$

where: $\mathbf{v}(k)$ and $\mathbf{w}(k)$ are the process and measurement zero-mean Gaussian noise sequence with $\mathbf{Q}(k)$ and $\mathbf{R}(k)$ covariance matrices, respectively. The observer equation for the corresponding Kalman filter (Santina, Stubberud et al. 1994) is

$$\hat{x}_{k|k} = A\hat{x}_{k|k-1} + K_k\left[y_k - C\hat{x}_{k|k-1}\right] + Bu(k) \qquad 2.10$$

where the optimal Kalman Gain $\mathbf{K}_k$ is given by

$$K_k = P_{k|k-1}C_k(C_kP_{k|k-1}C_k^{'} + R_k)^{-1} \qquad 2.11$$

Kalman gain is defined in terms of the state prediction covariance matrix $P_{k|k-1}$. $P_{k|k-1}$:

$$P_{k|k-1} = A(k-1)P_{k-1|k-1}A(k-1) + Q(k-1) \qquad 2.12$$

which is given in terms of the state estimation error covariance at the previous filter iteration, $\mathbf{P}_{k-1|k-1}$

$$P_{k-1|k-1} = \left[I - K(k-1)C(k-1)\right]P_{k-1|k-2} \qquad 2.13$$

The estimation error covariance definition is recursive because of the $\mathbf{P}_{k-1|k-2}$ term. If however the noise covariances and the system do not vary with time a non-recursive solution for $\mathbf{K}$ can be obtained by solving the following algebraic Riccatti equation for the state prediction covariance matrix $\mathbf{P}_{k|k-1}$

$$P_{k|k-1} = AP_{k|k-1}F' - AP_{k|k-1}C'\left[CP_{k|k-1}C' + R\right]^{-1}CP_{k|k-1}C' + Q \qquad 2.14$$

The solution of this equation gives Kalman gain.

2.3 Observer-based control

State feedback controllers designed as **Linear Quadratic Regulators (LQR)** have impressive robustness properties (Anderson and Moore 1989), including the rejection of disturbances from unknown inputs, actuator faults and plant perturbations. A LQR however requires access to system state variables. When state variables are unavailable a state observer must be included in the feedback loop, but this drastically reduces the robustness of state feedback control. The **Linear Quadratic Gaussian/ Loop Transfer Recovery (LQG/LTR)** techniques were introduced to recover some of the robustness properties of LQR, but these techniques often do not achieve full loop transfer recovery, and do not have disturbance rejection properties of the original LQR.

The PI observer, can be used to enhance the LTR procedure; the integral action of the PI observer allows additional freedom in adjusting the controller. A PI observer based controller can be designed with frequency and time recovery properties.

An equivalent loop transfer recovery procedure, however, does not exist for the adaptive observer-based controller. Therefore, LTR techniques can not be used to recover the disturbance rejection properties of the original LQR. Instead, the adaptive control community has developed Model Reference Adaptive Control (MRAC) for linear systems with unknown parameters (Narendra and Lin 1980; Kaufmann, Bar-Kana et al. 1994). MRAC does not use state information in the controller implementation, but rather relies only on measurement of the input and output of the system and a separate reference model to adapt the control gain. While MRAC can adapt to known disturbances (Kaufmann, Bar-Kana et al. 1994) it does not provide estimate of the disturbance or states.

2.4 Model Uncertainty and Robust Control

A key reason for using feedback is to reduce the effects of uncertainty which may appear in different forms as disturbances or as other imperfections in the models used to design the feedback law. Model uncertainty and robustness have been a central theme in the development of the field of automatic control.

A central problem in the early development of automatic control was to construct feedback amplifiers whose properties remain constant in spite of variations in supply voltage and component variations. This problem was the key for the telephone industry that emerged in the 1920s. The problem was solved by [1]. We quote from his paper:

" .. by building an amplifier whose gain is deliberately made say 40 decibels higher than necessary (10 000 fold excess on energy basis) and then feeding the output back on the input in such a way as to throw away excess gain, it has been found possible to effect extraordinarily improvement in constancy of amplification and freedom from nonlinearity."

Blacks invention had a tremendous impact and it inspired much theoretical work. This was required both for understanding and for development of design method. A novel approach to stability was developed in [2], fundamental limitations were explored by [3] who also developed methods for designing feedback amplifiers, see [4]. A systematic approach to design controllers that were robust to gain variations were also developed by Bode.

The work on feedback amplifiers became a central part of the theory of servomechanisms that appeared in the 1940s, [5], [6]. Systems were then described using transfer functions or frequency responses. It was very natural to capture uncertainty in terms of deviations of the frequency responses. A number of measures such as amplitude and phase margins and maximum sensitivities were also introduced to describe robustness. Design tools such as the Bode diagram introduced to design feedback amplifiers also found good use in design of servomechanisms. Bode's work on robust design was generalized to deal with arbitrary variations in the process by Horowitz [7]. The design techniques used were largely graphical.

The state-space theory that appeared in the 1960s represented significant paradigm shift. Systems were now described using differential equations. There was a very vigorous development that gave new insight, new concepts [8], [9] and new design methods. Control design problems were formulated as optimization problems which gave effective design methods, [10], [11], and [12]. Control of linear systems with Gaussian disturbances and quadratic criteria, the LQG problem, was particularly attractive because it admitted analytical solutions [13], [14], and [15]. The design computations were also improved because it was possible to capitalize on advances in numerical linear algebra and efficient software. The controller obtained from LQG theory also had a very interesting structure. It was a composition of a Kalman filter and a state feedback.

The state-space theory became the predominant approach, [16]. Safonov and Athans [17] showed that the phase margin is at least 60Æ and the amplitude margin is infinite for an LQG problem where all state variables are measured.

The paper [18] represented a paradigm shift which brought robustness to the forefront. It started a new development that led to the so called H_∞ theory. The idea was to develop systematic design methods that were guaranteed to give stable closed loop systems for systems with model uncertainty. The original work was based on frequency responses and interpolation theory which led to compensators of high order. Game theory is another approach to H_∞ theory. The H_∞ theory is well described in books [19], [20] and [21].

Major advances in robust design were made in the book [22] where the H_∞ control problem was regarded as a loop shaping problem. This gave effective design methods and it also reestablished the links with classical control. This line of research has been continued by [23] who has obtained definite results relating modeling errors and robust control.

2.4.1 Robust Adaptive Control of Nonaffine Nonlinear Plants

Most adaptive control systems assume the nonlinear plant: 1) is affine with respect to the input; 2) has *accessible* states [24], [25]; 3) is *feedback linearizable*; 4) has a known *well defined relative degree* [25]; and 5) satisfies the *minimum phase* condition ([26, pp. 11, 12]; [27, p. 16]). The first two assumptions are used to simplify the analysis, while the third imposes well known restrictive conditions ([3, pp. 39–42], [27, p. 16], [28, p. 438–439]) to transform the plant dynamics from nonlinear to linear via a change of coordinates and feedback. The last two assumptions are needed to guarantee closed-loop stability when an analytic output or state feedback controller is used for exact (or asymptotic) tracking of an arbitrary reference signal [29], [30].

Unlike affine plants, there are relatively few adaptive control techniques for nonaffine plants. As a result control system designers are often forced to approximate plant dynamics with linear models (such as ARMA models [25]) or make mathematical approximations to solve control-design equations derived from nonaffine system equations [24]. In either case, an assumption of "sufficiently small input magnitudes" is

made in addition to some of the assumptions cited above. Recently, situations have arisen where required inputs surpass the small magnitude restriction and one or more of the assumptions listed above [24]. Such situations often arise when new and more sophisticated systems are built and/or higher performance is needed from existing plants. If states are not accessible and inputs surpass the standard magnitude restriction, affine approximate input–output models such as NARMA-L1 and NARMA-L2 models, can be used to identify nonaffine plants in adaptive control schemes [24]. Though NARMA-L1 and NARMA-L2 models (and adaptive control systems derived from them) allow larger inputs than ARMA models, the allowable inputs may not be big enough for some applications. Furthermore, the maximum permissible input magnitude is not easily determined.

An affine approximate model, where input magnitude restrictions are replaced by a restriction on input changes [31], [32], was developed by linearizing the NARMA[25], [33], [34] model with respect to the last input instead of a stationary point as is typically done. Such a model is suitable for applications that require large inputs. Using this model, we developed a tracking control law for nonaffine plants models. This control law works for discrete nonaffine plants including: 1) minimum phase; 2) nonminimum phase; 3) feedback linearizable; and 4) nonfeedback linearizable plants that require large input magnitudes. Unlike most controllers, the control signal of this controller is not an analytic function of the plant output and the control-signal changes are restricted to small values. Restricting the control signal changes to small values limits the controller to applications where the plant output is required to track slowly varying reference signals.

The NARMA model may be approximated online with the affine model of [30], [31] and realized with a neural network. Convergence of the neural network weights is guaranteed by a deadzone approach to prevent instability due to modeling errors. Closed-loop stability is proved by showing all signals are bounded and neural network weights converge to finite values. A simulation example is provided for illustration.

2.5 H_∞: Preview Control

Much work has focused on the H_∞ feedback control problem, since the landmark state-space solution was presented [35]. A controller that optimizes the H_∞ norm was found to exist if and only if the stabilizing solutions of two algebraic Riccati equations satisfy three inequality conditions (see [35]). Although it is generally accepted that feedforward control can greatly enhance performance, most work in the H_∞ control focuses on the use of single-degree-of-freedom (SDOF), i.e., feedback only, controller design. In several earlier works, [36], [37], [38], and [39], the two-degree of freedom (2DOF) H_∞ control problems were investigated. Most of these formulations often increase the order of the controller structure, which is a major drawback.

It has been shown that preview control can improve performance when future information about the desired output or exogenous disturbance is available. The LQ based preview control formulation is well established. In an early derivation, [39], the optimal preview control signal was found to consist of three control terms: one feedback

and two feed forward terms. These two feed forward terms consist of the preview signal inside the preview window (a convolution term) and outside the preview window (a "kick" term). If the kick term is neglected [40], results into a simpler preview control algorithm. These LQ based preview control algorithms have been applied to a wide range of applications [41], [42]).

Preview control algorithms based on the H_∞ norm have gained increased interest recently. [43] and [44], a derivation is proposed based on game theories. The Riccati Equation is then modified accordingly and the feed forward control law is assumed unchanged from the LQ-preview control formulation. [45] describes H_∞ preview control formulation based on stored disturbances on finite-dimensional operations. The stored disturbances describes the perturbations of the preview control systems and allows for the derivation of the control law. A Hamiltonian based formulation [46] developed to allow for the simultaneous design of feedback, feed forward and preview control components. The approach of [46] may be generalized to a framework suitable for the continuous and discrete-time, LQ, H_∞ tracking and regulation problems.

2.5.1 H_∞: Optimal Preview Controller

It is doubtless that H_∞ optimization is one of the most important fruits of automatic control theory. The method for deriving the (sub-) optimal solution to the problem is well known as far as causal control scheme is concerned. However, there are some systems that we can measure the future value of exogenous disturbances, such as active noise reduction systems. If we consider H_∞ optimization problem for those systems, we can obtain better controller than the cases when the future value cannot be measured. We consider this problem for continuous- time systems in this note.

There are two approaches proposed in existing literatures. One approach is studied by Shaked *et al.* [51], [52]. They use game theory and a saddle point, as is done for causal H_∞ optimization problem by Green *et al.* [57]. The other approach is proposed by Kojima *et al.* [47]–[50]. They reformulated the problem in infinite-dimensional setting, and obtain the optimal solution by using functional-analytic technique.

These two approaches are essentially based on state-space method. [16] is based on transfer function method, as in [53]. Bilateral Laplace transformation [56] plays a key role in our argument. In continuous-time system theory, preview (or delay) element is essentially infinite-dimensional system. However, by our method, H∞-optimal preview problem is reduced to a causal and finite-dimensional H_∞ optimization problem, whose solution is well known. No infinite-dimensional technique is required in this note.

Furthermore, the relationship between preview horizon and optimal H_∞ performance is discussed in this note. Obviously, the optimal value of H_∞ norm is a nonincreasing function of preview horizon. However, it does not necessarily decrease monotonously. As pointed out in [47], there are some cases that the optimal H_∞ norm reaches its minimum value for some finite preview horizon.

2.5.2 Stochastic H_2/H_∞ Control With State-Dependent Noise

Mixed H_2/H_∞ control problem for deterministic systems has become a popular research topic in recent years. It has attracted much attention and has been widely applied to various fields; [64]–[69] and the references therein. In recent years, some researchers have turned their attentions to the stochastic H_∞ control problem. For example, on the systems governed by It's stochastic differential equation, a class of very general linear H_∞ stochastic problem with state- and control- dependent noise was studied by [70], and a stochastic bounded real lemma was derived, which has important applications in robust H_∞ stochastic filtering (see, e.g., [71]). Reference [72] treated with the robust control in the presence of stochastic uncertainty. [73] is on H_2 and H_∞-control (H_2 and H_∞ filtering) for Markov jump linear systems, and [74] on H_∞ control (filtering) for discrete-time stochastic systems.

H_∞ control is an important robust control design for eliminating efficiently the effect of disturbance v (t), and has been widely employed to deal with robust performance control problem with uncertain disturbance. Obviously, there may be more than one solution to H_∞ control problem with a desired robustness. In engineering practice, we want the control u(t) not only to eliminate the effect of disturbance, but also to minimize a desired control performance when the worst case disturbance $v^*(t, x)$is imposed. Since the H_2 performance is more appealing for control engineering, it naturally leads to the mixed H_2/H_∞ control problem [64], [67], [68]. If the solution $(u^*(t, x), v^*(t, x))$ of the above design exists, then we say the $H_2/H\infty$ problem has a pair of solutions $(u^*(t, x), v^*(t, x))$.

In deterministic H_∞ theory, H_∞ norm is defined by a norm of the rational transfer matrix, which cannot be directly generalized to nonlinear or stochastic systems [70], [75]. However, from the view of time-domain, a norm of the transfer function is the same as L_2 - induced norm of the input-output operator with initial state zero, this important feature makes it possible to develop the nonlinear or stochastic H_∞ control theory. In [70], the stochastic H_∞ norm is given by a norm of the perturbation operator L, which measures the worst case effect that the stochastic disturbance may have on the controlled output z.

One of the important approaches solving H_2/H_∞ problems belongs to the Nash game theory [64], [76] and [77]. By constructing two performances $J_1(u, v)$and $J_2(u, v)$ associated with H_∞ robustness and H_2 optimization, respectively, the control can be converted into finding the Nash equilibria point (u^*, v^*), such that [64]

$$J_1(u^*, v^*) \leq J_1(u^*, v) \quad J_2(u^*, v^*) \leq J_2(u, v^*).$$

In [64], by using a Nash game approach, the mixed H_2/H_∞ control problem of deterministic linear systems was solved, and the necessary and sufficient conditions were presented in terms of the existence of solutions of a cross-coupled Riccati equations. Because the results of [64], on the one hand, are very elegant in theory; and on the other hand, the cross-coupled Riccati equations may be solved by a standard numerical integration, it has become a popular paper in this area. The method used in [64] has been generalized to the nonlinear [78] and output feedback H_2/H_∞ control [79].

Up to now, few results have been obtained on the stochastic H_2/H_∞ problem with state- or control-dependent noise.[63]extends the results of [44] on the deterministic H_2/H_∞ control problem to the stochastic systems governed by Itô differential equations with state-dependent noise

In order to develop a parallel stochastic H_2/H_∞ theory to that of [64], two essential difficulties arise, i.e., how to extend [70, Lemma 4] and [67, Lemma 2.2] to stochastic systems. By utilizing a comparison theorem on the algebraic Riccati equation (ARE) [71] stochastic bounded real lemma [70], and the standard theory of differential equations, [63] overcome these two difficulties, and obtain two more general results. Based on which the infinite and finite horizon H_2/H_∞ stochastic state feedback control problems may be solved, respectively. When the state variables cannot be measured directly, how to design a stochastic controller based on the available information, is very valuable in practice, and has been studied in stochastic [70] and multiobjective H_2/H_∞ control [79]. By solving a two-step convex optimization problem, an observer-based suboptimal $H_2/H\infty$ dynamic output feedback control design is developed for the stochastic systems with uncertain disturbance.

2.6 BILINEAR SYSTEMS

A number of practical systems such as biochemical process nuclear fission processes, physiological processes, population of species, thermal control processes, complex power systems, automobile, air craft etc. exist in bilinear and linear time varying nature. The aim of the section is to develop a design procedure to design optimum reduced order observer-estimator for bilinear-cum-linear time varying systems.

The problem of estimating the state variables of a dynamic system, given observations of the output, variables, is of fundamental importance in the design of an optimal control system. If one considers the class of linear systems, then there are two approaches available in the literature. If the output variables can be measured exactly and if there are no other stochastic disturbances acting on the system, then one can use a Luenberger Observer. On the other hand, if all the output variables are corrupted by additive white noise, then one can use a Kalman filter for state estimation.

There are many cases in which some of the output variables are noise free while others are noisy. One can argue that no measurement is exactly noise free. On the other hand, there are many engineering systems in which the accuracy of measuring one variable is much greater than the accuracy of measuring some others. In such problems the measurement covariance matrix is almost singular and it can lead to ill-conditioned matrices and numerical problems. Thus, one can attempt to model the very accurate measurements as being deterministic.

Bilinear systems [80] [81] are special class of nonlinear systems as they are linear separately with respect to state and Control variables, but not jointly i.e. products of state and control variables appear in the system equations. The Continuous time SISO Bilinear System (BLS) are characterized by the following dynamic equations.

$$\dot{X}(t) = A^0 X(t) + A^1 U(t) X(t) + B U(t); \quad X(0)=Xo \quad .... (2.15)$$
$$Y(t) = C X(t) \quad ..... (2.16)$$

Where: the state vector X, input vector U, output vector Y, the System matrices A^0, A^1, B and C are of dimensions n, 1, 1, (n x n), (n x n), (n x 1) and (1 x n)3 respectively.

Bilinear systems can also be defined to be time varying or time invariant according to how A^0, A^1, B, C depend on time t. They are called:

- homogeneous in state if B=0
- homogeneous in the input if $A^0 = 0$ and
- strictly bilinear if $A^0 = B = 0$

The input controls the state evolution not only additively by means of term B and U(t) but also multiplicatively be means of term $A^1 U(t) X(t)$. Hence control for the BLS is more effective than for linear systems. With respect to optimization BLS offer better performance. The multiplicative control allows the modeling of those systems whose dynamics depend at least approximately in a linear fashion on a control law. Dynamics of general AC machine is represented by a quadratic system.

2.6.1 Observer Design

Consider the continuous time, bilinear time invariant system given by

$$\dot{X}(t) = A^0 X(t) + \Sigma_{J=1}^{P} A^J U_J(t) X(t) + B U(t) \quad ..... (2.17)$$
$$Y(t) = C X(t) \quad ..... (2.18)$$

Where: the dimensions of state vector X, input U, output Y, system matrices A^0, A^J, B and C are n, p, m, (n x n), (n x n), (n x p) and (m x n) respectively System can also be rewritten as a linear time varying system (LTVS) in the form,

$$\dot{X}(t) = [A^0 + \Sigma_{J=1}{}^{P} A^J U_J(t)] X(t) + B U(t) \quad \ldots 2.19$$

$$= A(t) X(t) + B U(t) \quad \ldots\ldots\ldots (2.20)$$

where: $A(t) = A^0 + A^1 U_1(t) + A^2 U_2(t) + \ldots + A^P U_p(t)$

$$= A^0 + \Sigma_{J=1}{}^{P} A^J U_J(t) \quad \ldots\ldots\ldots (2.21)$$

Let the q-dimensional observer be of the form

$$\dot{Z}(t) = D Z(t) + F(t) Y(t) + T(t) B U(t) \quad \ldots\ldots\ldots (2.22)$$

with r-dimensional output

$$W(t) = G(t) Z(t) + H(t) Y(t) \quad \ldots\ldots\ldots (2.23)$$

Where: Dimensions of the observer state Z, observer matrices D, F, T, G and H are q, (q x q), (q x m), (q x n), (r x q) and (r x m) respectively.

For the design of observer, various approaches such as: Funahansi [82-83], Generalized Matrix Inverse [84], Hara's and are made.

We will use Generalized Matrix Inverse method subject to the conditions:

- System must be completely observable.
- D is a stable matrix
- T(t) is differentiable matrix i.e. non singular for all t and $T^{-1}(t)$ must exist.
- $F(t) C = [\dot{T}(t) + T(t) A(t) - D T(t)]$
- Consistency condition, $[\dot{T}(t)+T(t) A(t)-D T(t)] (I-C^{-1}C)=0$ must be satisfied.
- $G(t) T(t) + H(t) C = K(t)$, where K(t) is the feedback gain matrix.

Discrete linear time varying system (LTVS) in the form,

$$X(k+1) = [A^0 + A^1 U_1(k) + A^2 U_2(k) + \ldots + A^P Up(k)] X(t) + BU(k)$$

$$= A(k) X(k) + B U(k) \quad \ldots\ldots\ldots (2.24)$$

where $A(k) = [A0 + \Sigma_{J=1}{}^{P} A^J U_J(k)]$

Since U(k) is function of time, A(k) is function of time itself and its derivatives are bounded. Assuming that observer is of the form:

$$Z(k+1) = D Z(k) + F(k) Y(k) + T(k+1) B U(k) \quad \ldots. (2.25)$$

can be designed using measurement on input and output such that Z(k) represents a linear transformation of states as $Z(k) = T(k) X(k)$, provided:

- The system is completely observable. $|T(k+1)\ ;\ C\ |^T$ is invertible.
- The consistency condition $[T(k+1)\ A(k) - D\ T(k)][I - C^{-1}\ C] = 0$ must be satisfied
- $F(k) = [T(k+1)\ A(k) - D\ T(k)]C^{-1}$
  Where C^{-1} is the generalized inverse of matrix C.

It can be proved that if the above conditions are satisfied $Z(k) \rightarrow T(k)\ X(k)$ as $k \rightarrow \infty$.
It is possible to design a reduced order observer of dimension q=n-m can be designed using following stepwise procedure:

1. Check system observability.

2. Choose a constant [(n-m) x (n-m)] matrix D, whose eigenvalues are arbitrary, uncommon with A (k) and be with in the unit circle of stability.

3. Let T (k+1) be [(n-m) x n] matrix, whose elements are unspecified function.

4. Matrix T(k) must satisfy the consistency condition
   $[T(k+1)A(k)-DT(k)][I - C^{-1}C] = 0$

5. Assuming C has a full rank i.e. system has no redundant outputs [rank c = m < n]
   Compute
   $F(k) = [T(k+1)A(k)-DT(k)]C\text{-}1$

6. Use unspecified element of D and T(k) to satisfy the condition
   $\det\ |T(k+1) : C|^T \neq 0$ for all $k > 0$

7. The estimated state X (k+1) is obtained using the relation:
   $\{ [T(k+1)\ \ C]^T \}^{-1}\ \{ [Z(k+1)\ \ Y(k+1)]^T \}^{-1}$

2.6.2 C Implementation

This section describes case studies of optimal reduced order observer design for second order, third order systems with single/multiple outputs.

2.6.2.1 Optimum reduced order observer-estimator for Second Order LTVS with single output

Consider a second order system with single output as,

$$X(k+1) = \begin{bmatrix} \alpha_1(k) & 1 \\ 0 & \alpha_2(k) \end{bmatrix} X(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} U(k$$

$Y(k+1) = [1\ \ 0]\ X(k)$. With initial condition $X^T(0) = [\ 1\ \ 1\ ]$ and $U_\infty(k) = I$ .The performance index: $J_{30} = 1/2\ [10\ X^2(30)] + 1/2\ d\ [X^2(k) + U^2(k)]$
Design an optimum reduced order observer for two different sets of eigen values observer system matrix D. Assume sinusoidal variation of $\alpha(k)$. Study for

the exponential variation of $\alpha(k)$.

Let us assume: $\alpha_1(k) = - 0.2 - 0.10 \text{ Sin } (k \Pi /6)$

$\alpha_2(k) = - 0.3 - 0.15 \text{ Sin } (k \Pi /6)$ ... for sinusoidal Variation

and

$\alpha_1(k) = - 0.2 - 0.10 \text{ exp } (-0.1 k)$

$\alpha_2(k) = - 0.3 - 0.15 \text{ exp } (-0.2 k)$ ... for exponential Variation

```
/* Sinusoidal variation of alpha if A=1 , B=0 */
/* Exponential variation of alpha if A=0 , B=1 */
#include <stdio.h>
#include<conio.h>
#include <math.h>
#define pi  3.14
FILE *fd1;
void main()
        { char file1[8];
          int A,B,k;
          float D11,DEL,alpha1[30],alpha2[30],P11[30],P12[30],P21[30],
              P22[30],K11[30],K12[30],F11[30],G11[30],H11[30],Z1[30],
              WN1[30],X1[30],X2[30],X1E[30],X2E[30],N[30],L[30],O[30]
              ,Z1A[30],Z1O[30],X1OE[30],X2OE[30];
          clrscr();
          printf("enter o/p file name\n");
          scanf("%s",file1);
          clrscr();
          printf("Enter the values of A,B,D11\n");
          scanf("%d %d %f",&A,&B,&D11);
          for(k=0;k<=30;k++)
           { alpha1[k]=A*(-0.2-0.010*sin(k*pi/6))+B*(-0.2-0.1*exp(-0.1
                    *k));
             alpha2[k]=A*(-0.3-0.015*sin(k*pi/4))+B*(-0.3-0.2*exp(-0.2
                    *k));
           }
/* Determination of optimal control vector    */
          P11[30]=10.0;
          P12[30]=0.0;
          P21[30]=0.0;
          P22[30]=10.0;

          for(k=29;k>=0;k--)
           { DEL=1+P22[k+1];
             K11[k]=(alpha1[k]*P21[k+1])/DEL;
             K12[k]=(P21[k+1]+alpha2[k]*P22[k+1])/DEL;
             P11[k]=1+alpha1[k]*alpha1[k]*P11[k+1]-alpha1[k]*P21[k+1]
                    *K11[k];
             P12[k]=alpha1[k]*(P11[k+1]+alpha2[k]*P12[k+1]-P21[k+1]*
```

```
                K12[k]);
            P21[k]=alpha1[k]*(P11[k+1]+alpha2[k]*P21[k+1])-(P21[k+1]
                +alpha2[k]*P22[k+1])*K11[k];
            P22[k]=1+P11[k+1]+alpha2[k]*P12[k+1]+alpha2[k]*(P21[k+1]
                +alpha2[k]*P22[k+1])-(P21[k+1]+alpha2[k]*P22[k+1])
                *K12[k];
           }
/* Determination of observer parameters */
          for(k=1;k<=30;k++)
           { F11[k]=-alpha1[k]*alpha2[k]+D11*alpha1[k]+D11*alpha2[k-1]
                -D11*D11;
            G11[k]=K12[k];
            H11[k]=K11[k]-K12[k]*(D11-alpha2[k-1]);
           }
/* Estimation of states with noiseless observer output */
          Z1[1]=0.3;
          X1[1]=1.0;
          X2[1]=1.0;
          for(k=1;k<=29;k++)
           { X1[k+1]=alpha1[k]*X1[k]+X2[k];
            X2[k+1]=-K11[k]*X1[k]+(alpha2[k]-K12[k])*X2[k]+1;
            Z1[k+1]=D11*Z1[k]+F11[k]*X1[k]-K11[k]*X1[k]-K12[k]*X2[k]
                +1;
            X1E[k+1]=X1[k+1];
            X2E[k+1]=Z1[k+1]+(alpha2[k]-D11)*X1[k+1];
           }
/* Estimation of states when output of observer is corrupted by noise */
          O[1]=1.0;
          Z1O[1]=1.0;
          for(k=2;k<=30;k++)
           { N[k]=0.2*sin(k*pi/12)+0.1*cos(k*pi/9);
            L[k]=D11*D11*O[k-1];
            O[k]=L[k]/(1+2*L[k]*G11[k]*G11[k]);
            Z1A[k]=D11*Z1O[k-1]+F11[k-1]*X1[k-1]-K11[k-1]*X1[k-1]
                -K12[k-1]*X2[k-1]+1;
            WN1[k]=G11[k]*Z1[k]+H11[k]*X1[k]+N[k];
            Z1O[k]=Z1A[k]+2*O[k]*G11[k]*(WN1[k]-G11[k]*Z1A[k]-H11[k]
                *X1[k]);
            X1OE[k]=X1[k];
            X2OE[k]=Z1O[k]+(alpha2[k-1]-D11)*X1[k];
           }
/* Output results and Creation of results files */
          fd1=fopen(file1,"w");//output will be printed in out.cpp file//
          if(A==1 && B==0)
                { printf("Sinusoidal variation of alpha\n");
                 fprintf(fd1,"Sinusoidal variation of alpha\n");
```

```
                    printf("alpha1[k] =-0.2-0.010*sin(k*pi/6)\n");
                    fprintf(fd1,"alpha1[k] =-0.2-0.010*sin(k*pi/6)\n");
                    printf("alpha2[k] =-0.3-0.015*sin(k*pi/4)\n");
                    fprintf(fd1,"alpha2[k] =-0.3-0.015*sin(k*pi/4)\n");
                   }
             else
                   { printf("Exponential variation of alpha\n");
                    fprintf(fd1,"Exponential variation of alpha\n");
                    printf("alpha1[k] =-0.2-0.1*exp(-0.1*k)\n");
                    fprintf(fd1,"alpha1[k] =-0.2-0.1*exp(-0.1*k)\n");
                    printf("alpha2[k] =-0.3-0.2*exp(-0.2*k)\n");
                    fprintf(fd1,"alpha2[k] =-0.3-0.2*exp(-0.2*k)\n");
                   }
          fpprintf("A=%d    B=%d    D11=%.1f\n",A,B,D11);
          fprintf(fd1,"A=%d    B=%d    D11=%.1f\n",A,B,D11);
          printf("-------------------------------------------------\n");
          fprintf(fd1,"-------------------------------------------------\n");
          printf("K11[k]  K12[k]  X2[k]   X2E[k]  X2OE[k] \n");
          fprintf(fd1,"K11[k]  K12[k]  X2[k]   X2E[k]  X2OE[k] \n");
          printf("-------------------------------------------------\n");
          fprintf(fd1,"-------------------------------------------------\n");
          printf("\n");
          fprintf(fd1,"\n");
    for(k=2;k<=30;k++)
    {
    printf("%.4f  %.4f  %.4f %.4f  %.4f \n"  ,K11[k],  K12[k], X2[k],X2E[k],
X2OE[k]);
    fprintf(fd1," %.4f  %.4f  %.4f %.4f  %.4f \n ",K11[k],  K12[k],  X2[k],
X2E[k],X2OE[k]);
     }
          printf("-------------------------------------------------\n");
          fprintf(fd1,"-------------------------------------------------\n");
          getch();
          fclose(fd1);
}
```

2.6.2.2 Optimum reduced order observer-estimator for Third Order LTVS with single output

```
/* Sinusoidal variation of alpha if A=1 , B=0 */
/* Exponential variation of alpha if A=0 ,B=1 */
    #include <stdio.h>
    #include<conio.h>
    #include<math.h>
    #define pi  3.14
    FILE *fd1;
```

```
void main ()
{ int A,B,k;
 char file1[9];
 float D11,D22,DEL[30],DEL1[30],DEL2[30],DEL3[30],alpha1[30],
       alpha2[30],alpha3[30],P11[30],P12[30],P13[30],P21[30],
       P22[30],P23[30],P31[30],P32[30],P33[30],K11[30],K12[30],
       K13[30],F11[30],F21[30],G11[30],G12[30],H11[30],T11[30],
       T12[30],T21[30],T22[30],Z1[30],Z2[30],WN1[30],WN2[30],
       X1[30],X2[30],X3[30],X1E[30],X2E[30],X3E[30],N[30],
       L11[30],L12[30],L21[30],L22[30],OA11[30],OA12[30],
       OA21[30],OA22[30],O11[30],O12[30],O21[30],O22[30],
       Z1A[30],Z2A[30],Z1O[30],Z2O[30],X1OE[30],X2OE[30],
       X3OE[30];
 clrscr();
 printf("enter o/p file name\n");
 scanf("%s",file1);
 clrscr();
 printf("enter the values of A,B,D11,D22\n");

 scanf("%d %d %f %f",&A,&B,&D11,&D22);
 for(k=0;k<=30;k++)
  { alpha1[k]=A*(-0.2-0.010*sin(k*pi/6))+B*(-0.2-0.1
              *exp(-0.1*k));
    alpha2[k]=A*(-0.3-0.015*sin(k*pi/4))+B*(-0.3-0.2
              *exp(-0.2*k));
    alpha3[k]=A*(-0.4-0.012*sin(k*pi/5))+B*(-0.4-0.3
              *exp(-0.3*k));
  }
```

/* Determination of optimal control vector */

```
 P11[30]=10.0;
 P12[30]=0.0;
 P13[30]=0.0;
 P21[30]=0.0;
 P22[30]=10.0;
 P23[30]=0.0;
 P31[30]=0.0;
 P32[30]=0.0;
 P33[30]=10.0;
 for(k=29;k>=0;k--)
  { DEL[k]=1+P33[k+1];
    K11[k]=-P33[k+1]/DEL[k];
    K12[k]=(alpha1[k]*P31[k+1]-P33[k+1])/DEL[k];
    K13[k]=(alpha2[k]*P32[k+1]-alpha3[k]*P33[k+1])/DEL[k];
    P11[k]=1+(1+K11[k])*P33[k+1];
    P12[k]=-alpha1[k]*P31[k+1]+(1+K12[k])*P33[k+1];
```

```
            P13[k]=-alpha2[k]*P32[k+1]+(alpha3[k]+K13[k])*P33[k+1];
            P21[k]=-alpha1[k]*P13[k+1]+P33[k+1]-K11[k]*(alpha1[k]
                  *P31[k+1]-P33[k+1]);
            P22[k]=1+alpha1[k]*(alpha1[k]*P11[k+1]-P13[k+1]-P31[k+1])
                  +P33[k+1]-K12[k]*(alpha1[k]*P31[k+1]-P33[k+1]);
            P23[k]=alpha1[k]*(alpha2[k]*P12[k+1]-alpha3[k]*P13[k+1])
                  -alpha2[k]*P32[k+1]+alpha3[k]*P33[k+1]-K13[k]
                  *(alpha1[k]*P31[k+1]-P33[k+1]);
            P31[k]=-alpha2[k]*P23[k+1]+alpha3[k]*P33[k+1]-K11[k]
                  *(alpha2[k]*P32[k+1]-alpha3[k]*P33[k+1]);
            P32[k]=alpha2[k]*(alpha1[k]*P21[k+1]-P23[k+1])-alpha3[k]
                  *(alpha1[k]*P31[k+1]-P33[k+1])-K12[k]*(alpha2[k]
                  *P32[k+1]-alpha3[k]*P33[k+1]);
            P33[k]=1+alpha2[k]*(alpha2[k]*P22[k+1]-alpha3[k]*P23[k+1])
                  -alpha3[k]*(alpha2[k]*P32[k+1]-alpha3[k]*P33[k+1])
                  -K13[k]*(alpha2[k]*P32[k+1]-alpha3[k]*P33[k+1]);
           }
/* Determination of observer parameters */
        for(k=2;k<=30;k++)
         { T11[k]=(1+D11*(alpha3[k-2]+D11)/alpha2[k-2])/alpha1[k-1];
          T12[k]=(alpha3[k-1]+D11)/alpha2[k-1];
          T21[k]=(1+D22*(alpha3[k-2]+D22)/alpha2[k-2])/alpha1[k-1];
          T22[k]=(alpha3[k-1]+D22)/alpha2[k-1];
          F11[k]=-1-D11*T11[k];
          F21[k]=-1-D22*T21[k];
          G11[k]=(K12[k]-K13[k]*T22[k])/(T12[k]-T22[k]);
          G12[k]=K13[k]-G11[k];
          H11[k]=K11[k]-G11[k]*T11[k]-G12[k]*T21[k];
         }

/* Estimation of states with noiseless observer output */
        Z1[2]=0.3;
        Z2[2]=0.2;
        X1[2]=1.0;
        X2[2]=1.0;
        X3[2]=1.0;
        for(k=2;k<=29;k++)
         { X1[k+1]=alpha1[k]*X2[k];
          X2[k+1]=alpha2[k]*X3[k];
          X3[k+1]=-(1+K11[k])*X1[k]-(1+K12[k])*X2[k]-(alpha3[k]
                +K13[k])*X3[k]+1;
          Z1[k+1]=D11*Z1[k]+(F11[k]-K11[k])*X1[k]-K12[k]*X2[k]
                -K13[k]*X3[k]+1;
          Z2[k+1]=D22*Z2[k]+(F21[k]-K11[k])*X1[k]-K12[k]*X2[k]
                -K13[k]*X3[k]+1;
          DEL1[k]=T12[k+1]-T22[k+1];
```

```
X1E[k+1]=X1[k+1];
X2E[k+1]=(Z1[k+1]-Z2[k+1]+(T21[k+1]-T11[k+1])*X1[k+1])
        /DEL1[k];
X3E[k+1]=(-T22[k+1]*Z1[k+1]+T12[k+1]*Z2[k+1]+(T11[k+1]
        *T22[k+1]-T12[k+1]*T21[k+1])*X1[k+1])/DEL1[k];
}
```

/* Estimation of states with noisy observer output */

```
O11[2]=1.0;
O12[2]=1.0;
O21[2]=1.0;
O22[2]=1.0;
Z1O[2]=1.0;
Z2O[2]=1.0;
 for(k=3;k<=30,k++)
  { N[k]=0.2*sin(k*pi/12)+0.1*cos(k*pi/9);
   L11[k]=O11[k-1]*D11*D11;
   L12[k]=O12[k-1]*D11*D22;
   L21[k]=O21[k-1]*D11*D22;
   L22[k]=O22[k-1]*D22*D22;
   DEL2[k]=((1+2*G11[k]*G11[k]*L11[k]+2*G11[k]*G12[k]*L12[k])
        *(1+2*G11[k]*G12[k]*L21[k]+2*G12[k]*G12[k]*L22[k])
        -4*(G11[k]*G12[k]*L11[k]+G12[k]*G12[k]*L12[k])
        *(G11[k]*G11[k]*L21[k]+G11[k]*G12[k]*L22[k]));
   OA11[k]=(1+2*G11[k]*G12[k]*L21[k]+2*G12[k]*G12[k]*L22[k])
        /DEL2[k];
   OA12[k]=-2*(G11[k]*G12[k]*L11[k]+G12[k]*G12[k]*L12[k])
        /DEL2[k];
   OA21[k]=-2*(G11[k]*G11[k]*L21[k]+G11[k]*G12[k]*L12[k])
        /DEL2[k];
   OA22[k]=(1+2*G11[k]*G11[k]*L11[k]+2*G11[k]*G12[k]*L12[k])
        /DEL2[k];
   O11[k]=OA11[k]*L11[k]+OA12[k]*L21[k];
   O12[k]=OA11[k]*L12[k]+OA12[k]*L22[k];
   O21[k]=OA21[k]*L11[k]+OA22[k]*L21[k];
   O22[k]=OA21[k]*L12[k]+OA22[k]*L22[k];
   WN1[k]=G11[k]*Z1[k]+G12[k]*Z2[k]+H11[k]*X1[k]+N[k];
   Z1A[k]=D11*Z1O[k-1]+(F11[k-1]-K11[k-1])*X1[k-1]-K12[k-1]
        *X2[k-1]-K13[k-1]*X3[k-1]+1;
   Z2A[k]=D22*Z2O[k-1]+(F21[k-1]-K11[k-1])*X1[k-1]-K12[k-1]
        *X2[k-1]
        -K13[k-1]*X3[k-1]+1;
   Z1O[k]=Z1A[k]+2*(O11[k]*G11[k]+O12[k]*G12[k])*(WN1[k]
        -G11[k]*Z1A[k]-G12[k]*Z2A[k]-H11[k]*X1[k]);
   Z2O[k]=Z2A[k]+2*(O21[k]*G11[k]+O22[k]*G12[k])*(WN1[k]
        -G11[k]*Z1A[k]-G12[k]*Z2A[k]-H11[k]*X1[k]);
```

```
            DEL3[k]=(T12[k]-T22[k]);
            X1OE[k]=X1[k];
            X2OE[k]=(Z1O[k]-Z2O[k]+(T21[k]-T11[k])*X1[k])/(DEL3[k]);
            X3OE[k]=(-T22[k]*Z1O[k]+T12[k]*Z2O[k]+(T11[k]*T22[k]
                    -T12[k]*T21[k])*X1[k])/(DEL3[k]);
          }
```

/* Output results and Creation of results files */

```
        fd1=fopen(file1,"w");/*Output will be printed in out.cpp file*/
        if(A==1 && B==0)
         { printf("Sinusoidal variation of alpha \n");
           fprintf(fd1,"Sinusoidal variation of alpha \n");
           printf("alpha1[k]=-0.2-0.010*sin(k*pi/6) \n");
           fprintf(fd1,"alpha1[k]=-0.2-0.010*sin(k*pi/6) \n");
           printf("alpha2[k]=-0.3-0.015*sin(k*pi/4) \n");
           fprintf(fd1,"alpha2[k]=-0.3-0.015*sin(k*pi/4) \n");
           printf("alpha3[k]=-0.4-0.012*sin(k*pi/5) \n");
           fprintf(fd1,"alpha3[k]=-0.4-0.012*sin(k*pi/5) \n");
         }
        else
         { printf("Exponential variation of alpha \n");
           fprintf(fd1,"Exponential variation of alpha \n");
           printf("alpha1[k]=-0.2-0.1*exp(-0.1*k) \n");
           fprintf(fd1,"alpha1[k]=-0.2-0.1*exp(-0.1*k) \n");
           printf("alpha2[k]=-0.3-0.2*exp(-0.2*k) \n");
           fprintf(fd1,"alpha2[k]=-0.3-0.2*exp(-0.2*k) \n");
           printf("alpha3[k]=-0.4-0.3*exp(-0.3*k) \n");
           fprintf(fd1,"alpha3[k]=-0.4-0.3*exp(-0.3*k) \n");
         }
        printf("A=%d  B=%d  D11=%.1f  D22=%.1f \n\n", A,B,D11,D22);
        fprintf(fd1,"A=%d  B=%d  D11=%.1f  D22=%.1f \n\n", A,B,D11
                ,D22);
        printf("-------------------------------------------------------------------------------\n");
        fprintf(fd1,"-------------------------------------------------------------------------------\n");
printf("k X1[k]  X2[k]  X3[k]  X1E[k] X2E[k]  X3E[k] XO1E[k]  X2OE[k] X3OE[k] \n");
fprintf(fd1,"k X1[k] X2[k] X3[k] X1E[k] X2E[k] X3E[k] XO1E[k] X2OE[k] X3OE[k] \n");
printf("-------------------------------------------------------------------------------\n\n");
fprintf(fd1,"-------------------------------------------------------------------------------\n\n");
   for(k=3;k<=30;k++)
       {
    printf("%d %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f %.4f
          \n",k,X1[k],X2[k],X3[k],X1E[k],X2E[k],X3E[k],X1OE[k],X2OE[k],X3OE[k]);
    fprintf(fd1,"%d %.3f %.4f %.4f %.3f %.4f %.4f %.3f %.4f %.4f
          \n",k,X1[k],X2[k],X3[k],X1E[k],X2E[k],X3E[k],X1OE[k],X2OE[k],X3OE[k]);
          }
printf("-------------------------------------------------------------------------------\n");
```

```
fprintf(fd1,"----------------------------------------------------------------------------\n");
getch();
fclose(fd1);
}
```

2.6.2.3 Optimum reduced order observer-estimator for Third Order LTVS with Two outputs

Consider a Third order system with Two outputs as,

$$X(k+1) = \begin{bmatrix} \alpha_1(k) & 1 \\ 0 & \alpha_2(k) \end{bmatrix} X(k) + \begin{bmatrix} 0 \\ 1 \end{bmatrix} U(k$$

$Y(k+1) = [1\ 0]\ X(k)$. With initial condition $X^T(0) = [\ 1\ \ 1\ ]$ and $U_\infty(k) = I$ .The performance index: $J_{30} = 1/2\ [10\ X^2(30)] + 1/2\ d\ [X^2(k) + U^2(k)]$

Design an optimum reduced order observer for two different sets of eigen values observer system matrix D. Assume sinusoidal variation of $\alpha(k)$. Study for the exponential variation of $\alpha(k)$.

Let us assume: $\alpha_1(k) = -0.2 - 0.10\ \text{Sin}\ (k\ \Pi/6)$

$\alpha_2(k) = -0.3 - 0.15\ \text{Sin}\ (k\ \Pi/6)$ ... for sinusoidal Variation

and

$\alpha_1(k) = -0.2 - 0.10\ \exp(-0.1\ k)$

$\alpha_2(k) = -0.3 - 0.15\ \exp(-0.2\ k)$ ... for exponential Variation

/* Sinusoidal variation of alpha if A=1 , B=0 */

/* Exponential variation of alpha if A=0 , B=1 */

```
#include <stdio.h>
#include<conio.h>
#include <math.h>
#define pi 3.14
FILE *fd1;
void main()
{ char file1[9];
 int A,B,k;
 float D11,DEL,alpha1[30],alpha2[30],alpha3[30],P11[30],P12[30]
       ,P13[30],P21[30],P22[30],P23[30],P31[30],P32[30],P33[30]
       ,K11[30],K12[30],K13[30],F11[30],F21[30],G11[30],H11[30]
       ,H12[30],Z1[30],WN1[30],X1[30],X2[30],X3[30],X1E[30],
       X2E[30],X3E[30],N[30],L[30],O[30],Z1A[30],Z1O[30],
       X1OE[30],X2OE[30],X3OE[30];
 clrscr();
 printf("enter o/p filename\n");
 scanf("%s",file1);
 clrscr();
 printf("enter the values of A,B,D11\n");
 scanf("%d %d %f",&A,&B,&D11);
```

```
        for(k=0;k<=30;k++)
         { alpha1[k]=A*(-0.2-0.010*sin(k*pi/6))+B*(-0.2-0.1*exp(-0.1
                    *k));
           alpha2[k]=A*(-0.3-0.015*sin(k*pi/4))+B*(-0.3-0.2*exp(-0.2
                    *k));
           alpha3[k]=A*(-0.4-0.012*sin(k*pi/5))+B*(-0.4-0.3*exp(-0.3
                    *k)) ;
         }
/* Determination of optimal control vector */
        P11[30]=10.0;
        P12[30]=0.0;
        P13[30]=0.0;
        P21[30]=0.0;
        P22[30]=10.0;
        P23[30]=0.0;
        P31[30]=0.0;
        P32[30]=0.0;
        P33[30]=10.0;
        for(k=29;k>=0;k--)
         { DEL=1+P33[k+1];
           K11[k]=-P33[k+1]/DEL;
           K12[k]=(alpha1[k]*P31[k+1]-P33[k+1])/DEL;
           K13[k]=(P31[k+1]+alpha2[k]*P32[k+1]-alpha3[k]*P33[k+1])/
                   DEL;
           P11[k]=1+(1+K11[k])*P33[k+1];
           P12[k]=-alpha1[k]*P31[k+1]+(1+K12[k])*P33[k+1];
           P13[k]=-P31[k+1]-alpha2[k]*P32[k+1]+(alpha3[k]+K13[k])
                   *P33[k+1];
           P21[k]=-alpha1[k]*P13[k+1]+P33[k+1]-K11[k]*(alpha1[k]
                   *P31[k+1]-P33[k+1]);
           P22[k]=1+alpha1[k]*(alpha1[k]*P11[k+1]-P13[k+1]-P31[k+1])
                   +P33[k+1]-K12[k]*(alpha1[k]*P31[k+1]-P33[k+1]);
           P23[k]=alpha1[k]*(P11[k+1]+alpha2[k]*P12[k+1]-alpha3[k]
                   *P13[k+1])-P31[k+1]-alpha2[k]*P32[k+1]+alpha3[k]
                   *P33[k+1]-K13[k]*(alpha1[k]*P31[k+1]-P33[k+1]);
           P31[k]=-P13[k+1]-alpha2[k]*P23[k+1]+alpha3[k]*P33[k+1]
                   -K11[k]*(P31[k+1]+alpha2[k]*P32[k+1]-alpha3[k]
                   *P33[k+1]);
           P32[k]=alpha1[k]*P11[k+1]-P13[k+1]+alpha2[k]*(alpha1[k]
                   *P21[k+1]-P23[k+1])-alpha3[k]*(alpha1[k]*P31[k+1]
                   -P33[k+1])-K12[k]*(P31[k+1]+alpha2[k]*P32[k+1]
                   -alpha3[k]*P33[k+1]);
           P33[k]=1+P11[k+1]+alpha2[k]*P12[k+1]-alpha3[k]*P13[k+1]
                   +alpha2[k]*(P21[k+1]+alpha2[k]*P22[k+1]-alpha3[k]
                   *P23[k+1])-alpha3[k]*(P31[k+1]+alpha2[k]*P32[k+1]
                   -alpha3[k]*P33[k+1])-K13[k]*(P31[k+1]+alpha2[k]
```

```
                *P32[k+1]-alpha3[k]*P33[k+1]);
        }
/* Determination of observer parameter */
        for(k=1;k<=30;k++)
         { F11[k]=-1-D11;
          F21[k]=alpha1[k]-1-D11*(D11+alpha3[k-1]-1)/alpha2[k-1];
          G11[k]=K13[k];
          H11[k]=K11[k]-K13[k];
          H12[k]=K12[k]-K13[k]*(D11+alpha2[k-1]-1)/alpha2[k];
         }
/* Estimation of states with noiseless observer output */
        Z1[1]=0.3;
        X1[1]=1.0;
        X2[1]=1.0;
        X3[1]=1.0;
        for(k=1;k<=29;k++)
         { X1[k+1]=alpha1[k]*X2[k]+X3[k];
          X2[k+1]=alpha2[k]*X3[k];
          X3[k+1]=-(1+K11[k])*X1[k]-(1+K12[k])*X2[k]-(alpha3[k]
                +K13[k])*X3[k]+1;
          Z1[k+1]=D11*Z1[k]+(F11[k]-K11[k])*X1[k]+(F21[k]-K12[k])
                *X2[k]-K13[k]*X3[k]+1;
          X1E[k+1]=X1[k+1];
          X2E[k+1]=X2[k+1];
          X3E[k+1]=Z1[k+1]-X1[k+1]-((D11+alpha3[k]-1)/alpha2[k])
                 *X2[k+1];
         }
/* Estimation of states with noisy observer output */
        O[1]=1.0;
        Z1O[1]=1.0;
        for(k=2;k<=30;k++)
         { N[k]=0.2*sin(k*pi/12)+0.1*cos(k*pi/9);
          L[k]=D11*D11*O[k-1];
          O[k]=L[k]/(1+2*L[k]*G11[k]*G11[k]);
          Z1A[k]=D11*Z1O[k-1]+(F11[k-1]-K11[k-1])*X1[k-1]+(F21[k-1]
                -K12[k-1])*X2[k-1]-K13[k-1]*X3[k-1]+1;
          WN1[k]=G11[k]*Z1[k]+H11[k]*X1[k]+H12[k]*X2[k]+N[k];
          Z1O[k]=Z1A[k]+2*O[k]*G11[k]*(WN1[k]-G11[k]*Z1A[k]-H11[k]
                *X1[k]-H12[k]*X2[k]);
          X1OE[k]=X1[k];
          X2OE[k]=X2[k];
          X3OE[k]=Z1O[k]-X1[k]-((D11+alpha3[k-1]-1)/alpha2[k-1])
                 *X2[k];
         }
/* Output results and Creation of results files */
        fd1=fopen(file1,"w");/*Output will be stored in out.cpp file*/
```

```
        if(A==1 && B==0)
         { printf(" Sinusoidal variation of alpha \n");
           fprintf(fd1," Sinusoidal variation of alpha \n");
           printf("alpha1[k]=-0.2-0.010*sin(k*pi/6)\n");
           fprintf(fd1,"alpha1[k]=-0.2-0.010*sin(k*pi/6)\n");
           printf("alpha2[k]=-0.3-0.015*sin(k*pi/4)\n");
           fprintf(fd1,"alpha2[k]=-0.3-0.015*sin(k*pi/4)\n");
           printf("alpha3[k]=-0.4-0.012*sin(k*pi/5)\n");
           fprintf(fd1,"alpha3[k]=-0.4-0.012*sin(k*pi/5)\n");
         }
        else
         { printf("Exponential variation of alpha \n");
           fprintf(fd1,"Exponential variation of alpha \n");
           printf("alpha1[k]=-0.2-0.1*exp(-0.1*k) \n");
           fprintf(fd1,"alpha1[k]=-0.2-0.1*exp(-0.1*k) \n");
           printf("alpha2[k]=-0.3-0.2*exp(-0.2*k) \n");
           fprintf(fd1,"alpha2[k]=-0.3-0.2*exp(-0.2*k) \n");
           printf("alpha3[k]=-0.4-0.3*exp(-0.3*k) \n");
           fprintf(fd1,"alpha3[k]=-0.4-0.3*exp(-0.3*k) \n");
         }
printf("A=%d    B=%d    D11=%.1f \n",A,B,D11);
fprintf(fd1,"A=%d    B=%d    D11=%.1f \n",A,B,D11);
printf("-------------------------------------------------------------------\n");
fprintf(fd1,"-------------------------------------------------------------------\n");
printf("K X1[k]  X2[k]  X3[k]  X1E[k]  X2E[k]  X3E[k]  X1OE[k]  X2OE[k]  X3OE[k]\n ");
fprintf(fd1,"K X1[k] X2[k] X3[k] X1E[k] X2E[k] X3E[k] X1OE[k] X2OE[k] X3OE[k]\n");
printf("-------------------------------------------------------------------\n");
fprintf(fd1,"-------------------------------------------------------------------\n");
printf("\n");
fprintf(fd1,"\n");
        for(k=2;k<=30;k++)
        {
        printf("%d %.3f %.3f %.4f %.3f %.3f %.4f %.3f %.3f %.4f \n"
              ,k,X1[k],X2[k],X3[k],X1E[k],X2E[k],X3E[k],X1OE[k],X2OE[k],X3OE[k]);
        fprintf(fd1,"%d %.3f %.3f %.4f %.3f %.3f %.4f %.3f %.3f %.4f \n"
              ,k,X1[k],X2[k],X3[k],X1E[k],X2E[k],X3E[k],X1OE[k],X2OE[k],X3OE[k]);
         }
printf("-------------------------------------------------------------------\n");
fprintf(fd1,"-------------------------------------------------------------------\n");
getche();
fclose(fd1);
}
```

2.7 Software Development Tools

MATLAB/SIMULINK [85-86] is a software package for high performance numerical computation and visualization. It provides an interactive environment with hundreds of built-in functions for technical computation, graphics, and animation. Best of all, it also provides easy extensibility with its own high-level programming language.

2.7.1 MATLAB

The basic building block of MATLAB [85] is the matrix. The fundamental data-type is the array. Vectors, scalars, real matrices and complex matrices are all automatically handled as special cases of the basic data-type. Also it never requires declaring the dimensions of a matrix. MATLAB simply loves matrices and matrix operations. The built-in functions are optimized for vector operations. Consequently, vectorized commands or codes run much faster in MATLAB than in C. MATLAB engines incorporate the LAPACK and BLAS libraries, embedding the state of the art in software for matrix computation.

In university environments, it is the standard instructional tool for introductory and advanced courses in mathematics, engineering, and science. In industry, MATLAB is the tool of choice for high-productivity research, development, and analysis. Fig 2.3 depicts MATLAB desktop containing tools (graphical user interfaces) for managing files, variables, and applications associated with MATLAB.

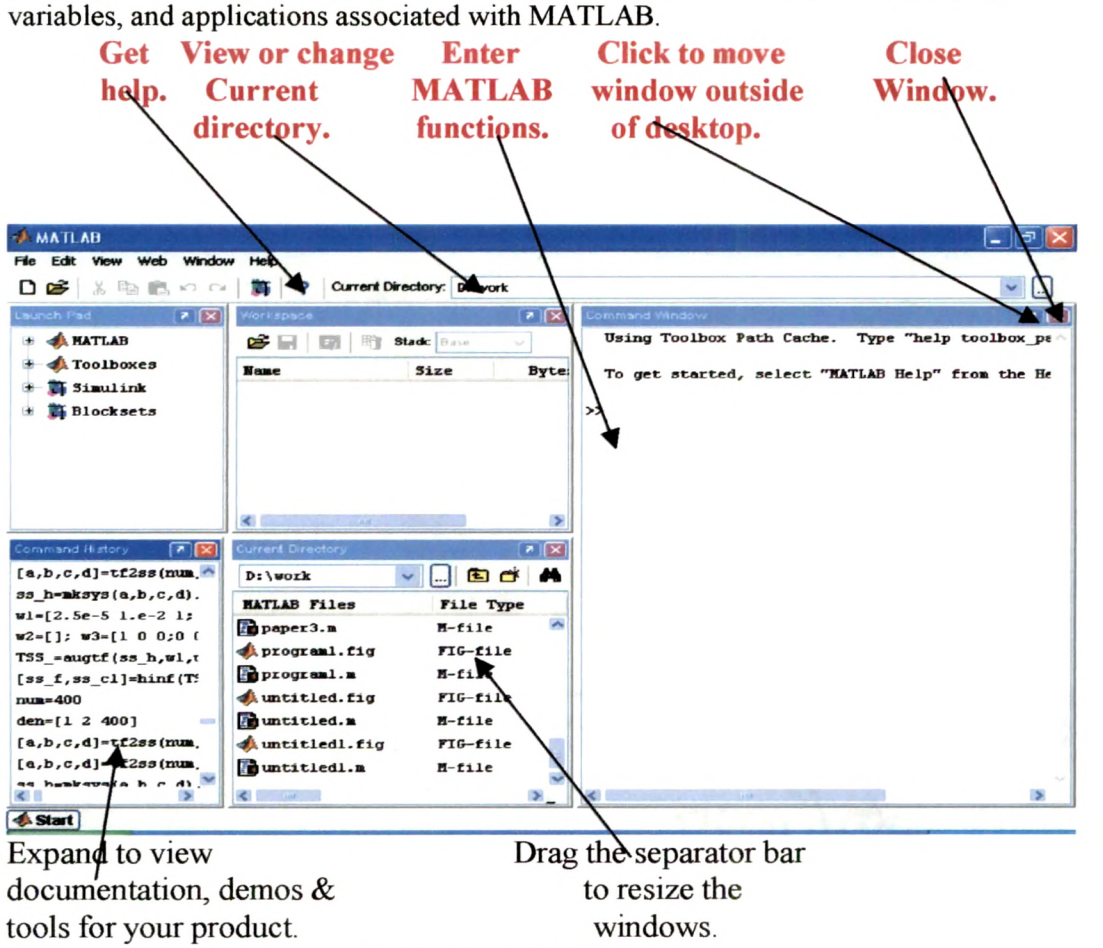


Fig 2.3 MATLAB Desktop

2.7.1.1 MATLAB Implementation

This section describes a sample MATLAB code for optimal reduced order observer design

```
% Sample MATLAB code
k=2:1:30;
alpha1(k)=-0.2-.010*sin(k*pi/6);
alpha2(k)=-0.3-.015*sin(k*pi/4);
P11(31)=10;   P12(31)=0;     P21(31)=0;
P22(31)=10;
D11=input('Enter the Value of D11 = ');
A=input('Enter the Value of A = ');
B=input('Enter the Value of B = ');

k=30:-1:1;
for i=30:-1:1
   DEL(i)=1+P22(i+1);
end
DEL(31)=1;
K11(k)=(alpha1(k).*P21(k+1))./DEL(k);
K12(k)=P21(k+1)+alpha2(k).*P22(k+1)./DEL(k);
P11(k)=1+alpha1(k).*alpha1(k).*P11(k+1)-alpha1(k).*P21(k+1).*K11(k);
P12(k)=alpha1(k).*P11(k+1)-alpha2(k).*P12(k+1)-P21(k+1).*K12(k);
P21(k)=alpha1(k).*(P11(k+1)+alpha2(k).*P21(k+1))-
(P21(k+1)+alpha2(k).*P22(k+1)).*K11(k);
P22(k)=1+P11(k+1)+alpha2(k).*P12(k+1) + alpha2 (k) .*
(P21(k+1))+alpha2(k).*P22(k+1)-(P21 (k+1) + alpha2(k) .*P22(k+1)).*K12(k);

k=2:1:30;
F11(k)=-alpha1(k).*alpha2(k)+D11*alpha1(k)+D11* alpha2(k-1)-D11*D11;
G11=K12;
H11(k)=K11(k)-K12(k).*(D11-alpha2(k-1));
Z1=zeros(1,30);
X1=zeros(1,30);
X2=zeros(1,30);
Z1(2)=0.3;
X1(2)=1;
X2(2)=1;

k=2:1:30;
X1(k+1)=alpha1(k).*X1(k)+X2(k);
X2(k+1)=-K11(k).*X1(k)+(alpha2(k)-K12(k).*X2(k))+1;
Z1(k+1)=D11*Z1(k)+F11(k).*X1(k)-K11(k).*X1(k)-K12(k).*X2(k)+1;
X1E(k+1)=X1(k+1);
X2E(k+1)=Z1(k+1)+(alpha2(k)-D11).*X1(k+1);
```

```
O=zeros(1,30);
Z10=zeros(1,30);

O(2)=1;
Z10(2)=1;

k=3:1:30;
N(k)=0.2*sin(k*pi/12)+0.1*cos(k*pi/9);
L(k)=D11*D11*O(k-1);
O(k)=L(k)./(1+2*L(k).*G11(k).*G11(k));
Z1A(k)=D11*Z10(k-1)+F11(k-1).*X1(k-1)-K11(k-1).*X1(k-1)-K12(k-1).*X2(k-1)+1;
WN1(k)=G11(k).*Z1(k)+H11(k).*X1(k)+N(k);
Z10(k)=Z1A(k)+2*O(k).*G11(k).*(WN1(k)-G11(k).*Z1A(k)-H11(k).*X1(k));
X10E(k)=X1(k);
X20E(k)=Z10(k)+(alpha2(k-1)-D11).*X1(k);

plot(k,X2(k),'y')
hold on
plot(k,X2E(k),'r')
plot(k,X20E(k),'b' )
grid on
title('Sinusoidal variation of Alpha, D11=0.5')
xlabel('----> k')
ylabel('----> X2')
print –dpsc
```

Fig 2.4 shows typical observer estimate for D11 = 0.5, A = 1 and B = 0.

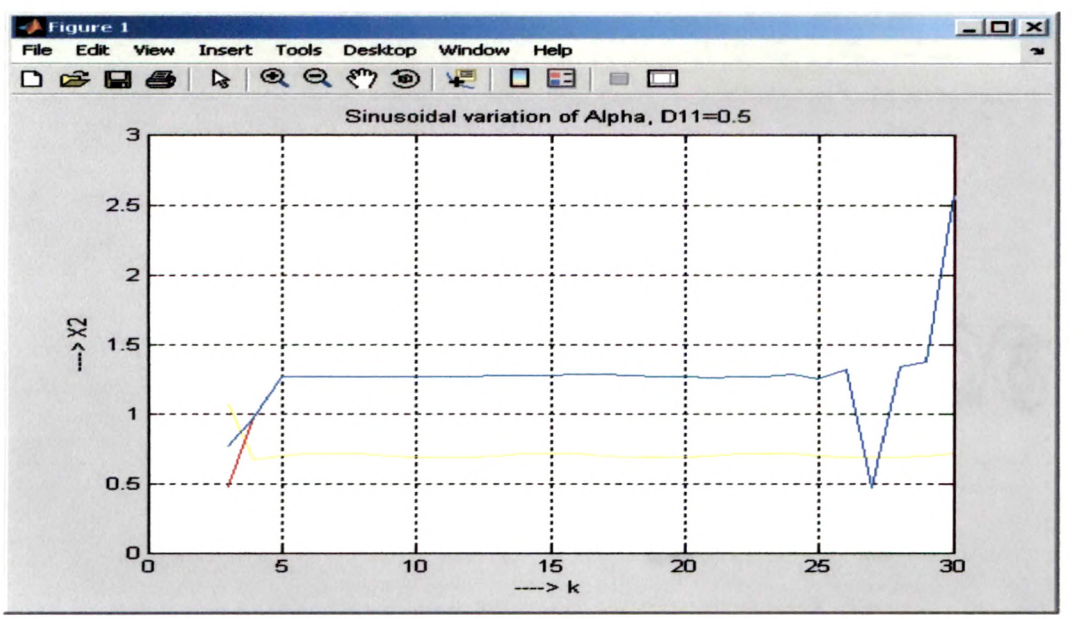


Fig 2.4 Observer-Estimator Output

2.7.2 SIMULINK

Simulink [86] is a software package for modeling, simulating, and analyzing dynamic systems. It supports linear and nonlinear systems, modeled in continuous time, sampled time, or a hybrid of the two. Systems can also be multirate, i.e., have different parts that are sampled or updated at different rates. Simulink is built on top of MATLAB. As an extension of MATLAB, Simulink adds many features specific to dynamic systems while retaining all of generl purpose functionality of MATLAB. So using Simulink we can direct access the wide range of MATLAB based tools for generating, analyzing, and optimizing system implemented in Simulink

Most of the natural systems and man made systems like the speed control system of car, a signal processing filter enabling telephone communication, can be considered as dynamic systems - that is a system which is characterized by change. Dynamic systems can often be viewed as comprising many elementary dynamic systems and can be considered as an object or block Fig 2.5 which is excited by external inputs and produces response i.e. outputs.

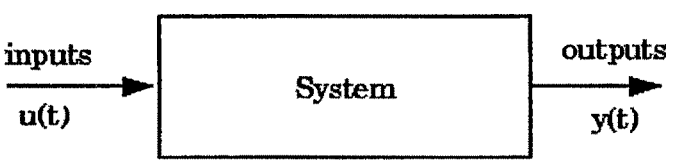


Fig 2.5 Block Diagram Representation of Dynamic System

A Simulink block diagram model is a graphical representation of a mathematical model of a dynamic system. A mathematical model of a dynamic system is described by a set of equations. The mathematical equations described by a block diagram model are known as algebraic, differential, and/or difference equations. At any given instant of time, these equations may be viewed as relationships between the system's output response, the system's inputs at that time, the current state of the system, the system parameters, and time. The state of the system may be thought of as a numerical representation of the dynamically changing configuration of the system.

Simulink provides a graphical editor that allows us to create and connect instances of block types selected from libraries of block types via a library browser. Simulink provides libraries of blocks representing elementary systems that can be used as building blocks. The blocks supplied with Simulink are called built-in blocks. We can also create our own block types and use the Simulink editor to create instances of them in a diagram. Blocks that we create are called custom blocks.

Simulink's primary design goal is to enable the modeling, analysis, and implementation of dynamics systems. Simulink provides many high-level abstractions which facilitate the design of such systems. In addition, Simulink has an open interface which allows easy customization (e.g., the addition of new blocks) and interoperability with other tools. Simulink provides a high-degree of interoperability with MathWorks provided products and other 3rd party products, further simplifying the task of working with dynamic systems.

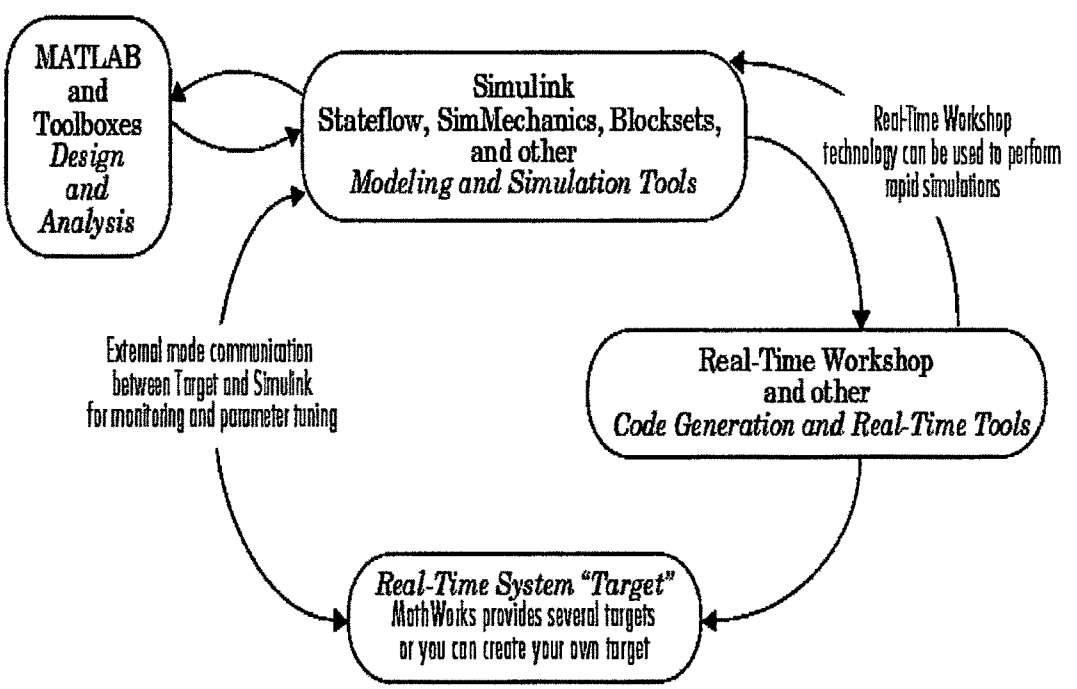


Fig 2.6 Facilities : SIMULINK

Fig 2.6 illustrates facilities and tools provided by the MathWorks that may help in the design, analysis, and implementation of dynamic systems. Simulink provides a richest of modeling capabilities for dynamic systems, which can further be extended by domain specific products such as Stateflow for event driven systems, SimMechanics for modeling physical systems, and many Blocksets such as the DSP Blockset for signal processing. If we are modeling a system that will be deployed outside of the simulation environment on, we can use the Real-Time Workshop and related products to automatically generate highly optimized code for the block diagram.

2.8 Robust Optimal Observer

The section describes design, development and implementation of optimal and Robust observer using SIMULINK.

2.8.1 Optimal observer

Consider the system model is given by,

$$\dot{x}(t) = A\,x(t) + B\,u(t) + w(t)$$
$$y(t) = C\,x(t) + v(t) \qquad \text{----------2.26}$$

where, $w(t) = n\times 1$ plant noise.

$v(t) = m\times 1$ measurement noise.

The Observer equation will be,

$$\dot{\hat{x}}(t) = A\,\hat{x}(t) + B\,u(t) + G\,[\,y(t) - C\,\hat{x}(t)] \qquad \text{----------2.27}$$

$$\therefore\ \dot{\tilde{x}} = \dot{x} - \dot{\hat{x}}$$

$$= A\,x(t) + B\,u(t) + w(t) - A\,\hat{x}(t) - B\,u(t) - G[y - C\hat{x}(t)]$$

$$= A\,[x(t) - \hat{x}(t)] + w(t) - G\,[\,C\,x(t) + v(t) - C\,\hat{x}(t)\,]$$

$$= A\,[x(t) - \hat{x}(t)] - GC\,[x(t) - \hat{x}(t)] + w(t) - Gv(t)$$

$$\therefore\ \dot{\tilde{x}} = (A - GC)\,[x(t) - \hat{x}(t)] + w(t) - Gv(t) \qquad \text{---------- 2.28}$$

Let us consider the estimate of a scalar function,

$$z = k^T x \qquad \text{----------2.29}$$

The estimation error is,

$$\tilde{z} = z - \hat{z}$$

$$= k^T x - k^T \hat{x}$$

$$= k^T \tilde{x} \qquad \text{----------2.30}$$

The zero state response $\tilde{z}_w(t)$ to an input $w(t) = w_o u_o(t)$ is,

$$\tilde{z}_w(t) = k^T e^{(A-GC)t} w_o, \quad t>0 \qquad \text{---------- 2.31}$$

similarly, zero state response $\tilde{z}_v(t)$ to $v(t) = v_o u_o(t)$ is,

$$\tilde{z}_v(t) = - k^T e^{(A-GC)t} G v_o , \quad t>0 \qquad \text{---------- 2.32}$$

From equation 2.31, $\tilde{z}_w(t)$ will be driven rapidly to zero if eigen values of A-GC are large and stable. So G must have large elements. From equation 2.32, large G will give large value of $\tilde{z}_v(t)$ for small t, which is not desirable. So a compromise between speed of response and initial value is done by minimizing the following norm as performance index:

$$J= \int[z^2_w(t) + z^2_v(t)] dt \qquad \text{----------2.33}$$

$$J = \int[k^T e^{(A-GC)t} w_o w_o^T e^{(A-GC)Tt} k + k^T e^{(A-GC)t} G v_o v_o G^T e^{(A-GC)Tt} k] dt$$

$$= \int[k^T e^{(A-GC)t} w_o w_o^T e^{(A-GC)Tt} k + k^T e^{(A-GC)t} G v_o^2 G^T e^{(A-GC)Tt} k] dt \qquad \text{------- 2.34}$$

($\because$ $\tilde{z}_w$ is a scalar hence $\tilde{z}_w^T = \tilde{z}_w$)

$$= \int k^T [e^{(AT-CTGT)Tt} (W + GVG^T) e^{(AT-CTGT)t}] dt\, k$$

[where, $W=w_o w_o^T$ and $V=v_o^2$]

$$= k^T \int [e^{(AT-CTGT)Tt} (W + GVG^T) e^{(AT-CTGT)t}] dt\, k \qquad \text{-------- 2.35}$$

Define Linear Quadratic system as,

$$\dot{\theta} = A^T\theta + C^T\psi \qquad \text{-------- 2.36}$$

A control law $\psi = - G^T\theta$ applied to this system results in

$$\dot{\theta} = (A^T - C^TG^T)\theta \qquad \text{------- 2.37}$$

The response for an initial state $\theta(0) = k$ is $\theta(t) = e^{(AT-CTGT)t} k$ Thus equation 2.35 will be,

$$J = \int \theta^T(t) [W + (G^T)^T V (G^T)] \theta(t) dt \qquad \text{------- 2.38}$$

Minimization of J, with $\theta(t)$ is an LQ problem. The matrix W replaces Q, V replaces R and G^T stands for k.

The solution of LQ problem is

$$G^T = V^{-1} (C^T)^T P \qquad \text{-------- 2.39}$$

where, P represents the solution of Riccati equation,

$$(A^T)^T P + P (A^T) - P (C^T) V^{-1} (C^T)^T P + W = 0 \qquad \text{------- 2.40}$$

Transposition of equation 2.39 yields

$$G = P C^T V^{-1} \qquad (\because \text{P and V are symmetric.})$$

Equation 2.40 can be written as

$$A P + P A^T - P C^T V^{-1} C P + W = 0 \qquad \text{------ 2.41}$$

2.8.2 Robust Optimal Observer

In addition to minimizing above Performance Index (equation 2.33), we can obtain robustness with H^2 or H^∞ controller. Consider the Fig 2.7 which is stable/optimal when LQR controller is used along with Kalman filter to estimate the states of the system.

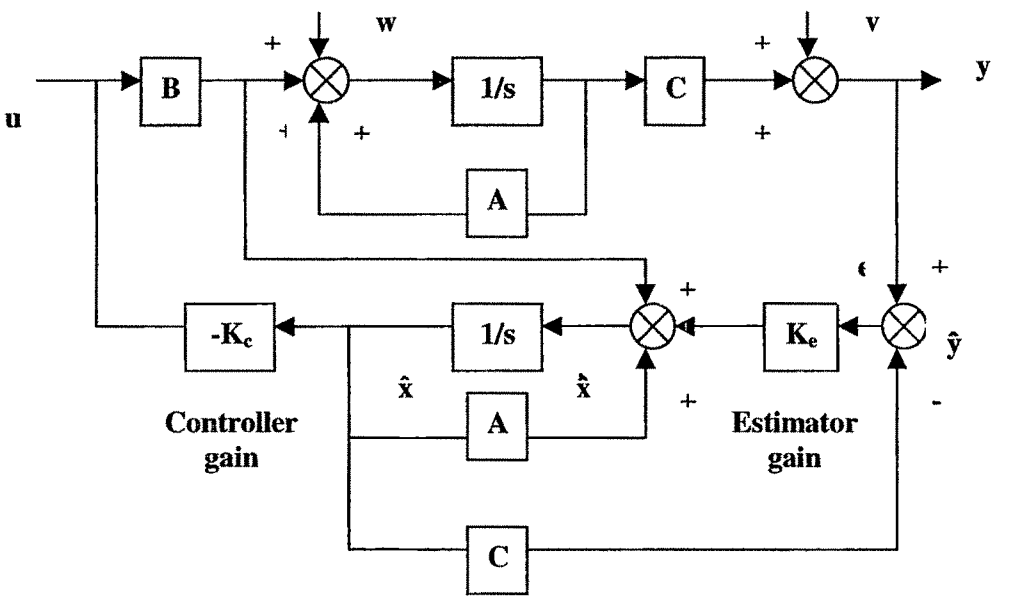


Fig 2.7 Mechanization: Optimal observer

The plant equations are

$$\dot{x}(t) = A\,x(t) + B\,u(t) + w(t)$$
$$y(t) = C\,x(t) + v(t) \qquad \text{----------2.42}$$

Controller equation is $u = -k\,\hat{x}(t)$ ---------- 2.43

where, $k = R^{-1}BP$
where P is solution of Riccati equation,

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

The filter dynamics are

$$\dot{\hat{x}} = A\,\hat{x}(t) + B\,u(t) + L[\,y - C\,\hat{x}(t)\,] \qquad \text{------2.44}$$
$$= \Sigma C^T R^{-1}$$

where Σ is the solution of Riccati equation,

$$AX + \Sigma A^T + Q_0 - \Sigma C^T R_0^{-1} C\Sigma = 0 \qquad \text{--------2.45}$$

Solution of LQG will be

$$H(s) = K\,(sI - A + BK + LC)^{-1} L \qquad \text{-------- 2.46}$$

From the experience it is found that this system may have drawbacks like

- System may fail to work in real environment.
- Solution has lack of robustness.
- System becomes more unstable if more realism is added.

This is because stress is given on optimality, rater than uncertainty. This can be overcome by using H^∞ solution.

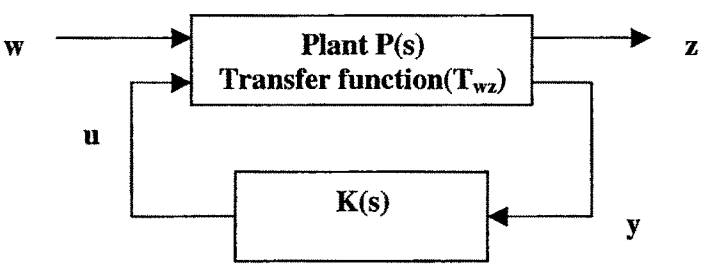


Fig 2.8: H^∞ problem formulation.

So in H^∞ controller design the problem formulation will be like find an internally stabilizing controller [K(s)] for the plant [P(s)], such that H^∞ norm is below a given level.

i.e. $\left\{ \begin{array}{c} \text{Find} \\ K(s) \text{ stabilizing} \end{array} \right\}$ such that $\|T_{wz}\|_\infty < \gamma$

In H^∞ controller, the exogenous inputs (disturbances, command inputs, sensor noise) are collected into one vector; the regulated outputs (control signals, errors) are collected into another vector. The objective is to maintain the peak in the closed loop frequency response of the system below a specified value γ. The optimal solution can be obtained by iterating on γ. The solution involves selecting weights and solving Riccati equations.. *Assumptions* made in designing this controller are:

1. The system is represented in augmented form 2.47 Hence the plant equations will be

$$\begin{aligned} x &= A\,x(t) + B_1\,w(t) + B_2\,u(t) \\ z &= C_1\,x(t) + D_{11}\,w(t) + D_{12}\,u(t) \\ y &= C_2\,x(t) + D_{21}\,w(t) + D_{22}\,u(t) \end{aligned} \qquad \text{----------2.47}$$

In packed matrix form it can be represented by

$$P(s) = \left[\begin{array}{c|cc} A & B_1 & B_2 \\ \hline C_1 & D_{11} & D_{12} \\ C_2 & D_{21} & D_{22} \end{array}\right]$$

2. (A,B_2) and (C_2,A) are detectable.(which is necessary condition for existence of controller.)

3. If dim x = n, dim w = m, dim u = m_2, dim z = p_1, dim y = p_2

   $\Re(D_{12}) = m_2$ and $\Re(D_{21}) = p_2$

   Ensures that controllers are proper and transfer function from (w → y) is non-zero at high frequencies.

4. $$\Re\begin{bmatrix} A\text{-}jwI & B_2 \\ C_1 & D_{12} \end{bmatrix} = n + m_2 \text{ for all frequencies.}$$

   $$\Re\begin{bmatrix} A\text{-}jwI & B_1 \\ C_2 & D_{21} \end{bmatrix} = n + p_2 \text{ for all frequencies.}$$

5. $D_{11} = D_{22} = 0$; simplifies solution.

Fig 2.8 depicts the flow chart for operation of H^{∞} controller

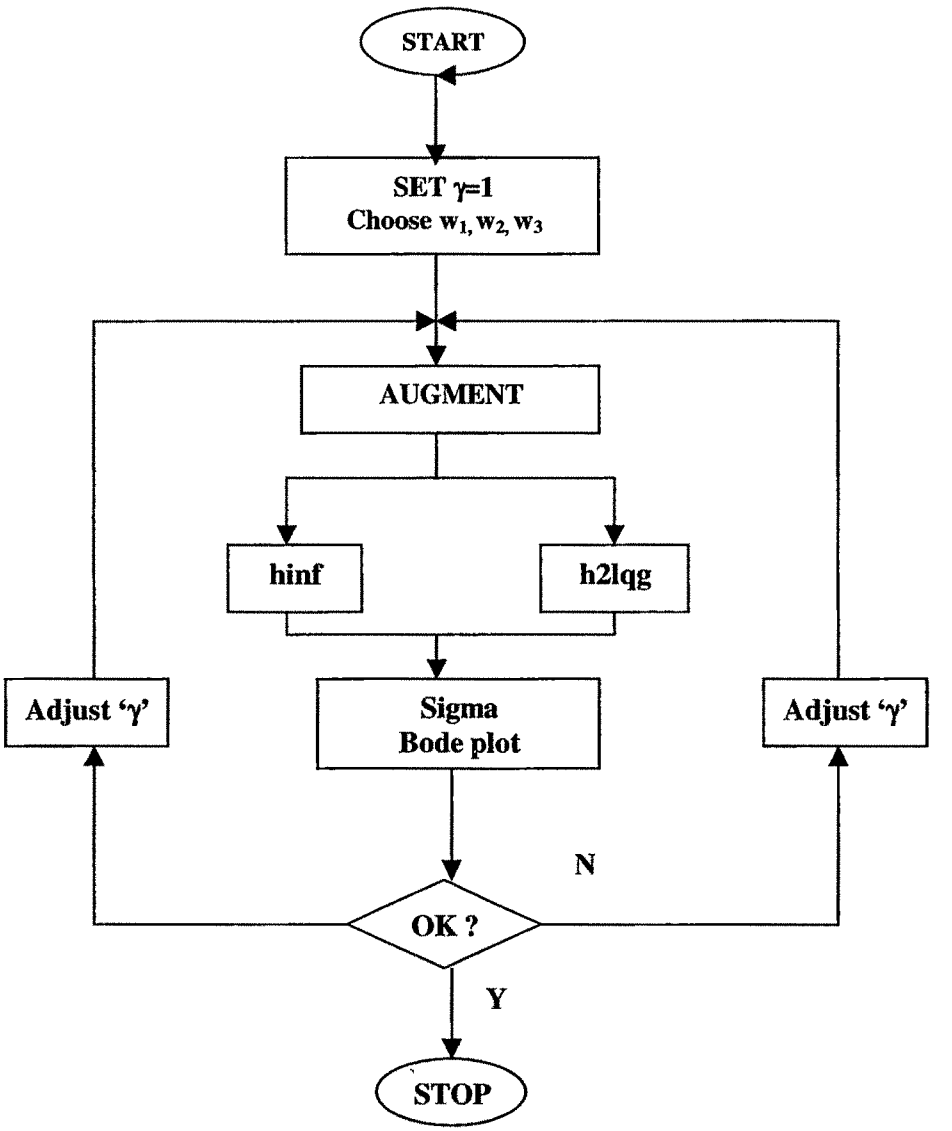


Fig 2.8(a): Flow chart of execution of H^{∞} controller.

Fig 2.9 and Fig 2.10 depicts the block diagrams of system in augmented form and H^{∞} controller.

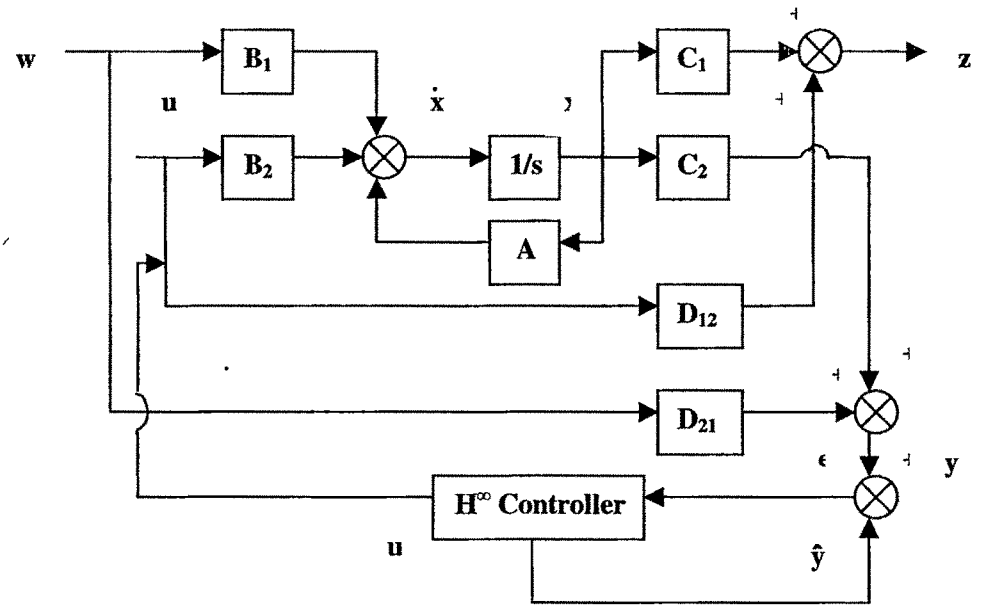


Fig 2.9: System in augmented form with H^{∞} Controller.

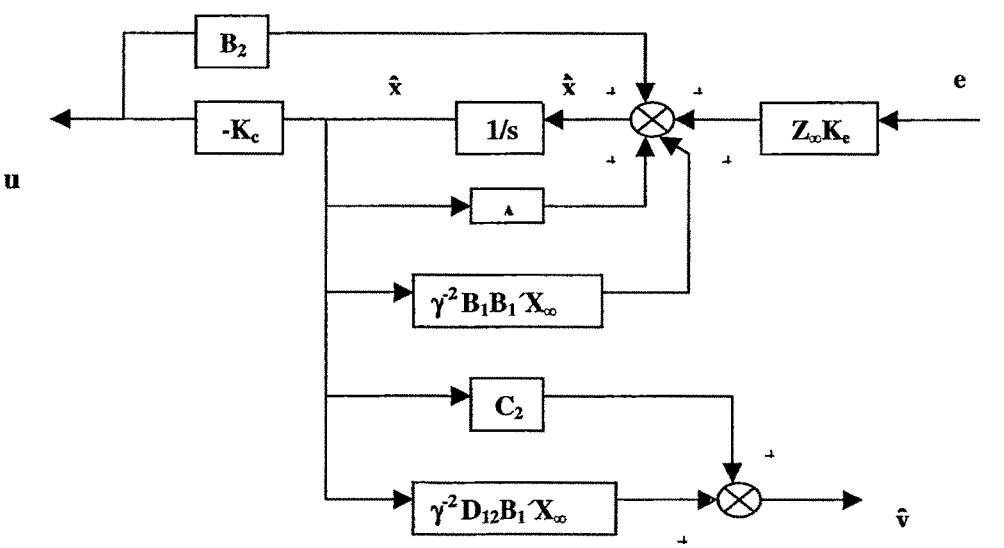


Fig 2.10: Block diagram of H^{∞} Controller.

Control law $u = -K_c\hat{x}$ -------- 2.48

$$\dot{\hat{x}} = A\hat{x} + B_2u + B_1\hat{w} + Z_{\infty}K_e(y - \hat{y}) \quad \text{-------- 2.49}$$

where, $\hat{w} = -\gamma^{-2}B_1{}^{T}X_{\infty}\hat{x}$

$$\hat{y} = C_2\hat{x} + \gamma^{-2}D_{21}B_1'X_\infty\hat{x} \qquad \text{-------- 2.50}$$

Controller gain : $K_c = D_{12}(B_2'X_\infty + D_{12}'C_1)$ -------- 2.51

where, $D_{12} = (D_{12}'D_{12})^{-1}$

Estimator gain : $K_e = (Y_\infty C_2' + B_1D_{21}')D_{21}$ ------- 2.52

where, $D_{21} = (D_{21}'D_{21})^{-1}$

The X_∞ and Y_∞ are solution of RICCATI equations for controller and estimator respectively.

$$X_\infty = \begin{bmatrix} A-B_2\tilde{D}_{12}D_{12}'C_1 & \gamma^{-2}B_1B_1'-B_2\tilde{D}_{12}D_{12}' \\ -\tilde{C}_1'C_1 & -(A-B_2\tilde{D}_{12}D_{12}'C_1)^T \end{bmatrix} \qquad \text{------- 2.53}$$

$$Y_\infty = \begin{bmatrix} A-B_1D_{21}'\tilde{D}_{21}w & \gamma^{-2}C_1'C_1-C_2'\tilde{D}_{21}C_2 \\ -\tilde{B}_1B_1' & -(A-B_1D_{21}'\tilde{D}_{21}C_2) \end{bmatrix} \qquad \text{-------- 2.54}$$

where, $\tilde{C}_1 = (I - D_{12}\tilde{D}_{12}D_{12}')C_1$ ------- 2.55

$$\tilde{B}_1 = [B_1(I - D_{21}'\tilde{D}_{21}D_{21})] \qquad \text{-------- 2.56}$$

And $Z_\infty = (I - \gamma^{-2}Y_\infty X_\infty)^{-1}$ --------- 2.57

$$K(s) = \begin{bmatrix} A - B_2K_c - Z_\infty K_eC_2 + \gamma^{-2}(B_1B_1' - Z_\infty K_eD_{21}B_1')X_\infty & Z_\infty K_e \\ -K_c & 0 \end{bmatrix} \qquad \text{-------- 2.58}$$

2.8.3 SIMULINK Implementation: Robust Adaptive Observer

Fig 2.11 and 2.12 depicts simulation models of augmented space and the H^{∞} controller, represented by block diagrams of fig 2.8 and fig 2.9.

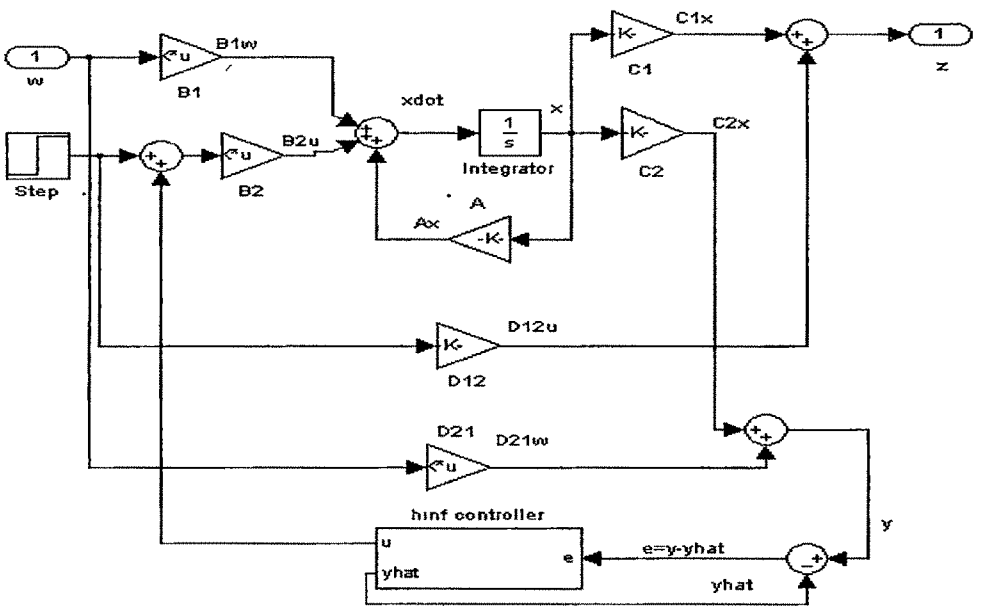


Fig 2.11: Implementation of system in augmented space.

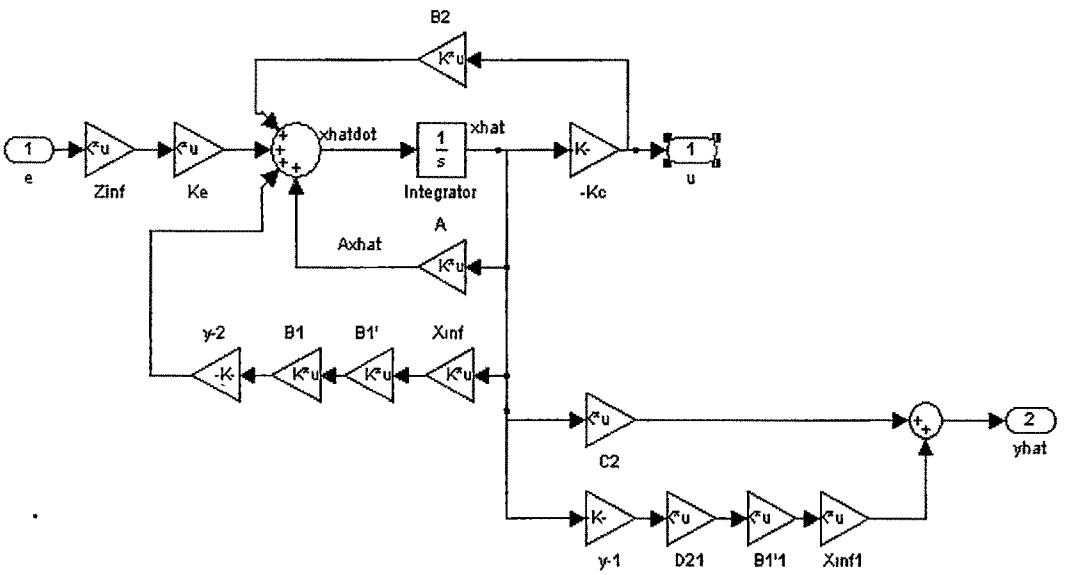
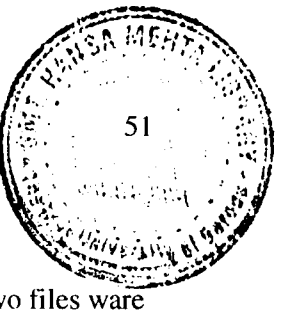


Fig 2.12 Implementation of H^{∞} controller.



2.8.4 APPLICATION:

The simulation study of speed control of Induction motor was carried out. Two files ware developed in MATLAB and a model file in SIMULINK. Table 2.1 shows details of files.

FILE NAME	STORED DATA	PURPOSE
motorparameter.m	Performance parameters of motor like R_s, R_r, L_{ls}, L_{lr}, P, J, X_{lr}, X_{ls}, w_b, T_r, L_r, a, b, c, d matrices of state space representation of motor model.	When we execute this file all the parameters will be stored at workspace.
rhzeros.m	Program for H^{∞} controller with zero padding.	After executing this file we will get optimal value of gamma, xinf, yinf, K_c, K_e, motor model in augmented space and all these values will be stored at workspace.
hinfcontroller.mdl	Simulink model for H^{∞} controller.	Simulating this model we will get robust adaptive observer and during simulation it will use the stored data in workspace.

Table 2.1: Summary of MATLAB/SIMULINK files.

The simulation steps are:

1. *motorparameter.m* file: To initialize the following motor parameters:

```
Rr=0.39;
Rs=0.19;
Lls=0.21e-3;
Llr=0.6e-3;
Lm=4e-3;
Fb=100;
P=4;
J=0.0226;
Lr=Llr+Lm;
```

```
Tr=Lr/Rr;33
Wb=2*pi*Fb;
Xls=Wb*Lls;
Xlr=Wb*Llr;
Xm=Wb*Lm;
Xmstar=1/(1/Xls+1/Xlr+1/Xm);
Vd=2*179.629/pi;

We=[4*pi*Fb]/P
Wr=0.95*We

%calculate a:
a11=[Rs*Wb*[[Xmstar/Xls]-1]]/Xls;
a12=-We;
a13=[[Rs*Wb*Xmstar]/[Xls*Xlr]];
a14=0;
a21=We;
a22=a11;
a23=0;
a24=a13;
a31=[[Rr*Wb*Xmstar]/[Xls*Xlr]];
a32=0;
a33=[Rr*Wb*[[Xmstar/Xlr]-1]]/Xlr;
a34=-[We-Wr];
a41=0;
a42=a31;
a43=-a34;
a44=a33;
a=[a11 a12 a13 a14;a21 a22 a23 a24; a31 a32 a33 a34; a41 a42 a43 a44]

%calculate b:
b=[Wb 0;0 Wb;0 0; 0 0]

%calculate c and d:
c=[1 0 0 0; 0 1 0 0]
d=[0 0;0 0]
w1=zeros(4,4);
w2=ones(4,4);
w3=ones(4,4);
%w1=[2.5e-5*[0.01 0 02 0.03 0.04 ];4e-6*[0.01 0.2 0.3 0.4 ];0.1 0.2 0.3 0.4 ; 10 20 30 40
];

%w2=[1e-3*[1 0.1 0.2 0.3 ]; 5e-4*[0.01 0.02 0.03 0.04] ; 2e-1*[0.1 0.2 0.3 0.4 ]; 0.001
0.009 0.008 0.007 ];
%w3=[1 0 0 0 ; 0 0 0 40 ; 1 2 3 4 ; 0.09 0.08 0.07 0.06 ];
```

2. rhzeros.m file: To generate H^{∞} Controller parameters

```
%num=input('The co-eff of numerator are: ');
%den=input('The co-eff of denominator are : ');
%printsys(num,den);
%[a, b, c, d]=tf2ss(num,den)
sys=mksys(a,b,c,d);
%%% input different noises:

%w1=input('The value of w1 is : ');
%w2=input('The value of w2 is : ');
%w3=input('The value of w3 is : ');

%%% get system in augmented form:
[A,B1,B2,C1,C2,D11,D12,D21,D22]=augtf(a,b,c,d,w1,w2,w3);

TSS=augtf(sys,w1,w2,w3);

%%% obtain optimal value of gamma.
[optigamma,ss_f,ss_cl]=hinfopt(TSS,1);

%%% get values of xinf and yinf = solution of
%%% controller and estimator riccati equations:
xinf=branch(ss_f)
yinf=branch(ss_cl)

%%% calculate controller gain kc:
x=inv(D12'*D12);
x1=B2'*xinf;
x2=D12'*C1;
Kc=x*(x1+x2);

%%% calculate estimator gain ke:
y=inv(D21*D21')
y3=[C2' zeros(10,8); zeros(10,10)]
y1=yinf*y3
y2=B1*D21'
y4=[y2 zeros(10,8); zeros(10,10)]
Ke=(y1+y4)*[y zeros(2,18);zeros(8,20)];

%%% calculate zinf:
%yinf1=[yinf zeros(20,10)];
xinf1=[xinf zeros(10,10);zeros(10,20)]
z=optigamma^-2*yinf*xinf1;
z1=eye(20,20)-z;
zinf=inv(z1);
```

- **The partial results are listed below:**

<< H-Infinity Optimal Control Synthesis >>

No	Gamma	D11<=1	P-Exist	P>=0	S-Exist	S>=0	lam(PS)<1	C.L.
1	1.0000e+000	OK	FAIL	FAIL	OK	OK	OK	STAB
2	5.0000e-001	OK	FAIL	OK	OK	OK	OK	UNST
3	2.5000e-001	OK	FAIL	FAIL	OK	OK	OK	STAB
4	1.2500e-001	OK	FAIL	FAIL	OK	OK	OK	UNST
5	6.2500e-002	OK	FAIL	FAIL	OK	OK	OK	UNST
6	3.1250e-002	OK	FAIL	FAIL	OK	OK	OK	UNST
7	1.5625e-002	OK	FAIL	FAIL	OK	OK	OK	UNST
8	7.8125e-003	OK	OK	FAIL	OK	OK	OK	UNST
9	3.9063e-003	OK	FAIL	OK	OK	OK	OK	STAB
10	1.9531e-003	OK	FAIL	FAIL	OK	OK	OK	UNST
11	9.7656e-004	OK	OK	FAIL	OK	OK	OK	UNST
12	4.88328e-004	OK	FAIL	FAIL	OK	OK	OK	UNST
13	2.4414e-004	OK	FAIL	FAIL	OK	OK	OK	UNST
14	1.2207e-004	OK	OK	FAIL	OK	OK	OK	UNST
15	6.1035e-005	OK	FAIL	OK	OK	OK	OK	STAB
16	3.0518e-005	OK	FAIL	FAIL	OK	OK	OK	STAB
17	1.5259e-005	OK	FAIL	FAIL	OK	OK	OK	UNST
18	7.6294e-006	OK	FAIL	FAIL	OK	OK	OK	STAB
19	3.8147e-006	OK	FAIL	FAIL	OK	OK	OK	UNST
20	1.9073e-006	OK	OK	FAIL	OK	OK	OK	UNST
21	9.5367e-007	OK	FAIL	FAIL	OK	OK	OK	STAB
22	4.7684e-007	OK	FAIL	FAIL	OK	OK	OK	UNST
23	2.3842e-007	OK	OK	FAIL	OK	OK	OK	UNST
24	1.1921e-007	OK	FAIL	FAIL	OK	OK	OK	STAB
25	5.9605e-008	OK	FAIL	FAIL	OK	OK	OK	UNST
26	2.9802e-008	OK	FAIL	FAIL	OK	OK	OK	UNST
27	1.4901e-008	OK	FAIL	FAIL	OK	OK	OK	UNST
28	7.4506e-009	OK	FAIL	FAIL	OK	OK	OK	UNST
29	3.7253e-009	OK	OK	FAIL	OK	OK	OK	STAB
30	1.8626e-009	OK	FAIL	FAIL	OK	OK	OK	UNST
31	9.3132e-010	OK	FAIL	FAIL	OK	OK	OK	UNST
32	4.6566e-010	OK	FAIL	FAIL	OK	OK	OK	UNST
33	2.3283e-010	OK	FAIL	OK	OK	OK	OK	UNST
34	1.1642e-010	OK	FAIL	OK	OK	OK	OK	STAB
35	5.8208e-011	OK	FAIL	FAIL	OK	OK	OK	UNST
36	2.9104e-011	OK	FAIL	OK	OK	OK	OK	UNST
37	1.4552e-011	OK	FAIL	FAIL	OK	OK	OK	UNST
38	7.2760e-012	OK	FAIL	FAIL	OK	OK	OK	UNST
39	3.6380e-012	OK	FAIL	FAIL	OK	OK	OK	UNST
40	1.8190e-012	OK	FAIL	FAIL	OK	OK	OK	STAB
41	9.0949e-013	OK	FAIL	OK	OK	OK	OK	UNST
42	4.5475e-013	OK	FAIL	OK	OK	OK	OK	STAB

43	2.2737e-013	OK	OK	OK	OK	OK	OK	STAB
44	3.4106e-013	OK	FAIL	FAIL	OK	OK	OK	STAB
45	2.8422e-013	OK	FAIL	FAIL	OK	OK	OK	UNST
46	2.5580e-013	OK	FAIL	OK	OK	OK	OK	UNST
47	2.4158e-013	OK	OK	OK	OK	OK	OK	UNST
48	2.3448e-013	OK	FAIL	OK	OK	OK	OK	UNST
49	2.3093e-013	OK	FAIL	FAIL	OK	OK	OK	STAB
50	2.2915e-013	OK	OK	FAIL	OK	OK	OK	UNST

Iteration no. 43 is your best answer under the tolerance: 0.0100

3. **hinficontroller.mdl:** Fig 2.14 depicts estimated output of H^{∞} controller (y-hat) on execution of the SIMULINK model Fig 2.13.

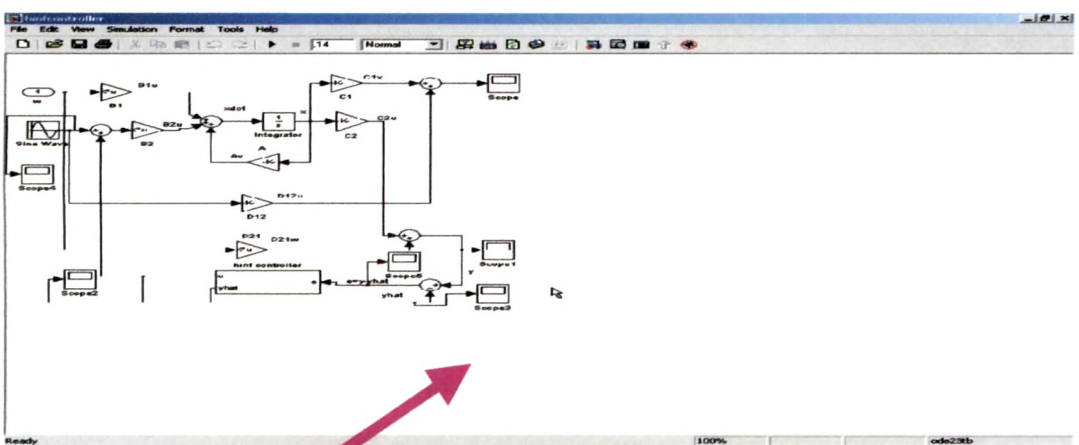


Fig 2.13 SIMULINK MODEL: H^{∞} Controller

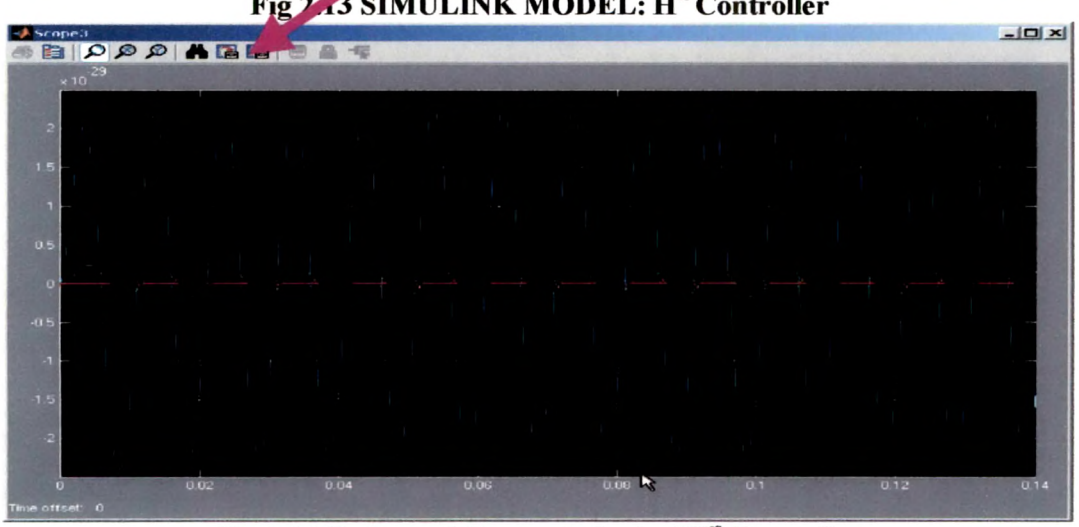


Fig 2.14 Estimated Output (y-hat) of H^{∞} Controller

The design, development and Implementation of an observer-estimator for a time varying system is discussed. The design and simulation of robust estimator using H^{∞} Controller using SIMULINK is also attempted.