ROBUST CONTROLLER DESIGN
FOR TIME-DELAY SYSTEMS

Master of Science Thesis

Sana ELMUHANDES

Eskişehir 2018

ROBUST CONTROLLER DESIGN FOR TIME-DELAY SYSTEMS

Sana ELMUHANDES

MASTER OF SCIENCE THESIS

Graduate School of Sciences
Electrical and Electronics Engineering Program
Supervisor: Prof. Dr. Altuğ İFTAR

Eskişehir
Anadolu University
Graduate School of Sciences
July 2018

FINAL APPROVAL FOR THESIS

This thesis titled "ROBUST CONTROLLER DESIGN FOR TIME-DELAY SYSTEMS" has been prepared and submitted by Sana ELMUHANDES in partial fullfillment of the requirements in "Anadolu University Directive on Graduate Education and Examination" for the Degree of Master of Science in Electrical and Electronics Engineering Department has been examined and approved on 19/07/2018.

Committe Members	Title Name Surname	Signature
Member (Supervisor)	: Prof. Dr. Altuğ İFTAR	
Member	: Prof. Dr. Osman PARLAKTUNA	
Member	: Asist. Prof. Dr. Hakkı Ulaş ÜNAL	

Prof. Dr. Ersin YÜCEL
Director of Graduate School of Science

ABSTRACT

ROBUST CONTROLLER DESIGN FOR TIME-DELAY SYSTEMS

Sana ELMUHANDES

Electrical and Electronics Engineering Program

Anadolu University, Graduate School of Sciences, July 2018

Supervisor: Prof. Dr. Altuğ İFTAR

The main objective of this thesis is to present a method to robustly stabilize retarded linear time-invariant time-delay systems with a presence of multiple uncertain time delays. As a result, the proposed method obtains the best guaranteed exponential decay rate. In this method, the spectral abscissa for the worst time delay is minimized using a formulated minimax dynamic game problem with two decision makers. Algorithms are proposed to solve the formulated problem depending on steepest gradient descent/ascent. Also, an initialization procedure for designing controller's free parameters is proposed. Both minimax game and initialization procedure used to build a complete robust output feedback controller design algorithm. A software package implements the controller design algorithm created. Illustrative numerical examples are solved.

Keywords: Time-delay systems, Eigenvalues, Uncertain time delays, Minimax problem, Robust stability, Feedback controller.

ÖZET

ZAMAN GECİKMELİ SİSTEMLER İCİN GÜRBÜZ DENETLEYİCİ TASARIMI

Sana ELMUHANDES

Elektrik-Elektronik Mühendisliği Anabilim Dalı

Anadolu Üniversitesi, Fen Bilimleri Enstitüsü, Temmuz 2018

Danışman: Prof. Dr. Altuğ İFTAR

Bu tezin ana amacı, doğrusal zamanla değişmeyen geri-tipli (retarded) zaman gecikmeli sistemleri çoklu belirsiz zaman gecikmelerinin varlığı durumunda gürbüz kararlı kılacak bir yöntem sunmaktır. Sonuç olarak, önerilen yöntem en iyi garantili üstel yaklaşım oranını elde etmektedir. Bu yöntemde, iki karar vericiyle formüle edilmiş en küçük-en büyük dinamik oyun problemi kullanılarak en kötü zaman gecikmesi için spektral apsis en aza indirgenmiştir. Algoritmalar, formüle edilmiş problemi en dik gradyan inişine/çıkışına bağlı olarak çözmek için önerilmiştir. Ayrıca, denetleyicinin serbest parametrelerinin tasarlanması için bir başlangıç prosedürü önerilmiştir. Hem en küçük-en büyük oyunu hem de başlangıç prosedürü, bütün bir gürbüz çıktı geri beslemeli denetleyici tasarım algoritması oluşturmak için kullanılmaktadır. Önerilen denetleyici tasarım algoritmasını uygulayan bir yazılım paketi oluşturulmuştur. Açıklayıcı sayısal örnekler çözülmüştür.
Anahtar Kelimeler: Zaman gecikmeli sistemler, Özdeğerler, Belirsiz zaman gecikmeleri, En küçük-en büyük problemi, Gürbüz kararlılık, Geri-beslemeli denetleyici.

ACKNOWLEDGMENT

I would like to express my appreciation to my project supervisor **Prof. Dr. Altuğ İFTAR** for his continuous support, motivation, and guidance throughout the project. It has been a great learning experience to work with him and I thank him for giving me this opportunity. I would like to appreciate his insights, suggestions, and ideas which have steered me in the right direction. I am thankful to **Hüseyin Ersin EROL** and **Süleyman Mert ÖZER** for giving valuable support throughout my project work.

Finally, I thank my **family** for their love, affection, and support for everything in my life. I thank my **father** who constantly encouraged me to take up this. **My husband** Without his support, I couldn't have even imagined achieving this goal.

Sana ELMUHANDES

19/07/2018

STATEMENT OF COMPLIANCE WITH ETHICAL PRINCIPLES AND RULES

I hereby truthfully declare that this thesis is an original work prepared by me; that I have behaved in accordance with the scientific ethical principles and rules throughout the stages of preparation, data collection, analysis and presentation of my work; that I have cited the sources of all the data and information that could be obtained within the scope of this study, and included these sources in the references section; and that this study has been scanned for plagiarism with "scientific plagiarism detection program" used by Anadolu University, and that "it does not have any plagiarism" whatsoever. I also declare that, if a case contrary to my declaration is detected in my work at any time, I hereby express my consent to all the ethical and legal consequences that are involved.

Sana ELMUHANDES

TABLE OF CONTENTS

Page

LIST OF FIGURES

Page

LIST OF NOTATIONS

$\mathbb{R}$	Real numbers
$\mathbb{R}_+$	Non-negative real numbers
$\mathbb{C}$	Complex numbers
$\mathrm{Re}(s)$	Real part of s, $s \in \mathbb{C}$
$Im(s)$	Imaginary part of s, $s \in \mathbb{C}$
i	Imaginary unit, $\mathrm{i}^2 = -1$
$\mathbb{C}_\mu^+$	$\{s \in \mathbb{C} \mid \mathrm{Re}(s) \geq \mu\}$, For $\mu \in \mathbb{R}$
$\mathbb{C}_\mu^-$	$\{s \in \mathbb{C} \mid \mathrm{Re}(s) < \mu\}$, For $\mu \in \mathbb{R}$
$\mathbb{C}^-$	Left-half complex plane
$[a, b]$	$\{h \in \mathbb{R} \mid a \leq h \leq b\}$, For $a, b \in \mathbb{R}$ with $a < b$
$\mathbb{R}^k$	Spaces of k-dimensional real vectors, k is a positive integer
$\mathbb{C}^k$	k-dimensional complex vectors, k is a positive integer
$\mathbb{R}^{k\times l}$	$k \times l$-dimensional real matrices, k and l are positive integers
I	Identity matrix of appropriate dimensions
0	Zero matrix of appropriate dimensions
I_k	$k \times k$ identity matrix
0_k	$k \times k$ zero matrix
$0_{k\times l}$	$k \times l$ zero matrix
$(\cdot)'$	Transpose of $(\cdot)$
$(\cdot)^*$	Complex-conjugate transpose of $(\cdot)$
$\det(\cdot)$	Determinant of $(\cdot)$
$\mathrm{rank}(\cdot)$	Rank of $(\cdot)$
$\dot{x}$	Derivative of x, $x : \mathbb{R} \to \mathbb{R}^n$

1. INTRODUCTION

1.1. Overview and Motivation

Delays appear frequently in real-world engineering systems [1]. In many applications, a response does not take place instantaneously due to different reasons. One of the reasons is the flowing of information between the process and the controller, this physical distance causes a time delay. Another reason is the controller itself when it needs a long time relatively to finish computation, see [2]. Dynamical systems can be classified into two classes: finite-dimensional systems and infinite-dimensional systems. State of an infinite-dimensional system cannot be represented at any time by a finite set of values like finite-dimensional systems which can be modeled by ordinary differential equations. On the other hand, delay differential equations are used to model time-delay systems.

Since the state of a time-delay system cannot be represented by a finite number of state variables, these systems are in the class of infinite-dimensional systems [3]. They are described by delay differential equations such that some data in the past is counted and presented on its model. When the system dynamics can be described by a delay differential equation where all derivatives of the state variables appear with delay free terms, such a system is named as a retarded time−delay system [4]. Retarded type linear time-invariant (LTI) time-delay system has special properties. Even it has an infinite number of modes, it has a finite number of modes in $\mathbb{C}_{\mu}^{+}$ to the right of $\mu \in \mathbb{R}$. The retarded LTI time-delay system modes change continuously with respect to changes in time delays and feedback controller parameters, see [5].

The transfer function of time-delay systems cannot be written as a ratio of two polynomials of s, it contains time-delay element which is represented as e^{-sh}, where $h > 0$ is the time delay. Time delays are often a source of instability and poor performance also greatly increase the difficulty of stability analysis and control design [1]. In this manner, time delay should be considered while designing the controller. To study the stability of such a system by Routh-Hurwits or Kharitonov stability test or root locus technique, an approximation for delay element is needed;

for exact known delay time, Pade approximation algorithm is used. Nyquist stability criterion is valid for large scale of time-delay systems without the need of approximation [2]. However, approximation approaches constitute limitations in the accuracy can lead to instability in the system and include non-minimum phase and thus high gain problems [6]. Since the introduction of the Smith predictor [7], many different approaches, such as those based on $\mathcal{H}_\infty$-optimization [8–14] or Lyapunov methods [15–18] have been proposed for designing controllers for time-delay systems. However, those methods need complex formulations.

Recently, control methods for LTI time-delay systems which are based on eigenvalues approach were widely used. Approaches based on eigenvalues solve the problem of having infinite system modes and a limited number of freedom in the controller parameters, by controlling the rightmost mode. One of the methods is continuous pole placement method. This method was first proposed in [19] for static feedback controller, such that a finite-dimensional controller designed for an infinite-dimensional problem. Continuous pole placement method is based on the fact that there is a finite number of unstable eigenvalues and that the eigenvalues move continuously with respect to changes in the feedback gain. The continuous pole placement strategy is to apply small changes to the feedback controller gain to shift the unstable eigenvalues to the left half plane. This method later got extended to neutral systems and then applied on large scale time-delay systems, see [20–22].

Another used method is the optimization. Optimization methods are based on the ability to stabilize the system with respect to system parameters by tuning the parameters to minimize the real part of the rightmost characteristic root or the spectral abscissa. In other words, it pushes the roots as far as possible to the left-half complex plane. Since spectral abscissa is, in general, a nonsmooth and nonconvex function, a nonsmooth optimization approach was used in [23] to stabilize time-delay systems, and later it is extended to neutral time-delay systems and applied on large scale time-delay systems, see [24, 25]. Recently, a new approach is proposed to robustly stabilize time-delay systems based on optimization methods [26].

During the process of deriving a mathematical model for the plant usually, a series of assumptions and simplifications are made [2]. Since the derived system (nominal system) is not exactly the same as the true physical system, it may contain uncertainty in time delays. In this case, controlling the nominal system is not sufficient and all possibilities of time delays should be considered. This will lead to a robustly stable system even in occurrence of the worst time delay. Feedback is used in control systems to change the dynamics of the system (usually to make the response stable and sufficiently fast) and to reduce the sensitivity of the system to signal uncertainty (disturbances) and model uncertainty [27].

In this work, an output feedback controller design to robustly stabilize LTI retarded time-delay systems will be proposed. The approach depends on placing the right-most eigenvalue of the system as far as possible in the left half plane such that it achieves the best guaranteed exponential decay rate for all delays possibilities. To achieve this aim, a dynamic non-cooperative mini-max game problem is formulated and solved depending on steepest gradient descent/ascent method.

1.2. Thesis Outline

In Section 2.1 a brief introduction for time-delay system is given; then the steepest descent/ascent as an optimization method is discussed in Section 2.2. The problem is stated in Chapter 3. The controller design method and the algorithms to implement it are presented in Chapter 4. Then, numerical examples are solved in Chapter 5; followed by a conclusion in Chapter 6. The developed Matlab codes are included in Appendix.

2. BACKGROUND

2.1. Time-delay Systems

A system is said to have a delay when the rate of variation in the system state depends on past states. Such a system is called a time−delay system [1]. Those systems are classified as infinite dimensional systems and modeled using delay differential equations. Time delay systems generally can be classified into two main types: retarded type time-delay systems, and neutral type time-delay systems. The main difference between neutral and retarded systems is that in retarded systems all the states derivative are free from delays. In this thesis, retarded LTI time-delay systems are considered. Such systems can be described as:

$$\dot{x}(t) = \sum_{l=0}^{\nu} A_l x(t - h_l) \tag{2.1}$$

where $x(t) \in \mathbb{R}^n$ is the state at time t, $A_l \in \mathbb{R}^{n\times n}$, is a constant matrix, and $h_l \in \mathbb{R}_+$, $l = 1, \ldots, \nu$, are the time-delays. And $h_0 := 0$ for notational convenience; i.e., $l = 0$ in (2.1) corresponds to the delay-free part of the system. It is proved in [5] that the eigenvalues of the system (2.1) are the system characteristic roots. The characteristic matrix of system (2.1) described as:

$$\Gamma(s) = sI - \sum_{l=0}^{\nu} A_l e^{-sh_l} \tag{2.2}$$

The roots of $det(\Gamma(s))$ are the system characteristic roots and can be called the open-loop modes. It is important to point that even the system has infinitely many modes, it has a finite number of modes in $\mathbb{C}_\mu^+$ to the right of any given $\mu \in \mathbb{R}$. Such modes are called μ-modes. Retarded type time-delay system's characteristic roots behave continuously with respect to variations in the system matrices and delays. All μ-modes in any right half plane to the right of $\mu \in \mathbb{R}$ can be calculated by the spectral discretization method in [28]. Characteristic roots in any bounded region can be computed using the quasi-polynomial root finder method in [29].

Similarly to other LTI systems, stability of LTI time-delay systems can be studied in the frequency domain. LTI time-delay system is considered stable if and

only if its all modes are taking place in the left-half complex plane, otherwise it is considered unstable system. The system response speed can be determined by finding the real part of the rightmost mode, it can be called the system exponential decay rate. The real part of the rightmost mode can be defined as the spectral abscissa and described as:

$$c := \max\{\mathrm{Re}(s) \mid \det(\Gamma(s)) = 0\} \tag{2.3}$$

2.2. Optimization

Optimization is a procedure to find the minimum or maximum of a function of several variables, minimization problem can be described as:

$$min_X f(X)$$

And the maximization problem described as:

$$max_X f(X)$$

Cauchy was the first to make use of the negative gradient direction in 1847 for minimization problems [30]. In this method an initial trail point X_1 is chosen, then it is iteratively moved along the steepest descent direction until the optimum point is found. This method requires that the function is continuous and differentiable at all the points. The updating rule can be described as:

$$X_{i+1} = X_i - \beta_0 \nabla f(X_i) \tag{2.4}$$

where β_0 is a prechosen small positive step size, and $\nabla f(X_i)$ is the gradient of function f with respect to parameter X_i.

Oppositely the maximizing method used the positive gradient direction, and similarly, an initial trail point X_1 is chosen, then it is iteratively moved along

the steepest ascent direction until the optimum point is found. The updating rule can be described as:

$$X_{i+1} = X_i + \beta_1 \nabla f(X_i) \tag{2.5}$$

where β_1 is a prechosen small positive step size.

This optimization method still valid to minimize/ maximize spectral abscissa since it is differentiable almost everywhere. But it is important to point that the spectral abscissa is a non-convex function with respect to system parameters, it may converge to local min/max.

3. PROBLEM STATEMENT

Consider an LTI retarded time-delay system which contains ν independent time-delays in its state, input, and/or output, which is described as:

$$\dot{x}(t) = \sum_{l=0}^{\nu} A_l x(t-h_l) + \sum_{l=0}^{\nu} B_l u(t-h_l) \tag{3.1}$$

$$y(t) = \sum_{l=0}^{\nu} C_l x(t-h_l) \tag{3.2}$$

where $x(t) \in \mathbb{R}^n$, $u(t) \in \mathbb{R}^r$, and $y(t) \in \mathbb{R}^q$ are, respectively, the state, input, and output vectors at time t, $A_l \in \mathbb{R}^{n\times n}$, $B_l \in \mathbb{R}^{n\times r}$, and $C_l \in \mathbb{R}^{q\times n}$, $l = 0, \ldots, \nu$, are constant matrices, and $h_l \in \mathbb{R}_+$, $l = 1, \ldots, \nu$, are the time-delays. And $h_0 := 0$ for notational convenience; i.e., $l = 0$ in (3.1)–(3.2) corresponds to the delay-free part of the system. In this work, time-delays are assumed to be uncertain, but in some known interval. Consider that each time-delay $h_l \in \mathbb{R}_+$, $l = 1, \ldots, \nu$, is uncertain with a known minimal value $h_l^{\min} \geq 0$ and a known maximal value $h_l^{\max} \geq h_l^{\min}$; i.e., actual value of h_l satisfies

$$0 \leq h_l^{\min} \leq h_l \leq h_l^{\max} \tag{3.3}$$

For notational convenience, the *time-delay vector*:

$$h := \begin{bmatrix} h_1 & \cdots & h_\nu \end{bmatrix}' \in \mathbb{R}^\nu$$

and the set

$$\Omega := [h_1^{\min}, h_1^{\max}] \times \cdots \times [h_\nu^{\min}, h_\nu^{\max}] \subset \mathbb{R}^\nu$$

Then the Output feedback control law is considered

$$\begin{aligned} \dot{z}(t) =& Fz(t) + Gy(t) \\ u(t) =& Hz(t) + Ky(t) \end{aligned} \tag{3.4}$$

where $z(t) \in \mathbb{R}^m$ is the state of the controller and F, G, H, and K are real constant matrices. When the controller dimension m is zero, control law become a static feedback:

$$u(t) = Ky(t) \tag{3.5}$$

Since there is no direct connection between the input and output in the system (3.1–3.2) under control law (3.4), then not only the open-loop system is a retarded system but also the closed-loop system is a retarded system. It is useful to find the transfer function matrix $P(s)$ and then find the blocking zeros of $P(s)$, those blocking zeros will be used later in controller initialization procedure. Transfer function matrix can be described as:

$$P(s) = (\sum_{l=0}^{\nu} C_l e^{-sh_l})(sI - (\sum_{l=0}^{\nu} A_l e^{-sh_l}))(\sum_{l=0}^{\nu} B_l e^{-sh_l}) \tag{3.6}$$

To find the $P(s)$ blocking zeros in any bounded region quasi polynomial root finder method in [29] can be used. Let us define the blocking zero z to be $\|P(z)\| = 0$ such that z is a zero for all elements in the transfer function matrix.

3.1. Structure of the Controller Matrices

Using multivariable canonical forms [31] reduces the number of free parameters [21, 24]. Following the multivariable canonical forms in [31]. Such that the multivariable controllable canonical form is used when the number of outputs is more than the number of inputs and its matrices constructions are as following:

$$F = \begin{cases} \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{m,1} & \cdots & f_{m,m} \end{bmatrix} & m \leq q, \\ \begin{bmatrix} 0_{(m-q)\times q} & & I_{m-q} \\ f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{q,1} & \cdots & f_{q,m} \end{bmatrix} & m > q. \end{cases}$$

$$G := \begin{cases} \begin{bmatrix} & g_{1,1} & \cdots & g_{1,q-m} \\ I_m & \vdots & \ddots & \vdots \\ & g_{m,1} & \cdots & g_{m,q-m} \end{bmatrix} & m < q, \\ I_q & m = q, \\ \begin{bmatrix} 0_{(m-q)\times q} \\ I_q \end{bmatrix} & m > q. \end{cases}$$

where m is the controller dimension and q is the outputs number. The free parameters are $f_{i,j}$, $g_{i,j}$, and the elements of H and K matrices. The observable multivariable canonical form is used when the number of inputs is more than the number of outputs and its matrices constructions are as following:

$$F = \begin{cases} \begin{bmatrix} f_{1,1} & \cdots & f_{1,m} \\ \vdots & \ddots & \vdots \\ f_{m,1} & \cdots & f_{m,m} \end{bmatrix} & m \leq r, \\ \begin{bmatrix} 0_{p\times(m-r)} & f_{1,1} & \cdots & f_{1,r} \\ & \vdots & \ddots & \vdots \\ I_{m-r} & f_{m,1} & \cdots & f_{m,r} \end{bmatrix} & m > r. \end{cases}$$

$$H := \begin{cases} \begin{bmatrix} & I_m & \\ h_{1,1} & \cdots & h_{1,m} \\ \vdots & \ddots & \vdots \\ h_{r-m,1} & \cdots & h_{r-m,m} \end{bmatrix} & m < r, \\ I_r & m = r, \\ \begin{bmatrix} 0_{r\times(m-r)} & I_r \end{bmatrix} & m > r. \end{cases}$$

where m is the controller dimension and r is the inputs number. The free parameters are $f_{i,j}$, $h_{i,j}$, and the elements of G and K matrices. The controller parameters number is $(m + r) \times (q + m)$ and by constructing the controller matrices in its

multivariable canonical forms the number of free parameters will be $m(q+r)+qr$ so it is reduced by m^2. The controller's free parameters are collected in one vector p.

$$p := \begin{bmatrix} p_1 & p_i & \cdots & p_k \end{bmatrix}',$$

where $p \in \mathbb{R}^{m(q+r)+(qr)}$, $i = 1, \cdots, k$ and $k = m(q+r)+(qr)$.

3.2. Closed-loop System

The overall closed-loop system can be described as

$$E_0 \dot{\eta}(t) = \mathcal{A}_0(p)\eta(t) + \sum_{i=1}^{v} \mathcal{A}_i \eta(t-h_i) \tag{3.7}$$

where $\eta(t) := \begin{bmatrix} x(t)' & y(t)' & z(t)' & u(t)' \end{bmatrix}'$, and

$$E_0 := \begin{bmatrix} I_n & 0 & 0 & 0 \\ 0 & 0_q & 0 & 0 \\ 0 & 0 & I_m & 0 \\ 0 & 0 & 0 & 0_r \end{bmatrix}, \quad \mathcal{A}_0(p) := \begin{bmatrix} A_0 & 0 & 0 & B_0 \\ C_0 & -I_q & 0 & 0 \\ 0 & G & F & 0 \\ 0 & K & H & -I_r \end{bmatrix},$$

and, for $i = 1, \ldots, v$,

$$\mathcal{A}_i := \begin{bmatrix} A_i & 0 & 0 & B_i \\ C_i & 0_q & 0 & 0 \\ 0 & 0 & 0_m & 0 \\ 0 & 0 & 0 & 0_r \end{bmatrix}.$$

3.3. Stability of Closed-loop System

The closed-loop characteristic matrix can be described as

$$\Gamma(s,p,h) = sE_0 - \mathcal{A}_0(p) - \sum_{j=1}^{v} \mathcal{A}_j e^{-sh_j} \tag{3.8}$$

And the *spectral abscissa* of this closed-loop system can then be defined as:

$$c(p,h) := \max\{\mathrm{Re}(s) \mid \det(\Gamma(s,p,h)) = 0\} \tag{3.9}$$

It is well known that the system (3.1–3.2) under control law (3.4) is stable if its all modes are in $\mathbb{C}^-$. The system is μ stable if $c(p,h) < \mu$. In other words, we can study the stability of retarded LTI time-delay systems in frequency domain similarly to other systems without delays, by finding the characteristic roots in the right half plane, if there is any root, the system is unstable. Otherwise, the system is considered stable.

3.4. Robust Stability

Representing the plants into mathematical model consists of a chain of assumptions and approximations and that may results in uncertainty in time delays, neglecting those uncertainties may affect the system stability. A reasonable formulation of this problem is to represent the true physical plant not by a single nominal but instead by a family $\mathbb{F}$ of possible plant models [32].

If it is possible to find a single controller which stabilizes every plant in family $\mathbb{F}$ then this controller called robustly stabilizing controller and the family of plant $\mathbb{F}$ is called robustly stabilized. The system (3.1–3.2) under the control law (3.4) is said to be robustly stable if and only if $c(p,h) < 0$ for all $h \in \Omega$. So for particular p let us define:

$$c^*(p) := \max_{h \in \Omega} c(p,h) \tag{3.10}$$

then its clear that the closed-loop system is robustly stable if and only if $c^* < 0$, and the exponential decay rate will not be worse than $c^*(p)$ for all $h \in \Omega$. Similarly to [26], convergence to local but not global minimum of the spectral abscissa does not represent an issue since any negative value of $c^*(p)$ ensures a robust stability of the system.

3.4.1. Minimax game problem

Here, a non-cooperative dynamic game with two decision makers (players) is considered [33]. Each player follows a different strategy to guarantee himself the best result. The first player tries to minimize the spectral abscissa c as much as he can by changing the controller's free parameters p. The second player tries to

maximize the spectral abscissa as much as he can by changing the time delays h within its bounded interval. They play to find a single controller which stabilizes every plant in family $\mathbb{F}$ the following minimax problem should be solved:

$$c(p^*, h^*) = \min_{p} \max_{h} c(p, h) \tag{3.11}$$

where p^* is the controller's free parameters vector which ensure robust stability of system (3.1–3.2) under control (3.4), and h^* is the time delay which maximizes c^*. At the end of the game the system will be robustly stable if $c(p^*, h^*) < 0$.

3.4.2. Gradients calculation

Now, for a particular p and $h \in \Omega$, let s_1 be a simple closed-loop mode. Then, $\det(\Gamma(s_1, p, h)) = 0$ and, hence, there exists non-zero vectors $u, v \in \mathbb{C}^n$ (which depends on p and h) such that;

$$u^*\Gamma(s_1, p, h) = 0 \tag{3.12}$$

and

$$\Gamma(s_1, p, h)v = 0 \tag{3.13}$$

Now, let us fix h and, following [19], take the derivative of both sides of (3.13) with respect to the $(i)^{\text{th}}$ element, p_i, where $i = 1, \cdots, j,\ m(q+r)+(qr)$, by keeping in mind that s_1 also depends on p_i:

$$\left(\frac{\partial \Gamma}{\partial p_i} + \frac{\partial s}{\partial p_i}\frac{\partial \Gamma}{\partial s}\right) v + \Gamma\left(\frac{\partial v}{\partial p_i} + \frac{\partial s}{\partial p_i}\frac{\partial v}{\partial s}\right) = 0 \tag{3.14}$$

Now, multiply both sides of (3.14) by u^* from left, use (3.12), and rearrange terms to obtain:

$$\frac{\partial s}{\partial p_i} = -\frac{u^*\left(\frac{\partial \Gamma}{\partial p_i}\right)v}{u^*\left(\frac{\partial \Gamma}{\partial s}\right)v} \tag{3.15}$$

Thus, if s_1 is the right-most closed-loop mode,

$$\frac{\partial c}{\partial p_i} = -\mathrm{Re}\left(\frac{u^*\left(\frac{\partial \Gamma}{\partial p_i}\right)v}{u^*\left(\frac{\partial \Gamma}{\partial s}\right)v}\right) \tag{3.16}$$

where the partial derivatives on the right-hand side must be calculated at $s = s_1$ and at current p and h. In fact, we have:

$$\frac{\partial \Gamma}{\partial p_i} = -\frac{\partial \mathcal{A}_0(p)}{\partial p_i} \tag{3.17}$$

$$\frac{\partial \Gamma}{\partial s} = E_0 + \sum_{l=1}^{\nu} h_l \mathcal{A}_l e^{-s_1 h_l} \tag{3.18}$$

The gradient of $c(p,h)$ with respect to p is then given as

$$\frac{\partial c}{\partial p} = \begin{bmatrix} \frac{\partial c}{\partial p_1} & \frac{\partial c}{\partial p_i} & \cdots & \frac{\partial c}{\partial p_k} \end{bmatrix}' \tag{3.19}$$

where $k = m(q+r) + (qr)$. Next, to find the gradient of $c(p,h)$ with respect to h, take the derivative of both sides of (3.13) with respect to h_l ($l = 1, \ldots, \nu$), multiply both sides by u^* from left, use (3.12), and rearrange terms to obtain:

$$\frac{\partial s}{\partial h_l} = -\frac{u^* \left(\frac{\partial \Gamma}{\partial h_l}\right) v}{u^* \left(\frac{\partial \Gamma}{\partial s}\right) v} \tag{3.20}$$

Thus, if s_1 is the right-most closed-loop mode;

$$\frac{\partial c}{\partial h_l} = -\mathrm{Re}\left(\frac{u^* \left(\frac{\partial \Gamma}{\partial h_l}\right) v}{u^* \left(\frac{\partial \Gamma}{\partial s}\right) v} \right) \tag{3.21}$$

where, as before, the partial derivatives on the right-hand side must be calculated at $s = s_1$ and at current p and h. In fact,

$$\frac{\partial \Gamma}{\partial h_l} = \sum_{l=1}^{\nu} s_1 \mathcal{A}_l e^{-s_1 h_l}$$

and $\frac{\partial \Gamma}{\partial s}$ is given by (3.18). The gradient of $c(p,h)$ with respect to h is then given as

$$\frac{\partial c}{\partial h} = \begin{bmatrix} \frac{\partial c}{\partial h_1} & \cdots & \frac{\partial c}{\partial h_\nu} \end{bmatrix}' \tag{3.22}$$

3.4.3. Updating rule

To solve the game problem (3.11) the players using different strategies. First player p starts from a selected initial value, then this value updated using the rule

$$p^{\text{new}} = p - \beta \frac{\partial c}{\partial p} \tag{3.23}$$

A line search used to choose the value of β. Line search applied by initially selecting a value for β and choosing β_{min} , then p^{new} calculated if $c(p^{\text{new}}) < c(p)$ then the value of p updated $p = p^{\text{new}}$ otherwise if the value of $\beta > \beta_{min}$ it will updated

to be $\beta = \beta/2$. The second player strategy is to maximize $c(p, h)$ by choosing h, it is started by selecting initial value for h then this value updated using the rule

$$h^{\text{new}} = h + \alpha \frac{\partial c}{\partial h} \tag{3.24}$$

A line search used to choose the value of α. Line search applied by initially selecting a value for α and choosing α_{min} , then h^{new} calculated if $c(h^{\text{new}}) > c(h)$ then the value of h updated $h = h^{\text{new}}$ otherwise if the value of $\alpha > \alpha_{min}$ it will updated to be $\alpha = \alpha/2$.

4. CONTROLLER DESIGN

The controller design constitutes of several main steps, minimizing the spectral abscissa with respect to controller's free parameters, maximizing the spectral abscissa with respect to time delays, and correcting the path of the optimization using a grid search. Before presenting the controller design algorithm, those steps will be presented and then initialization procedure will be proposed.

4.1. Minimizing Spectral Abscissa with Respect to Controller's Free Parameters

Similarly to [34] an algorithm to minimize $c(p, h)$ with respect to p at a particular h can be presented as following:[1]

Algorithm 1 (to minimize $c(p, h)$ with respect to p, starting with a given p^{init})**:**

1. Initialize $p = p^{\text{init}}$.
2. Choose a step size $\beta > 0$ (e.g., $\beta = 1$).
3. Calculate $\frac{\partial c}{\partial p}$. If $\|\frac{\partial c}{\partial p}\| < \epsilon_0$, go to step 7. Otherwise, continue with step 4.
4. Let $p^{\text{new}} = p - \beta\frac{\partial c}{\partial p}$.
5. If $c(p^{\text{new}}, h) < c(p, h)$, let $p = p^{\text{new}}$ and go to step 2. Otherwise, continue with step 6.
6. If $\beta > \beta^{\min}$ let $\beta \leftarrow \beta/2$ and go to step 4. Otherwise, continue with step 7.
7. Let $p^{\dagger} = p$ and return to the main algorithm.

This algorithm can be used to design a suitable controller with a specific dimension in a particular h without uncertainty. In this work, it is used as a part of the main algorithm which is established to design a controller for a LTI time-delay system with multiple uncertain delays.

[1] ϵ, ϵ_0, ϵ_{fm}, ϵ_z, ϵ_1, ϵ_2, $\beta^{\min}$, and $\alpha^{\min}$ are prechosen positive small numbers and h_l^{new} is the l^{th} element of h^{new}.

4.2. Maximizing Spectral Abscissa with Respect to Time Delays

Similarly to [34] an algorithm to maximize $c(p, h)$ with respect to h at a particular p. can be presented as following:

Algorithm 2 (to maximize $c(p, h)$ with respect to h, starting with a given $h^{\text{init}} \in \Omega$)**:**

1. Initialize $h = h^{\text{init}}$.
2. Choose a step size $\alpha > 0$ (e.g., $\alpha = 1$).
3. Calculate $\frac{\partial c}{\partial h}$. If $\|\frac{\partial c}{\partial h}\| < \epsilon_1$, go to step 8. Otherwise, continue with step 4.
4. Let $h^{\text{new}} = h + \alpha \frac{\partial c}{\partial h}$.
5. For $l = 1, \ldots, \nu$,

   i) if $h_l^{\text{new}} > h_l^{\max}$, let $h_l^{\text{new}} = h_l^{\max}$;

   ii) if $h_l^{\text{new}} < h_l^{\min}$, let $h_l^{\text{new}} = h_l^{\min}$;

   and update h^{new} accordingly. If $\|h^{\text{new}} - h\| < \epsilon_2$, go to step 8. Otherwise, continue with step 6.
6. If $c(p, h^{\text{new}}) > c(p, h)$, let $h = h^{\text{new}}$ and go to step 2. Otherwise, continue with step 7.
7. If $\alpha > \alpha^{\min}$, let $\alpha \leftarrow \alpha/2$ and go to step 4. Otherwise, continue with step 8.
8. Let $h^{\dagger} = h$ and return to the main algorithm.

4.3. Grid Search to Find Actual Maximizing Time Delays

Since the spectral abscissa is a non-concave function and it may contain multiple minima or maxima, there is no guarantee that the result of Algorithm 2 will converge to a global maximum within the bounded interval. Because of that, a search grid applied to find the actual h^{max} which maximize the spectral abscissa to be compared with the result of Algorithm 2, and corrects the game path by restarting from the found h^*. The grid search is applied by selecting a number of

delays and find the value of real part of the spectral abscissa for each selected delay then compare the values to find the actual maximizing time delays. This can be represented in steps as in the following algorithm.

Grid search Algorithm to find h^{max} which maximize $c(p,h)$

1. Select a grid of M points,$h^1,\ldots,h^M \in \Omega$.
2. Let $i = M$ and $h^{max} = h^*$.
3. if $c(p,h_i) > c(p,h^{max})$ let $h^{max} = h_i$. Continue to next step.
4. if $i > 1$; $i = i - 1$ and return to step 3.

4.4. Initialization Procedure

It is well known that the spectral abscissa is typically not everywhere differentiable, even not everywhere Lipschitz continuous, although it is differentiable almost everywhere [23]. However, the spectral abscissa is nonconvex with respect to the feedback controller parameters. Therefore, complete randomly initialization for the feedback controller may not result in the success of stabilizing the system.

Before starting the controller design we initially should check two important points. First, check if there is any unstable fixed mode or approximately fixed mode for all $h \in \Omega$, if there is any unstable fixed mode then we cannot stabilize the system using any controller dimension. Second, the initial controller should satisfy the parity interlacing property (PIP) for all $h \in \Omega$.

4.4.1. Fixed modes

If there exists an unstable fixed mode, then this mode cannot be moved by any feedback controller (not just by static output feedback, but even with dynamic or time-delay controllers - see [4]) Because of that, a static feedback controller is applied and the unstable modes are checked whether they are fixed modes or not for all $h \in \Omega$. The open-loop mode, s_1 (whether simple or multiple), is a *fixed mode* [4] of the original system (3.1)–(3.2), for some $h \in \Omega$, if

$$\text{rank}\left[\begin{array}{cc} s_1 I - \sum_{l=0}^{\nu} A_l e^{-s_1 h_l} & \sum_{l=0}^{\nu} B_l e^{-s_1 h_l} \end{array}\right] < n \tag{4.1}$$

or

$$\text{rank} \begin{bmatrix} s_1 I - \sum_{l=0}^{\nu} A_l e^{-s_1 h_l} \\ \sum_{l=0}^{\nu} C_l e^{-s_1 h_l} \end{bmatrix} < n \tag{4.2}$$

Another way to check if the modes are fixed or even approximately fixed modes is to find the gradient of spectral abscissa with respect to controller's free parameters at s_1 such that if it is equal zero, then s_1 is a fixed mode and if it is approximately zero, then the mode s_1 is an approximate fixed mode.

4.4.2. Satisfy parity interlacing property

PIP states that a plant is strongly stabilizable if and only if the number of poles of the plant between every pair of real extended right half plane zeros of the plant is even [35]. Where, the system considered strongly stabilizable if it is stabilizable by a stable controller. A robust stabilizing controller can be found if and only if the initial controller satisfies the so called PIP for all $h \in \Omega$. This means that between any two system blocking zeros there should be an even number of real modes and there is an even number of system real modes to the right of the rightmost system blocking zero. Note that we can set the minimum dimension of the initial controller to be equal the added number of eigenvalues to matrix F. The steps (4 to 6) in Algorithm 3 show the procedure in details.

4.5. Controller Design Main Algorithm

Now, we can present our main algorithm to design a robust controller. In this algorithm, we aim to design a controller with the least possible dimension m, not to exceed some given upper limit m_{max}, which moves all the closed-loop modes as far left as possible into $\mathbb{C}_{\mu}^{-}$, for some given $\mu < 0$, for all $h \in \Omega$.

Algorithm 3 (Main Algorithm)**:**

1. Let $iter = 0$ and $s_{rmrm} = \mu$.
2. Choose a grid of M points, $h^1, \ldots, h^M$, in Ω.
3. For $l = 1, \ldots, M$, do the following:
   i) Let $h = h^l$.

ii) Calculate the open-loop μ-modes, $\Psi_\mu(\Sigma_{\rm OL}(h))$, of the system $\Sigma_{\rm OL}(h)$, where $\Sigma_{\rm OL}(h)$ denotes the open-loop system (3.1) with the time-delay vector h.

iii) For each $s_1 \in \Psi_\mu(\Sigma_{\rm OL}(h))$, with $\mathrm{Im}(s_1) \geq 0$,[2] do the following:

a) Calculate $\frac{\partial s}{\partial p}$ at $s = s_1$ and $p = p^{\rm OL}$, using (3.15), where $p^{\rm OL} = 0_{qr\times 1}$ is the free parameter vector corresponding to $m = 0$ and $K = 0$. If $\|\frac{\partial s}{\partial p}\| < \epsilon_{fm}$, then s_1 is either a fixed mode or an approximate fixed mode of $\Sigma_{\rm OL}(h)$, and hence there is either no solution or such a solution requires a controller with very high gains (such a controller can not be robust in general). Thus, declare **failure** and **stop**.

b) If $\mathrm{Im}(s_1) = 0$ and $s_1 > s_{rmrm}$, let $s_{rmrm} = s_1$.

4. If $s_{rmrm} = \mu$, let $m = m_{min} = 0$ and go to step 7.

5. For $l = 1, \ldots, M$, do the following:

i) Let $h = h^l$.

ii) Find the real blocking zeros, $z_1^h, \ldots, z_{n_z}^h$ (in ascending order), of $\Sigma_{\rm OL}(h)$ between μ and s_{rmrm}. If $n_z = 0$ (i.e., if there are no such zeros), let $n_{rgn} = 0$ and go to step iv.

iii) Let $n_{rgn} = 0$, $z_{n_z+1} := s_{rmrm} + \epsilon_z$ and, for $k = 1, \ldots, n_z$, do the following:

If there are an odd number (counting multiplicities) of real modes of $\Sigma_{\rm OL}(h)$ between z_k and z_{k+1}, let $n_{rgn} = n_{rgn} + 1$, $\zeta^-_{n_{rgn}} = z_k$, $\zeta^+_{n_{rgn}} = z_{k+1}$.

iv) If $l = 1$, let $m_{min} = n_{rgn}$ and, if $n_{rgn} \neq 0$, for $k = 1, \ldots, n_{rgn}$, let $Z_k^- = \zeta_k^-$ and $Z_k^+ = \zeta_k^+$, and go to step viii.

v) If $m_{min} \neq n_{rgn}$, go to step vii. Otherwise, if $n_{rgn} \neq 0$, for $k = 1, \ldots, n_{rgn}$, do the following:

a) Let $Z_k^- = \max(Z_k^-, \zeta_k^-)$ and $Z_k^+ = \min(Z_k^+, \zeta_k^+)$.

[2]Since the modes of a real system are either real or occur in complex-conjugate pairs and a non-real mode is fixed if and only if its complex-conjugate is also fixed, we do not need to check for the modes with negative imaginary part.

b) If $Z_k^- \geq Z_k^+$, go to step vii.

vi) Go to step viii.

vii) PIP can not be satisfied for all $h \in \Omega$ by the same controller; i.e., there is no solution to the robust controller design problem. Thus, declare **failure** and **stop**.

viii) Continue.

6. Let $m = m_{min}$. If $m > 0$, for $k = 1, \ldots, m$, choose λ_k in the interval (Z_k^-, Z_k^+).

7. If $m > m_{min}$, choose $\lambda_{m_{min}+1}, \ldots, \lambda_m$ (each of which is either real or occur in complex-conjugate pairs) randomly in $\mathbb{C}_\mu^-$.

8. Initialize the free parameters of F so that F has eigenvalues $\lambda_1, \ldots, \lambda_m$. Initialize the free parameters of G, H, and K randomly (if $m = 0$, only the entries of K are initialized randomly). Then choose the initial free parameters vector p^0 accordingly. Also choose the initial time-delay vector $h^0 \in \Omega$ randomly and let $n = 0$ and $count = 0$.

9. Run Algorithm 1 with $h = h^n$ and $p^{\text{init}} = p^n$ to minimize $c(p, h^n)$. Let $p^{n+1} = p^\dagger$, the minimizing p.

10. Run Algorithm 2 with $p = p^{n+1}$ and $h^{\text{init}} = h^n$ to maximize $c(p^{n+1}, h)$. Let $h^{n+1} = h^\dagger$, the maximizing h.

11. If $\|h^{n+1} - h^n\| < \epsilon$, let $p^* = p^{n+1}$, $h^* = h^{n+1}$, and continue with the next step. Otherwise, let $n = n + 1$ and go to step 9.

12. Choose a grid of M points, $h^1, \ldots, h^M$, in Ω. Let $h^{max} = h^*$, $c^{max} = c(p^*, h^*)$, and, for $l = 1, \ldots, M$, do the following:

    If $c(p^*, h^l) > c^{max}$, let $h^{max} = h^l$ and $c^{max} = c(p^*, h^l)$.

13. If $h^{max} = h^*$ go to step 15. Otherwise, continue with the next step.

14. If $count < count_{max}$, let $count = count + 1$, $n = 0$, $h^0 = h^{max}$, $p^0 = p^*$ and go to step 9. Otherwise, let $h^* = h^{max}$ and continue with the next step.

15. If $iter = 0$, let $c^* = c(p^*, h^*)$, $p^{**} = p^*$, $iter = iter + 1$ and go to step 7. Otherwise, continue with the next step.

16. If $c(p^*, h^*) < c^*$, let $c^* = c(p^*, h^*)$ and $p^{**} = p^*$.

17. If $iter < iter_{max}$, let $iter = iter + 1$ and go to step 7. Otherwise, continue with the next step.

18. If $c^* < \mu$, form the **optimal robust controller** with the free parameters p^{**} and **stop**. Otherwise, continue with the next step.

19. If $m < m_{max}$, let $m = m + 1$, $iter = 0$, and go to step 7. Otherwise, declare **failure** and **stop**.

4.5.1. Algorithm 3 explanation

Algorithm 3 can be divided into two main parts, first part contains steps 1 to 6. This part implements the initialization procedure such that in step 1 the rightmost real mode 's_{rmrm}', and iteration number *iter* initialized. In steps 3 and 4 for every $h \in \Omega$ the open loop system modes to the right of $\mu < 0$ are checked whether they are approximate μ fixed modes or not, in the same time s_{rmrm} checked and updated. Then in step 5 the real blocking zeros in the region ($\mu < R < s_{rmrm}$) are found, then the real modes between every two blocking zeros are counted, also the real modes between the rightmost zero and 's_{rmrm}' are counted. If any region contains an odd number of real modes then the counter 'n_{rgn}' increased by one for each region. This step repeated for each $h \in \Omega$ and for each h the resulted regions compared by the regions resulted by the previous h, such that the region for h updated by finding the intersection between the regions and the previous found ones. If there is any conflict in regions such that any resulted region has a lower bound higher than its higher bound or if the number of regions is different, then the procedure declares failure. The minimum controller dimension 'm_{min}' determined in step 4 to be zero if the system doesn't have any real mode to the right of μ and steps 5 and 6 skipped, or 'm_{min}' determined in step 5, such that after success to find suitable regions, then '$m_{min} = n_{rgn}$' since the number of regions equals the

number of needed eigenvalues to be added to controller matrix F to satisfy PIP.

The second part contains steps 7 to 17, in those steps a controller is designed for specific controller dimension using multiple initial points. Such that, steps 9, 10, and 11 solves the minimax problem by running Algorithms 1 and 2. Step 12 applies a grid search to find the actual maximizing h^{max} to be compared in step 13 with h^*. In step 14, if the counter of iterations does not exceed its higher limit, then the minimax problem solving with actual h^{max} and p^*, The counter is added such that the steps 9 to 14 restricted by a max number of iterations to avoid oscillations. Step 15, 16, and 17 give the opportunity to test $iter_{max}$ number of initial points results to be compared and the best to be chosen.

In steps 18, and 19 the algorithm stopped either by success if $c^* < \mu$ or by failure if m reach its predetermined maximum. Algorithm 3 implemented in Matlab codes package which contains a main function and sub functions to design a robust output feedback controller for retarded LTI time-delay systems. Codes and their explanations are given in the Appendix.

4.6. Limitations of the Procedure

The controller design using Algorithm 3 may fail to design the desired controller because of the following reasons:

1. Existence of an unstable fixed mode or an approximately fixed mode for any $h \in \Omega$.

2. If it is not possible to initialize a controller such that it satisfies the so called PIP for all $h \in \Omega$.

3. Occurrence of oscillation such that the Algorithm keeps running between steps 9 to 11, the minimization and maximization keep oscillating between some values.

If the mentioned reasons 1 or 2 occurs then the system cannot be robustly stabilized for all $h \in \Omega$ by a single controller. But if the third problem occurs, then an

alternative method explained in the next section can be used.

4.6.1. Second option to solve the oscillation problem

To solve the oscillation problem between step 9 to 11 in the main algorithm, an altarnative algorithm given as following:

Algorithm 4 (Alternative Algorithm)**:**

1-7. Same as steps 1-7 of Algorithm 3.

8. Initialize the free parameters of F so that F has eigenvalues $\lambda_1, \ldots, \lambda_m$. Initialize the free parameters of G, H, and K randomly (if $m = 0$, only the entries of K are initialized randomly). Then choose the initial free parameters vector p^0 accordingly. Also choose the initial time-delay vector $h^0 \in \Omega$ randomly.

9. Let $p = p^0$ and $h = h^0$

10. Choose a grid of M points, $h^1, \ldots, h^M$, in Ω.

11. For $l = 1, \ldots, M$, do the following:

    If $c(p, h^l) > c(p, h)$, let $h = h^l$.

12. Choose a step size $\beta > 0$ (e.g., $\beta = 1$).

13. Calculate $\frac{\partial c}{\partial p}$ at present p and h. If $\|\frac{\partial c}{\partial p}\| < \epsilon_0$, go to step 18. Otherwise, continue with the next step.

14. Let $p^{\text{new}} = p - \beta \frac{\partial c}{\partial p}$.

15. Let $h^{max} = h$, and for $l = 1, \ldots, M$, do the following:

    If $c(p^{\text{new}}, h^l) > c(p^{\text{new}}, h^{max})$, let $h^{max} = h^l$.

16. If $c(p^{\text{new}}, h^{max}) < c(p, h)$, let $p = p^{\text{new}}$, $h = h^{max}$ and go to step 12. Otherwise continue with the next step.

17. If $\beta > \beta^{\min}$ let $\beta \leftarrow \beta/2$ and go to step 14. Otherwise, continue with the next step.

18. If $iter = 0$, let $p^* = p$, $h^* = h$, $iter = iter + 1$ and go to step 7. Otherwise, continue with the next step.

19. If $c(p, h) < c(p^*, h^*)$, let $p^* = p$, $h^* = h$.

20. If $iter < iter_{max}$, let $iter = iter + 1$ and go to step 7. Otherwise, continue with the next step.

21. If $c(p^*, h^*) < \mu$, form the **optimal robust controller** with the free parameters p^* and **stop**. Otherwise, continue with the next step.

22. If $m < m_{max}$, let $m = m + 1$, $iter = 0$, and go to step 7. Otherwise, declare **failure** and **stop**.

This alternative is valid for all cases but it consumes longer time compared to Algorithm 3. The results of both alternatives were approximately equal for the tested examples.

5. NUMERICAL EXAMPLES

5.1. Example 1

A single input single output system with single delay in its state is considered:

$$\dot{x}(t) = A_0 x(t) + A_1 x(t - h_1) + B_0 u(t) \tag{5.1}$$

$$y(t) = C_0 x(t) \tag{5.2}$$

where

$$A_0 = \begin{bmatrix} -1.5 & 0 \\ 0 & -0.9 \end{bmatrix}, \quad A_1 = \begin{bmatrix} -1 & 0 \\ -1 & -1 \end{bmatrix}$$

$$B_0 = \begin{bmatrix} -0.5 \\ 1 \end{bmatrix}, \quad C_0 = \begin{bmatrix} 1 & 0.1 \end{bmatrix}$$

The nominal value of the uncertain time-delay h_1 is assumed to be ${h_1}^0 = 8$ and its minimum and maximum values are assumed to be 6 and 9, respectively. The open-loop system has an unstable pair of modes at 0.004 ± 0.3469i. Algorithm 3 used to design a robustly stable controller, since the system for all $h \in \Omega$ has no unstable real modes, steps 5 and 6 in algorithm 3 are skipped and set $m_{min} = 0$. Robust stabilizing the system by static controller failed and the best exponential decay rate can be achieved for all $h \in \Omega$ is 0.0030 at $K = -0.7919$.

The controller dimension m is increased by 1, then a robust controller described as:

$$\dot{z}(t) = 0.0145 z(t) + y(t)$$

$$u(t) = 0.1131 z(t) - 0.8338 y(t)$$

is designed, it has an exponential decay rate -0.0028 for the worst delay case $h^* = 9$. The open-loop μ-modes and closed-loop μ-modes are shown in Figure 5.1.

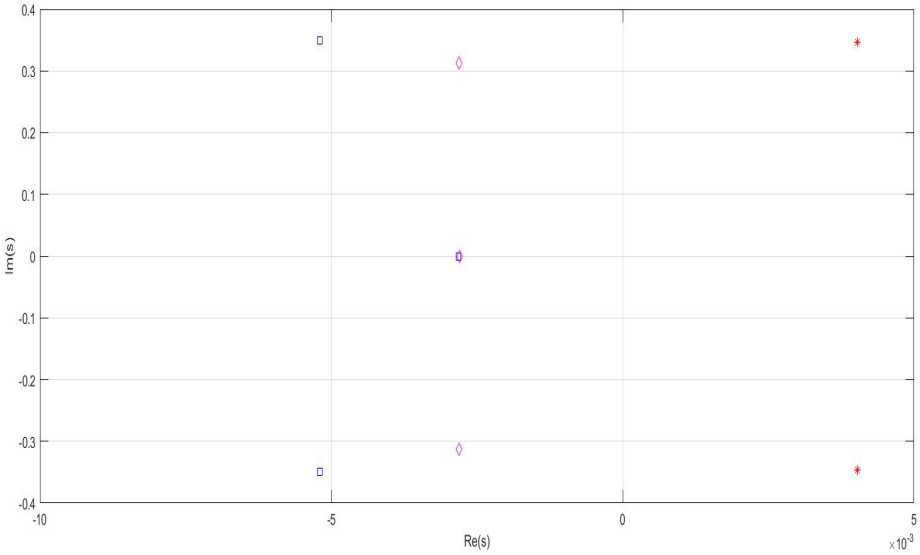

Figure 5.1 *(i) Open-loop μ-modes for $h = h^0$ (red stars); (ii) Closed-loop μ-modes at $h = h^0$(blue squares), (iii) Closed-loop μ-modes at $h = h^*$ (purple diamonds). For Example 1, $\mu = -0.01$.*

5.2. Example 2

A single input single output system with single delay in its input is considered.

$$\dot{x}(t) = A_0 x(t) + B_1 u(t - h_1) \tag{5.3}$$

$$y(t) = C_0 x(t) \tag{5.4}$$

where

$$A_0 = \begin{bmatrix} -4.8 & 4.7 & 3 \\ -0.3 & 1.4 & -0.55 \\ -0.7 & 3.1 & -1.5 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0.3 \\ 0.7 \\ 0.1 \end{bmatrix}, \quad C_0 = \begin{bmatrix} -2 & 3 & 0.20 \end{bmatrix}$$

The nominal value of the uncertain time-delay h_1 is assumed to be ${h_1}^0 = 0.5$ and its minimum and maximum values are assumed to be 0.35 and 0.65, respectively.

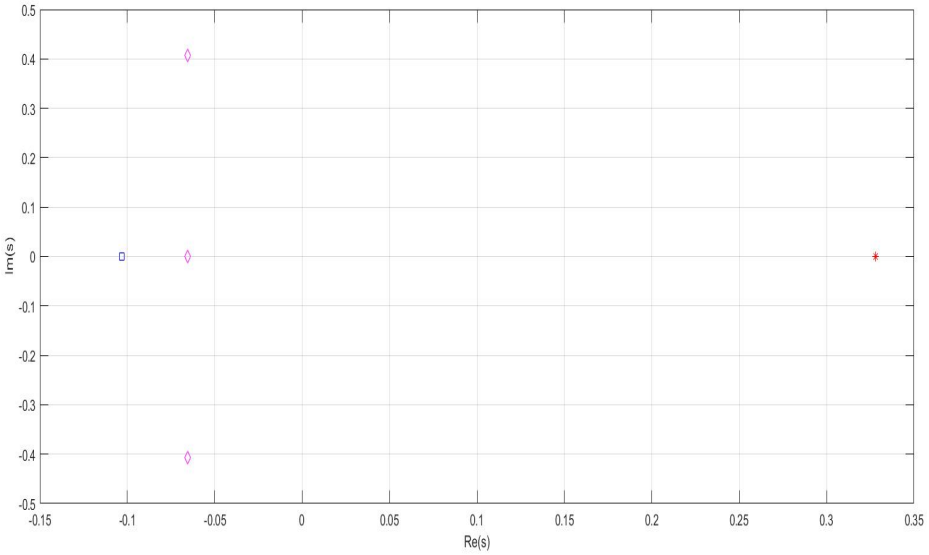

Figure 5.2 *(i) Open-loop μ-modes for $h = h^0$ (red stars); (ii) Closed-loop μ-modes controller with dimension 1 and $h = h^0$ (blue squares), (iii) Closed-loop μ-modes for controller with dimension 1 and $h = h^*$ (purple diamonds). For Example 2, $\mu = -0.15$.*

The system has unstable open-loop mode at 0.3278 and one blocking zero at 0.2155. The minimum controller dimension set to $m_{min} = 1$, since the blocking zero and the unstable mode are delay independent, the initial controller eigenvalue selected at the middle of the region $[0.2115, 0.3278]$ to be 0.2717. Then a robust controller, described as:

$$\dot{z}(t) = 0.5758z(t) + y(t)$$

$$u(t) = -0.8719z(t) - 0.2916y(t)$$

is designed to finally has an exponential decay rate -0.0653 at the worst time-delay case $h^* = 0.65$. The open-loop μ-modes and closed-loop μ-modes are shown in Figure 5.2.

5.3. Example 3

A single input single output system with one delay in state and one delay in input system described as:

$$\dot{x}(t) = A_0x(t) + A_2x(t - h_2) + B_1u(t - h_1) \tag{5.5}$$

$$y(t) = C_0x(t) \tag{5.6}$$

where

$$A_0 = \begin{bmatrix} -2.5 & 0.5 \\ 0 & -3.5 \end{bmatrix}, \quad A_2 = \begin{bmatrix} 3 & -1 \\ 0 & 5 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} -3.5 \\ 3 \end{bmatrix}, \quad C_0 = \begin{bmatrix} 0.5 & 2.25 \end{bmatrix}$$

The nominal value of the uncertain time-delay h_1 is assumed to be ${h_1}^0 = 0.7$ and its minimum and maximum values are assumed to be 0.65 and 0.75, respectively. The nominal value of the uncertain time-delay h_2 is assumed to be ${h_2}^0 = 0.8$ and its minimum and maximum values are assumed to be 0.7 and 0.85, respectively. This example is taken from [36] where a finite controller dimension and an infinite controller dimension were designed, here an uncertainty added to time delays and a robust finite controllers of dimension 0 and 1 are designed.

The system has no fixed modes and has no blocking zeros in the right half plane so the minimum controller dimension set to $m_{min} = 0$. The real rightmost open-loop mode of the system is changed between 0.3175 and 0.3489 by changing h_1 so $s_{rmrm} = 0.3489$. A static feedback controller is first designed $K = -0.4957$ at nominal time delays and resulted in $c = -0.0233$, then the design steps continued to finally have $c = -0.0085$ using the robust static output feedback controller $K = -0.4969$. To improve the result a robust controller of dimension 1, described as:

$$\dot{z}(t) = -0.9737z(t) + y(t)$$

$$u(t) = -0.2069z(t) - 0.2782y(t)$$

is designed with guaranteed exponential decay rate $c = -0.0185$. For both controllers h^* found to be $\begin{bmatrix} 0.75 & 0.75 \end{bmatrix}'$. The open-loop μ-modes and closed-loop μ-modes are shown in Figure 5.3.

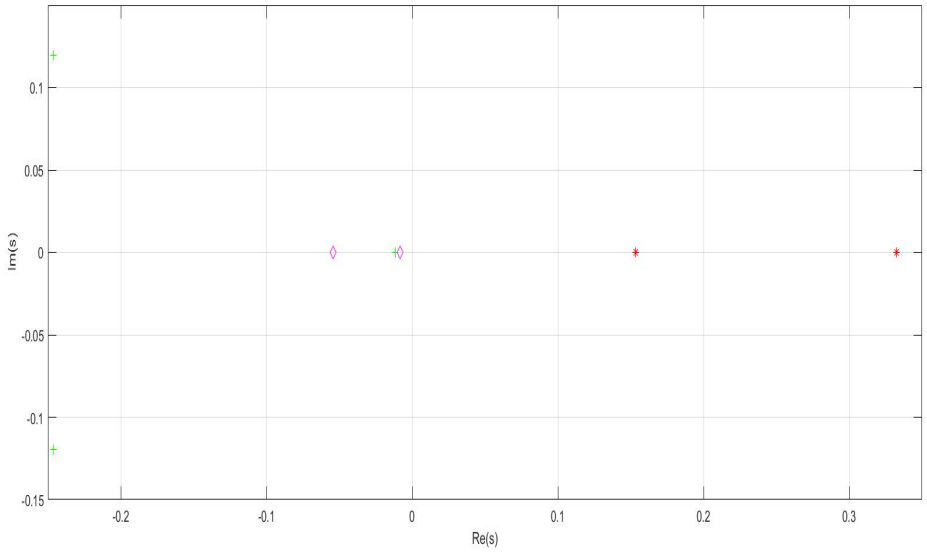

Figure 5.3 *(i) Open-loop μ-modes for $h = h^0$ (red stars); (ii) Closed-loop μ-modes for $K = -0.4969$ and $h = h^*$ (purple diamonds), (iii) Closed-loop μ-modes for controller with dimension 1 and $h = h^*$ (green pluses). For Example 3, $\mu = -0.15$.*

5.4. Example 4

A system with two input and two outputs, with two delays is considered.

$$\dot{x}(t) = A_0 x(t) + A_1 x(t - h_1) + B_1 u(t - h_1) + B_2 u(t - h_2) \tag{5.7}$$

$$y(t) = C_1 x(t - h_1) \tag{5.8}$$

where

$$A_0 = \begin{bmatrix} 0.1 & -0.5 & -0.3 & 0.3 \\ 0 & 0.1 & 0.2 & -0.2 \\ 0 & 0 & 0.1 & 0 \\ 0 & 0.4 & 0.3 & -0.4 \end{bmatrix}$$

$$A_1 = \begin{bmatrix} 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \\ -0.0360 & -0.426 & -2.66 & -3.84 \end{bmatrix}$$

$$B_1 = \begin{bmatrix} 0 & 0 \\ 0 & 0 \\ 0 & 0 \\ 0 & 1 \end{bmatrix}, \quad B_2 = \begin{bmatrix} 0.3 & 0 \\ 0.1 & 0 \\ 0.3 & 0 \\ 0.1 & 0 \end{bmatrix}$$

$$C_1 = \begin{bmatrix} 0.01 & -0.13 & 0.2 & 1 \\ 0.04 & -0.58 & 1.7 & 1 \end{bmatrix}$$

The nominal value of the uncertain time-delay h_1 is assumed to be ${h_1}^0 = 0.25$ and its minimum and maximum values are assumed to be 0.20 and 0.30, respectively. And the nominal value of the uncertain time-delay h_2 is assumed to be ${h_2}^0 = 1.8$ and its minimum and maximum values are assumed to be 1.75 and 1.85, respectively. The system has a pair of unstable open loop modes at 0.116 ± 0.0926i, and it has no unstable real modes so the minimum controller dimension set to $m_{min} = 0$ and a static feedback controller, described as:

$$K = \begin{bmatrix} 1.2380 & 0.1661 \\ -0.9326 & -0.0368 \end{bmatrix}$$

is designed to robustly stabilize the system even in the worst time-delay case $h^* = \begin{bmatrix} 0.3 & 1.85 \end{bmatrix}'$ with an exponential decay rate -0.0289. The open-loop μ-modes and closed-loop μ-modes are shown in Figure 5.4.

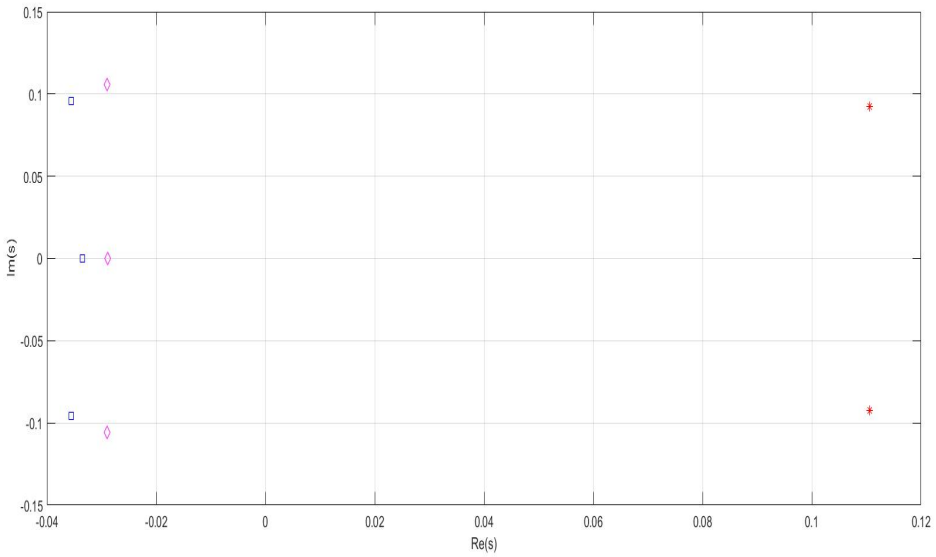

Figure 5.4 *(i) Open-loop μ-modes for $h = h^0$ (red stars); (ii) Closed-loop μ-modes at $h = h^0$(blue squares), (iii) Closed-loop μ-modes at $h = h^*$ (purple diamonds). For Example 4, $\mu = -0.04$.*

6. CONCLUSION

An approach to robustly stabilize retarded LTI time-delay systems with multiple uncertain time delays has been presented. Both static and dynamic output feedback controllers have been considered. The approach has been implemented by algorithms, which includes initialization procedure, and solves the minimax problem which has been formulated to obtain the best guaranteed exponential decay rate, even in the worst time delay case. The initialization procedure first checks the possibility of robustly stabilizing the plant by a single controller to avoid waste of time for systems which cannot be robustly stabilized by a single controller. Then it determines the minimum possible controller dimension. Numerical examples have been solved to show the ability to robustly stabilize retarded LTI time-delay systems using the proposed approach.

REFERENCES

[1] J.-H. S. Min Wu, Yong He, *Stability Analysis and Robust Control of Time-Delay Systems*. Berlin: Scince Press Beijing and Springer-Verlag, 2010.

[2] H. Özbay, *Introduction to Feedback Control Theory*. Boca Raton: CRC Press, 1999.

[3] J. K. Hale and S. M. Verduyn-Lunel, *Introduction to Functional Differential Equations*. New York: Springer-Verlag, 1993.

[4] H. E. Erol and A. İftar, "Stabilization of decentralized descriptor-type neutral time-delay systems by time-delay controllers," *Automatica*, vol. 64, pp. 262 – 269, 2016.

[5] W. Michiels and S.-I. Niculescu, *Stability and Stabilization of Time-Delay Systems*. Philadelphia: SIAM, 2007.

[6] S. Yi, A. G. Ulsoy, and P. W. Nelson, "Design of observer-based feedback control for time-delay systems with application to automotive powertrain control," *Journal of the Franklin Institute*, vol. 347, no. 1, pp. 358 – 376, 2010.

[7] O. J. M. Smith, "A controller to overcome dead time," *Instrument Society of America Journal*, vol. 6, pp. 28–33, 1959.

[8] O. Toker and H. Özbay, "$\mathcal{H}^\infty$ optimal and suboptimal controllers for infinite dimensional SISO plants," *IEEE Transactions on Automatic Control*, vol. 40, pp. 751–755, 1995.

[9] G. Tadmor, "The standart $\mathcal{H}_\infty$ problem in systems with a single input delay," *IEEE Transactions on Automatic Control*, vol. 45, pp. 382–397, 2000.

[10] G. Meinsma and H. Zwart, "On $\mathcal{H}_\infty$ control for dead-time systems," *IEEE Transactions on Automatic Control*, vol. 45, pp. 272–285, 2000.

[11] P.-F. Quet, B. Ataşlar, A. İftar, H. Özbay, S. Kalyanaraman, and T. Kang, "Rate-based flow controllers for communication networks in the presence of uncertain time-varying multiple time-delays," *Automatica*, vol. 38, pp. 917–928, 2002.

[12] G. Meinsma and L. Mirkin, "$\mathcal{H}^\infty$ control of systems with multiple I/O delays via decomposition to adobe problems," *IEEE Transactions on Automatic Control*, vol. 50, pp. 199–211, 2005.

[13] H. U. Ünal, B. Ataşlar-Ayyıldız, A. İftar, and H. Özbay, "Robust flow control in data-communication networks with multiple time-telays," *International Journal of Robust and Nonlinear Control*, vol. 20, pp. 1529–1548, 2010.

[14] H. U. Ünal and A. İftar, "Stable $\mathcal{H}^\infty$ controller design for systems with multiple input/output time-delays," *Automatica*, vol. 48, pp. 563–568, 2012.

[15] V. Kolmanovskii, S.-I. Niculescu, and J. P. Richard, "On the Liapunov-Krasovskii functionals for stability analysis of linear delay systems," *International Journal of Control*, vol. 72, pp. 374–384, 1999.

[16] E. Fridman and U. Shaked, "An improved stabilization method for linear time-delay systems," *IEEE Transactions on Automatic Control*, vol. 47, pp. 1931–1937, 2002.

[17] Y. He, M. Wu, J.-H. She, and G.-P. Liu, "Parameter dependent Lyapunov functional for stability of time-delay systems with polytopic-type uncertainties," *IEEE Transactions on Automatic Control*, vol. 49, pp. 828–832, 2004.

[18] P. Park and J. W. Ko, "Stability and robust stability for systems with a time-varying delay," *Automatica*, vol. 43, pp. 1855–1858, 2007.

[19] W. Michiels, K. Engelborghs, P. Vansevenant, and D. Roose, "Continuous pole placement for delay equations," *Automatica*, vol. 38, pp. 747–761, 2002.

[20] H. E. Erol and A. İftar, "Controller design by continuous pole placement for neutral time-delay systems," in *Proceedings of the TOK Automatic Control National Meeting*, Denizli, Turkey, Sept. 2015, pp. 304–309, (In Turkish).

[21] ——, "Decentralized controller design by continuous pole placement for neutral time-delay systems," *Transactions of the Institute of Measurement and Control*, vol. 39, pp. 297–311, 2017.

[22] ——, "Decentralized controller design by continuous pole placement for commensurate-time-delay systems," in *Proceedings of the 19th IFAC World Congress*, Cape Town, South Africa, Aug. 2014, pp. 9419–9424.

[23] J. Vanbiervliet, K. Verheyden, W. Michiels, and S. Vandewalle, "A nonsmooth optimisation approach for the stabilisation of time-delay systems," *ESAIM: Control, Optimisation and Calculus of Variations*, vol. 14, no. 3, pp. 478–493, 2008.

[24] S. M. Özer and A. İftar, "Decentralized controller design for time-delay sys-

tems by optimization," in *Preprints of the 12th IFAC Workshop on Time-delay Systems*, Ann Arbor, MI, USA, June 2015, pp. 462–467.

[25] ——, "Controller design for neutral time-delay systems by nonsmooth optimization," in *Preprints of the 13th IFAC Workshop on Time-delay Systems*, Istanbul, Turkey, June 2016, pp. 212–217.

[26] F.Borgioli and W.Michiels, "Robust stabilisation of linear time-delay systems with uncertainities in the system matricies and in the delay terms." in *Proceedings of the 14th IFAC Workshop on Time Delay Systems*, Budapest, Hungary, June 2018.

[27] I. P. Sigurd Skogestad, *Multivariable Feedback Control Analysis and Design.* John Wiley & Sons, 2001.

[28] Z. Wu and W. Michiels, "Reliably computing all characteristic roots of delay differential equations in a given right half plane using a spectral method," *Journal of Computational and Applied Mathematics*, vol. 236, pp. 2499–2514, 2012.

[29] T. Vyhlidal and P. Zitek, "QPmR - Quasi-polynomial root-finder: Algorithm update and examples," in *Delay Systems: From Theory to Numerics and Applications,* T. Vyhlidal, J.-F. Lafay, and R. Sipahi (Eds.), Springer, New York, 2014, pp. 299–312.

[30] N. Raju, *Optimization Methods for Engineers.* Delhi: PHI Learning Private Limited, 2014.

[31] D. G. Luenberger, "Canonical forms for linear multivariable systems," *IEEE Transactions on Automatic Control*, vol. AC–12, pp. 290–293, 1967.

[32] T. L. Ting, "Robustness in feedback systems," Ph.D. dissertation, University of Illinois at Urbana-Champaign, 1987.

[33] T. Başar and P. Bernhard, *H^∞-Optimal Control and Related Minimax Design Problems: A Dynamic Game Approach.* Boston: Birkhäuser, 1991.

[34] S. Elmuhandes and A. İftar, "Robust static output feedback controller design for systems with uncertain time-delays," in *Proceedings of the 14th IEEE International Conference on Control and Automation*, Anchorage, Alaska, 2018.

[35] M. Vidyasagar, *Control System Synthesis: A Factorization Approach.* Cambridge, MA: M.I.T. Press, 1985.

[36] H. E. Erol and A. Iftar, "Time-delay compensator design," in *Proceedings of the 11th Asian Control Conference*, Gold Coast, Australia, 2017, pp. 2441–2446.

APPENDIX

Algorithms 3 and 4 implemented in a Matlab codes package, this appendix contains the Matlab codes and functions and explanation for some parts. [1]

1. Basic Functions

Basic functions are functions called by other functions, those functions written as separate functions to make coding easier.

1.1. Finding Characteristic Matrix Function

This function finds the system characteristic matrix X. It returns a vector t which is a time delays symbols to be used with other functions.

```
function [t,X] = char_matrix_d(E0,A,t0)
%E0,A ar ethe cloosed loop system matricies where A={A0,A1
    ,...,Ai}
%   t0 is time delays vector where t0={0;h1;...;hi}
syms s;
t=sym('t',size(t0));
SizeA=size(A);
X=s*E0;
for i=1:SizeA(2)
    X=X-A{i}*exp(-s*t(i));
end
end
```

1.2. Finding the μ Modes and Spectral Abscissa in a Bounded Region

This function use QPmR function to find the μ modes of the system R and the spectral abscissa c by finding the roots of the characteristic matrix in a given bounded region.

```
function [R, c] = computec( X,K,Ki,t,t0,region)
X=subs(X,K,Ki);
X=subs(X,t,t0);
```

[1] The symbols differ between the algorithms and codes

```
Xd=det(subs(X));
F=matlabFunction(Xd);
R=[];
ds=0.01;
while (isempty(R))
    R=QPmR(region,F,1e-6,ds);
    if isnan(R)
        R=QPmR(region,F,1e-6,ds*10);
    end
    [rs I]=max(real(R));
    c=R(I);
    if isnan(R)
        R=[];
        a=(region(1,2)-region(1,1))/7
        region1=region
        for reg=1:1:7
            region1=[region1(1,1), (region1(1,1)+a),region1
                (1,3),region1(1,4)];
            Rn=QPmR(region1,F,1e-6,ds);
            if isnan(Rn)
                Rn=[];
                a_1=(region1(1,4)-region1(1,3))/7
        region21=region1
        for reg2=1:1:7
            region1=[region21(1,1), (region21(1,2)),region21
                (1,3),region21(1,3)+a_1];
            Rn2=QPmR(region21,F,1e-6,ds);
             if isnan(Rn2)
                Rn2=[];
            end
            if ~isempty(Rn2)
                siz_R2=size(Rn)
                siz_Rn2=size(Rn2)
```

```
                Rn(siz_R2(1)+1:siz_R2(1)+siz_Rn2(1))=Rn2
            end

        end
            if ~isempty(Rn)
                siz_R=size(R)
                siz_Rn=size(Rn)
                R(siz_R(1)+1:siz_R(1)+siz_Rn(1))=Rn
            end
            region1(1,1)=region1(1,2);
            end
        end
        [rs I]=max(real(R));
        c=R(I);
    end
    if isempty(R)
        region(1,1)= region(1,1)-1;
        region(1,2)=  region(1,2)+1;
        region(1,3)= region(1,3)-1;
        region(1,4)= region(1,4)+1;

    end
end

end
```

1.3. Finding the System Eigenvalues to the Right of μ and Define a Region

This function use discretisation method function to find the system eigenvalues to the right of μ, then it defines a rectangle region such that it contains all the found eigenvalues and adds a margin.

```
function [region c eigs] = computeregion(E,A,t0,mu)
%   This function finds the system eigen values and then
```

```
    select the region
%    depending on the max and min values founded." extra margins
     added to
%    the selected region"
%
size_A=size(A);
j=1;
for i=1: size_A(2)
     if ~isempty(A{1,i})
      A1{1,j}=A{1,i};
      hA(j,1)=t0(i,1);
      j=j+1;
     end
end
options.minimal_real_part=(mu-0.01);
sys.E=E;
sys.hE=0;
sys.A=A1;
sys.hA=(hA);
[eigenvalues]=DDAE_characteristic_roots(sys,options);
if ((max(real(eigenvalues.l1)))+0.1)<options.minimal_real_part
     region=[round(max(real(eigenvalues.l1))-1.6),round(abs(max
         (real(eigenvalues.l1)))+9.6), -round(abs(max(imag(
         eigenvalues.l1)))+5.6) ,round(abs(max(imag(eigenvalues.
         l1))+5.6)) ];


else
region=[options.minimal_real_part ,round(abs(max(real(
    eigenvalues.l1)))+9.6), -round(abs(max(imag(eigenvalues.l1)
    )+5.6)) ,round(abs(max(imag(eigenvalues.l1))+5.6)) ];
end
c= max(real(eigenvalues.l1));
if isempty(c)
```

```
    region=[-2,3,-2,+2];
end
eigs=eigenvalues.l1;
end
```

1.4. Compute Gradients

The first function finds the gradient of the spectral abscissa with respect to controller parameters.

```
function [GradK] = gradMiMoK( X,Ki,ti,s1 )
%  finds the gradient of c with respect to Ki
syms s;
[m,n] = size(Ki);
ts=size(ti);
t=sym('t',[ts]);
K= sym('K', [m,n]);
GradK=zeros(m,n);
Kn=reshape(K,[],1);
Gs=diff(X,s);
Gs=subs(Gs,K,Ki);
Gs=subs(Gs,t,ti);
Gs=subs(Gs,s,s1);
X1=subs(X,K,Ki);
X1=subs(X1,t,ti);
X1=(subs(X1,s,s1));
[u S v]=svd(X1);
V=(v(:,end));
[u S v]=svd(transpose(X1));
U= transpose((v(:,end)));
% V=null(double(X1));
% U=transpose(null(transpose(double(X1))));
[i j]=size(Kn);
for j= 1:i
GK=diff(X,Kn(j));
```

```
GK=subs(GK,K,Ki);
GK=subs(GK,t,ti);
GK=subs(GK,s,s1);
GradK(j)=-real(double(((U)*(GK)*(V))/(((U)*((Gs))*(V)))));
end
GradK=reshape(GradK,[m,n]);
end
```

The second function finds the gradient of the spectral abscissa with respect to time delays.

```
function [ Gradt] = gradMiMot( X,Ki,ti,s1,tmin,tmax )
%  finds the gradient of c with respect to t at ti and Ki
%s1 is the most right eigenvalue.
syms s;
[m,n] = size(Ki);
ts=size(ti);
t=sym('t',[ts]);
K= sym('K', [m,n]);
Gradt=zeros(ts);
Gs=diff(X,s);
Gs=subs(Gs,K,Ki);
Gs=subs(Gs,t,ti);
Gs=subs(Gs,s,s1);
X1=subs(X,K,Ki);
X1=subs(X1,t,ti);
X1=(subs(X1,s,s1));
[u S v]=svd(X1);
V=(v(:,end));
[u S v]=svd(transpose(X1));
U= transpose((v(:,end)));
% V=null(double(X1));
% U=transpose(null(transpose(double(X1))));
[i j]=size(t);
for j= 1:i
```

```
    Gt=diff(X,t(j));
    Gt=subs(Gt,K,Ki);
    Gt=subs(Gt,t,ti);
    Gt=subs(Gt,s,s1);
    Gradt(j)=-real(double(((U)*(Gt)*(V))/(((U)*((Gs))*(V)))));
end
Gradt=reshape(Gradt,ts);
for i=1:ts(1)
    if (tmin(i)==tmax(i))
        Gradt(i)=0;
    end

end
end
```

1.5. Applying Grid Search to Find Actual Maximizing h

This function applies grid search to find actual maximizing h.

```
function [tnew,c] = max_c_h_grid(X,K,Ki,t,t0,tmin,tmax,region,
    a)
%   This function applyin a grid search to find the max c and
    related tnew.
cc=[];
ts=size(t0)
F=cell(ts)
tn={};
tnew=t0;
[~, c ] = computec( X,K,Ki,t,t0,region);
% plot(t0(2),c,'.r')
% hold on
% grid on
for i=1:1:ts(1)
    tn(i)={ndgrid(tmin(i):a:tmax(i))};
end
[F{:}]=ndgrid(tn{:});
```

```
gp=prod(size(F{1,1}))
for  j=1:1:gp
     for  i=1:1:ts(1)
     t0(i)=F{i,1}(j);
     end
     [~,  c1] = computec( X,K,Ki,t,t0,region);
%      plot(t0(2),c1,'.r')
     if  (real(c1)>real(c))
          c=c1;
          tnew=t0;
     end
     cc(j)=c1;
end


end
```

2. Initialization procedure Functions

Initialization procedure mainly implemented by two sub-functions in the Main function.

2.1. Checking Fixed Modes

First function checks if there is any fixed modes for all $h \in \Omega$ and at the same time finds the right most real open loop mode c_{rmrm} for all $h \in \Omega$

```
function  [c_rrm,Fixed,v] = start_2(A,B,C,D,t0,tmin,tmax,n,q,p,
    mu,a)
ts=size(t0);
F=cell(ts);
Fixed=[];
tn={};
d=0;
v=0;
c_rrm=mu
jj=1;
[E,Ac,~,~] =  closedsys_f(n,d,q,p,A,B,C,D,t0,[],1,[]);
```

```
Ki=zeros(p,q);
K=sym('K', [p,q]);
t=sym('t',[ts]);
[~,X] = char_matrix_d(E,Ac,t0);
for i=1:1:ts(1)
    tn(i)={ndgrid(tmin(i):a:tmax(i))};
end

[F{:}]=ndgrid(tn{:});
gp=prod(size(F{1,1}))
for j=1:1:gp
        for i=1:1:ts(1)
            t0(i)=F{i,1}(j);
        end

        [region ,c ,modes] = computeregion({eye(n)},A,t0,mu);
        size_modes=size(modes);
        ll=1;
        for ii=1:1:size_modes(1)
            if isreal(modes(ii))
                if modes(ii)>c_rrm
                    c_rrm=modes(ii)
                end
            end
            if (modes(ii)>=mu)
                [GradK] = gradMiMoK( X,Ki,t0,modes(ii) );
                if (norm(GradK)<0.00001)
                    Fixed=modes(ii);
                    disp(['Can not stabelize, fixed mode
                        detected'])
                    v=1;
                    break
                end
```

```
            end
        end
    if v==1
        break
    end
end
end
```

2.2. Finding the Region Where the Eigenvalues Should be Added for all $h \in \Omega$

Second function finds the region where the initial eigenvalues should be added to satisfy parity interlacing property for all $h \in \Omega$ and defines the minimum controller dimension

```
function [d,Feig,v,Reg_1] = start_3(A,B,C,D,t0,tmin,tmax,n,q,p
    ,mu,a ,c_rrm)
ts=size(t0);
F=cell(ts);
v=0;
tn={};
Reg_1=[];
Feig=[];
for i=1:1:ts(1)
    tn(i)={ndgrid(tmin(i):a:tmax(i))};
end
[F{:}]=ndgrid(tn{:});
gp=prod(size(F{1,1}))
for j=1:1:gp
    for i=1:1:ts(1)
        t0(i)=F{i,1}(j);
    end
    [~ ,c ,modes] = computeregion({eye(n)},A,t0,mu);
    [d,Reg]= init_free_par(A,B,C,D,t0,p,q,n,c_rrm,modes,mu);
    size_modes=size(modes);
```

```
  size_Reg=size(Reg)
  if j==1
      check_Reg=size(Reg)
      Reg_1=Reg;
  else
if size_Reg~=check_Reg
   v=1
   disp('can not stabilize , blocking zeros regions conflicts'
      )
else
    if ~isempty(Reg_1)
        r=1;
        while r<check_Reg(2)
            if Reg(r)>Reg_1(r)
                Reg_1(r)=Reg(r)
            end
            if Reg(r+1)<Reg_1(r+1)
                Reg_1(r+1)=Reg(r+1)
            end
            r=r+2
        end
    end
 end
end
if v==1
    break
end
end

for i=1:check_Reg(2)-1
    if (Reg_1(1,i+1))>(Reg_1(1,i))
    Feig(i)=((Reg_1(1,i+1))-(Reg_1(1,i)))/2+Reg_1(1,i)
    else
```

```
            v=1;
            break
        end
    end
    d=size(Feig)
    d=d(2)
    end
```

2.3. Finding the Region Where the Eigenvalues Should be Added for Specified h

This function finds the region where the initial eigenvalues should be added to satisfy parity interlacing property for specified $h \in \Omega$ and defines the minimum controller dimension for this h.

```
function [d,Reg] = init_free_par(A,B,C,D,t0,p,q,n,c_rrm,modes,
    mu)
% This function finds the transfer function of the system and
    finds the
% blocking zeros and the eigen values to be added by the
    controller Feig
%d is the min dim for controller
% n is the open loop system dim,q is the outputs number and p
    is the inputs number
syms s;
l= size(t0);
l= l(1);
CC=zeros(q,n);
DD=zeros(q,p);
BB=zeros(n,p);
AA=s*eye(n);
reg_bz=[]
Reg=[];
for i=1:1:l
    if (~isempty(C{i}))
```

```
        CC=CC+C{i}*exp(-s*t0(i));
    end
    if (~isempty(B{i}))
        BB=BB+B{i}*exp(-s*t0(i));
    end
    if (~isempty(D{i}))
        DD=DD+D{i}*exp(-s*t0(i));
    end
    if (~isempty(A{i}))
        AA=AA-A{i}*exp(-s*t0(i));
    end
end
AA=inv(AA);
TfM=CC*AA*BB+DD;
size_TfM=size(TfM);

region=[(mu-0.1),c_rrm+0.05,0,0]
ng=(round((region(1,2)-region(1,1))/0.01))+1;
R={};
Blocking_Z=[];
jj=1;
d=10;
for i=1:1:size_TfM(1)
    for j=1:1:size_TfM(2)
        if (TfM(i,j)~=0)
        F=matlabFunction(TfM(i,j));
        R{i,j}=(QPmR(region,F,1e-6,0.01));
        if isnan( R{i,j})
            R{i,j}=[];
            a=(region(1,3)+region(1,4))/1
            reg=1;
            while reg<2
                region=[region(1,1), region(1,2),region(1,3),
```

```
    region(1,3)+a];
Rn=QPmR(region,F,1e-6,0.01);
if isnan(Rn)
    Rn=[];
    a_1=(region(1,2)-region(1,1))/ng
    reg_1=1;
    while reg_1<(ng+1)
        region=[region(1,1), region(1,1)+a_1,
            region(1,3),region(1,4)];
        Rn=QPmR(region,F,1e-6,0.01);
          if isnan(Rn)
            Rn=[];
        end
        if ~isempty(Rn)
            siz_R=size(R{i,j})
            siz_Rn=size(Rn)
            R{i,j}(siz_R(1)+1:siz_R(1)+siz_Rn
                (1))=Rn;
            Rn=[];
        end
        reg_1=reg_1+1
        if reg_1>(ng)
       region(1,1)=region(1,1)-ng*a_1;
        else
              region(1,1)=region(1,2);

        end
    end
end


if ~isempty(Rn)
    siz_R=size(R{i,j})
```

```
                    siz_Rn=size(Rn)
                    R{i,j}(siz_R(1)+1:siz_R(1)+siz_Rn(1))=Rn;
                    Rn=[]
                end
                region(1,3)=region(1,4);
                reg=reg+1
            end
            if isempty(R{i,j})
                d=0;
            else
                size_R=size(R{i,j});
                for l=1:1:size_R(1)
                    T=subs(TfM,s,R{i,j}(l));
                    if (double(real(norm(T)))<1e-10)
                        Blocking_Z(jj)=R{i,j}(l);
                        jj=jj+1;
                    end
                end

            end
        end
        end
    end
end
if (d~=0)
    Blocking_Z=sort(real(Blocking_Z),'descend');
    Zc=zeros(size(Blocking_Z));
    for i=1:1:size(modes)
        for j=1:1:size(Blocking_Z)
            if real( modes(i))>real(Blocking_Z(j)) %Blocking_Z
               (j) sorted from max to min
                Zc(j)=Zc(j)+1;
                break
```

```
            end
        end
    end
    if size(Blocking_Z)>0
        for j=1:1:size(Blocking_Z)
            if (mod(Zc(j),2)==1)
                if j==1
                    reg_bz=[Blocking_Z(j),c_rrm];
                else
                    reg_bz=[Blocking_Z(j-1),Blocking_Z(j)];
                end
            end
            if ~isempty(reg_bz)
                    size_Reg=size(Reg)
                    Reg(1,size_Reg(2)+1:size_Reg(2)+2)=reg_bz;


        d=size_Reg(2)/2;
        d=d(1);
            end
        end
    else
        d=0;
    end


end

end
```

2.4. Obtain Controller Matrices

This function writes the free parameters into controller matrices F, G, H, K.

```
function [controller] = get_cont_matricies(Ki,m,p,q)
```

```
%extract the controller matricies from Ki
controller.G=Ki(1:m,1:q);
controller.F=Ki(1:m,q+1:q+m);
controller.K=Ki(m+1:m+p,1:q);
controller.H=Ki(m+1:m+p,q+1:q+m);
end
```

3. Obtain Closed-loop System Matrices and Initial Controller Parameters

This function creates the closed loop system matrices which contains the controller parameters, controller's free parameters are added to them as symbols, and the initial controller parameters Ki are constructed.

```
function [E,Ac,K,Ki] = closedsys_2(n,m,q,p,A,B,C,D,t,Feig,iter
    ,controller_form,mu)
%This function creats the controller matricies F G H K as one
    block in one
%of canonical forms either controllable or observable forms,
    and then it
%creats the cloosed loop system matricies E,Ac.
%   n is dimension of A
%m,q,p are the controller dimension, outputs numbers , and the
     input
%numbers
%A,B,C,D are the system matricies A={A0,A1,...,Ai}
%if there is no A1 then A={A0,[],...,Ai}
%K contains the free parameters of the controller
K=sym('K',[m+p,m+q])
Ki=randn((m+p),(m+q));

Feig_1=zeros(m,1);
size_Feig=size(Feig);
size_Feig=size_Feig(1);
for j_1=1:size_Feig
```

```
    Feig_1(j_1)=Feig(j_1);
end
for j_1=size_Feig+1:m
    Feig_1(j_1)=-(abs(randn(1,1)))+mu-0.1;
end
FF=flip(poly(Feig_1));
F_coef=-1*(FF(1,1:m));
H_coef=(eye(p,m));
for j=1:p
    H_coef(j,:)=( (H_coef(j,:)*randn(1,1)))/10
end
G_coef=(eye(m,q));
for j=1:m
    G_coef(j,:)= ((G_coef(j,:)*randn(1,1)))/10
end
K_coef=randn((p),(q))/10;

    if m>0
        if q>p
            % multivariable observable form

            if m>p
                K(1:p,q+1:q+m-p)=zeros(p,m-p)
                K(p+1:m,q+1:q+m-p)=eye(m-p)
                K(m+1:m+p,m+q-p+1:q+m)=eye(p)
                K(m+1:m+p,q+1:q+m-p)=zeros(p,m-p)
                Ki(1:p,q+1:q+m-p)=zeros(p,m-p)
                Ki(p+1:m,q+1:q+m-p)=eye(m-p)
                Ki(m+1:m+p,m+q-p+1:q+m)=eye(p)
                Ki(m+1:m+p,q+1:q+m-p)=zeros(p,m-p)
                Ki(1:m,m+q-p+1:m+q)= (F_coef);
                Ki(m+1:m+p,1:q)=K_coef
                Ki(1:m,1:q)=G_coef
```

```
        else
            K(m+1:m+m,q+1:q+m)=eye(m)
            Ki(m+1:m+m,q+1:q+m)=eye(m)
            Ki(1:m,q+1:q+m)=(F_coef);
            Ki(1:m,1:q)=G_coef;
            Ki(m+1:m+p,1:q)=K_coef

        end
    else
        % multivariable controllable canonical form

        if m>q
            K(1:m-q,q+1:2*q) =zeros(m-q,q)
            K(1:m-q,2*q+1:q+m)=eye(m-q)
            K(1:m-q,1:q)=zeros(m-q,q)
            K(m-q+1:m,1:q)=eye(q)
            Ki(1:m-q,q+1:2*q) =zeros(m-q,q)
            Ki(1:m-q,2*q+1:q+m)=eye(m-q)
            Ki(1:m-q,1:q)=zeros(m-q,q)
            Ki(m-q+1:m,1:q)=eye(q)
            Ki(m-q+1:m,q+1:q+m)=F_coef;
            Ki(m+1:m+p,1:q)=K_coef
            Ki(m+1:m+p,q+1:q+m)=H_coef;
        else
            if m~=0
                K(1:m,1:m)=eye(m)
                Ki(1:m,1:m)=eye(m)
                Ki(1:m,q+1:q+m)=F_coef;
                Ki(m+1:m+p,q+1:q+m)=H_coef;
                Ki(m+1:m+p,1:q)=K_coef
            end
```

```
            end
        end

    end
E=zeros((n+q+m+p),(n+q+m+p))
E(1:n,1:n)=eye(n)
E(1+n:n+q,1+n:n+q)=zeros(q,q)
E(1+n+q:n+q+m,1+n+q:n+q+m)=eye(m)
E(1+n+q+m:n+q+m+p,1+n+q+m:n+q+m+p)=zeros(p,p)

A0=zeros((n+q+m+p),(n+q+m+p))
if (isempty(B{1}))
    B{1}=zeros(n,p);
end
A0(1:n,n+q+m+1:n+q+m+p)=B{1}
if (isempty(D{1}))
    D{1}=zeros(q,p);
end
A0(1+n:n+q,n+q+m+1:n+q+m+p)=D{1}
if (isempty(A{1}))
    A{1}=zeros(n,n);
end
A0(1:n,1:n)=A{1}
if (isempty(C{1}))
    C{1}=zeros(q,n);
end
A0(1+n:n+q,1:n)=C{1}
A0(1+n:n+q,1+n:n+q)=-eye(q)

A0(1+n+q+m:n+q+m+p,1+n+q+m:n+q+m+p)=-eye(p)

FF=sym('f',[n+q+m+p,n+q+m+p])
FF(n+q+1:n+q+m+p,n+1:n+q+m)=K
```

```
FF=A0+FF
F=sym('f',[n+q+m+p,n+q+m+p])
FF=subs(FF,F,(zeros([n+q+m+p,n+q+m+p])))
A0=FF
l=size(t)
l=l(1)
Ac=cell(1,l);
Ac{1}=A0;

for i=2:1:l
    Ac{i}=zeros((n+q+m+p),(n+q+m+p))
    if isempty(B{i})
        B{i}=zeros(n,p);
    end
    Ac{i}(1:n,n+q+m+1:n+q+m+p)=B{i}
    if isempty(D{i})
        D{i}=zeros(q,p);
    end
    Ac{i}(1+n:n+q,n+q+m+1:n+q+m+p)=D{i}
    if isempty(A{i})
        A{i}=zeros(n,n)
    end
    Ac{i}(1:n,1:n)=A{i}
    if isempty(C{i})
        C{i}=zeros(q,n);
    end
    Ac{i}(1+n:n+q,1:n)=C{i}

end


end
```

4. Optimization Function

This function implements the steps 9 to 14 in Algorithm 3.

```
function [ Kopt, tn, c ] = RobustK2d(E,Ac,X,t,t0,tmin,tmax,K,
    Ki,region,mu,step_size,delay_step_size)
% Summary of this function goes here
% This function finds the minimum c by finding K* and h*
%K^* =Kopt
%h^*=tn
% where K is a matrix contains all free parameters of the
    controller.
%E, Ac contains the closed system matricies where Ac={A0,A1
    ,...,Ai}
%t0=[0;t1;t2;...;ti]
% tmax=[0;t_1_max;t_2_max;...;t_i_max]
% tmin=[0;t_1_min;t_2_min;...;t_i_min]
%X: is the cloosed loop charecteristic matrix.
%region is used for QPmR
%K contains the free parameters
m=1;
i_1=1;
tn=t0;
counter_iter=0;
f=1;
while f>0
    while (i_1>0)
        [ ~,c ] = computec( X,K,Ki,t,t0,region);
        K1=Ki;
        while (m>0)
            b=step_size;
            [GradK] = gradMiMoK( X,Ki,t0,c );
            if (norm(GradK)<0.0001)
                m=0;
                n=0;
                Knew=Ki;
            else
```

```
            n=1;
        end
        while (n==1)
            Knew=Ki-(b*GradK);
            [ ~,c1] = computec( X,K,Knew,t,t0,region);
            if (real(c1)>=real(c))
                if b>0.000009;
                    b=b/2 ;
                else
                    n=0;
                    Knew=Ki;
                    m=0;
                end
            else
                n=0;
                c=c1;
            end
        end
        Ki=Knew;
    end
    Ac1=Ac;% check if there is moodes out of region edit
       the region and back to minimization.
    Ac1{1}=double(subs(Ac1{1},K,Ki));
    [ region cch modes] = computeregion({E},Ac1,t0,mu);
    if isempty(cch)
        cch=c;
    end
    if (cch-c)<0.01
        m=2
    else
        m=1;
        Ki=K1;
    end
```

```
n=1;
ts=size(t0);
while (m>1)
    a=1;
    [ Gradt] = gradMiMot( X,Ki,t0,c,tmin,tmax );
    n=1;
    while (n==1)
        tnew=t0+a*Gradt;
        for i=1:ts(1)
            if (tnew(i)>=tmax(i))
                tnew(i)=tmax(i);
            end
            if (tnew(i)<=tmin(i))
                tnew(i)=tmin(i);
            end
        end
        [~, c1 ] = computec( X,K,Ki,t,tnew,region);
        if (real(c1)<real(c))
            if a>0.001
                a=a/2 ;
            else
                a=0;
                n=0;
            end
        else
            n=0;
            c=c1;
        end
    end
    if (norm(tnew-t0)<0.00005)
        m=0;
    end
    t0=tnew;
```

```
        end
        m=1;
        Ac1=Ac;
        Ac1{1}=double(subs(Ac1{1},K,Ki));
        [ region cc modes] = computeregion({E},Ac1,t0,mu);
        if isempty(cc)
            cc=c;
        end
        if (real(cc)-real(c))<0.01
            if (norm(tn-t0)<delay_step_size)
                i_1=0;
            else
                tn=t0;
                m=1;
            end
        end
    end
    [tg,cg] = max_c_h_grid(X,K,Ki,t,tn,tmin,tmax,region,
       delay_step_size)
    counter_iter=counter_iter+1;
    if counter_iter>3
        tn=tg;
        c=cg;
    end
    if (tn==tg)
        f=0;
    else tn=tg;
        t0=tg;
        i_1=1;
        m=1;
    end
end
Kopt=Ki;
```

```
end
```

5. Main Function

The main function is called RobustK4d, this function implements Algorithm 3.

```
function [K1, Ki, tm, s1 ,controller_form,modes_in_1] =
    RobustK4d(A,B,C,D,t0,tmin,tmax,n,dmax,q,p,mu,step_size,
    delay_step_size)
% n is the open loop system dim,dmax is a choosen max dim for
    controller,q
% is the outputs number and p is the inputs number
% step_size is the initial step size used in RobustK2d
    function
%  A={A_0,A_1,...,A_i}: (i.e) if there is no A_i related to
    t_i then A_i=[]
%but if there is no matrix related to t_i for B,C or D then it
     should be: []
%t0=[0;t1;t2;...;ti]
% tmax=[0;t_1_max;t_2_max;...;t_i_max]
% tmin=[0;t_1_min;t_2_min;...;t_i_min]
iter=0;
iter_max=5;% the number of initial points
controller_form={};
s1=[];
tm=[];
Ki=[]
K1={};
d=0;
Feig=[];
b=1;
[region ,~ ,modes_in_1] = computeregion({eye(n)},A,t0,mu);%%
    open loop modes
[c_rrm,fix,v] = start_2(A,B,C,D,t0,tmin,tmax,n,q,p,mu,
```

```
    delay_step_size)
if v~=1
    if c_rrm==mu
        d=0;
        Feig=[];
    else
        [d,Feig,v ,Reg_1] = start_3(A,B,C,D,t0,tmin,tmax,n,q,p
            ,mu,delay_step_size ,c_rrm)
    end
    if v~=1
        while (dmax>d)
            while iter<iter_max
                [E,Ac,K,Ki] = closedsys_f(n,d,q,p,A,B,C,D,t0,
                    Feig,b,controller_form,mu)
                [t,X] = char_matrix_d(E,Ac,t0);
                [ Kopti, tni, ci ] = RobustK2d(E,Ac,X,t,t0,
                    tmin,tmax,K,Ki,region,mu,step_size,
                    delay_step_size)
                if iter==0;
                    c=ci;
                    Kopt=Kopti
                    tn=tni
                    K1{b}=Ki;
                end
                if ci<c
                    c=ci;
                    Kopt=Kopti
                    tn=tni
                    K1{b}=Ki;
                end
                iter=iter+1;
            end
            iter=0;
```

```
                t0=tn;
                Ac1=Ac;
                Ac1{1}=double(subs(Ac1{1},K,Kopt));
                [region, c, modes_in] = computeregion({E},Ac1,tn,
                    mu);
                if (c<mu)
                    l=d;
                    d=dmax+1;
                else
                    l=d;
                    d=d+1;
                end
                [controller_form{b}]=get_cont_matricies(Kopt,l,p,q
                    );
                s1(b)=c;
                tm{b}=tn;
                b=b+1;
            end
        end
end
end
```

6. Algorithm 4

The main function can be used to implement Algorithm 4, after calling the function RobustK2d2 instead of calling the function RobustK2d.

```
function [ Kopt, tn, c ] = RobustK2d2(E,Ac,X,t,t0,tmin,tmax,K,
    Ki,region,mu,step_size,iter,c_iter)
% This function finds the minimum c by finding K* and h*
%K* =Kopt
%h*=tn
% where K contains all free parameters of the controller.
%E, Ac contains the closed system matricies where Ac={A0,A1
    ,...,Ai}
```

```
%t0=[0;t1;t2;...;ti]
% tmax=[0;t_1_max;t_2_max;...;t_i_max]
% tmin=[0;t_1_min;t_2_min;...;t_i_min]
%X: is the cloosed loop charecteristic matrix.
%region is used for QPmR
m=1;
n=1;
tn=t0;
Ac1=Ac;
Ac1{1}=double(subs(Ac1{1},K,Ki));
[region, c, modes_in] = computeregion({E},Ac1,t0,mu);
while (m>0)
    b=step_size;
    [GradK] = gradMiMoK( X,Ki,t0,c );
    if (norm(GradK))<0.0001
        m=0
    end
    n=0;
    while (n==0)

        Knew=Ki-(b*GradK);
        [tnew , c1 ]=max_c_h_grid(X,K,Knew,t,tn,tmin,tmax,
            region,0.1)
        if (real(c1)>real(c))
            if b>0.00001
                b=b/2 ;
            else
                n=1;
                m=0;
                n=1;
                Ki=Knew;
                c=c1;
            end
```

```
        else

            Ki=Knew;
            c=c1;
            t0=tnew;
            n=1;
        end

    end
end


[tn,c] = max_c_h_grid(X,K,Ki,t,tn,tmin,tmax,region,0.1)

Kopt=Ki;
end
```