****

**University Of Balamand**

**Faculty of Engineering**

**Optimal Control Systems**

**Balancing a Two-Wheeled Segway**

Presented to: Dr. Mohamed Khaldi

Presented by: Kifah Daher   
 Martine Chlela

Date: 17-01-2013

System description: 4

Theoretical Review: 8

I. Mathematical model of a system: 8

II. Testing the model: 11

Stability: 11

Controllability: 11

Observability: 12

Stabilizability: 13

Detectability: 13

III. State Feedback Control (SFC) Law: 15

i. Direct pole placement (DPP) 16

ii. Time domain design criteria (TDDC) 16

iii. Frequency domain design criteria (FDDC) 16

State Estimation: 17

IV. Full Order Observer: 17

Design Procedure: 18

FOO with SFC: 18

V. Reduced Order Observer: 19

ROO with SFC: 20

Program Simulation of the system using Matlab: 21

Pendulum position Analysis: 22

State Feedback Controller design with direct pole placement (DPP): 24

State Feedback Controller design with Time Domain Design Criteria (TDDC): 27

FOO: 29

FOO & SFC: 29

ROO & SFC : 29

Optimal SFC/FOO Design: 29

Cart position Analysis: 30

State Feedback Controller design with direct pole placement (DPP): 32

State Feedback Controller design with Time Domain Design Criteria (TDDC): 35

Full Order Observer design: 37

Full Order Observer design with State Feedback Control FOO & SFC using DPP: 39

Full Order Observer design with State Feedback Control FOO & SFC using TDDC: 42

Reduced Order Observer design with State Feedback Control ROO & SFC using DPP: 44

Reduced Order Observer design with State Feedback Control ROO & SFC using TDDC: 47

Optimal SFC/FOO Design: 50

Conclusion: 53

Appendix: 54

Matlab GUI code: 54

Table of figures 78

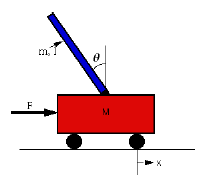
References: 79

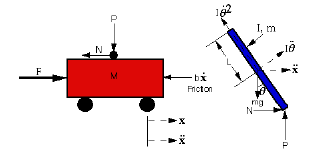
# System description:

The Segway is a two-wheeled self-balancing battery-powered electric vehicle. Controllers and motors in the base of the device keep the Segway upright. The Segway detects, as it balances, the change in its center of mass, fluid-based leveling sensors detect the weight shift. Segway are driven by electric motors and can go up to 12.5 miles per hour (20.1 km/h).



This problem is same as the classic problem of the inverted pendulum mounted on a cart. The cart must be moved so that mass **m** is always in an upright position. The state variables must be expressed in terms of the angular rotation (t) and the position of the cart x(t).





1-Summing the forces in the Free Body Diagram of the cart in the horizontal direction:



2-Summing the force in the free body diagram of the pendulum in the horizontal direction:



3-If you substitute this equation into the first equation; you get the first equation of motion on horizontal axis:



4-Summing the moments about the pendulum pivot point lead to:



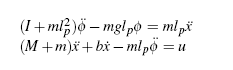
That needs to be combined with the sum of forces perpendicular to the pendulum:



Combining the two above, we get:



We then obtain the linear system:



State space model of the car - pendulum system:

By setting: y1 = x1 = x : the horizontal position of the cart

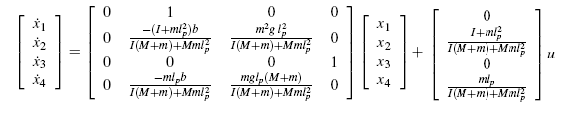
x2 = : the velocity of the cart

y2 = x3 = : the angular position of the pendulum

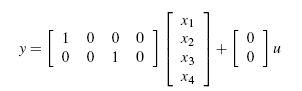
and x4 = : the angular velocity of the pendulum

We obtain the State Space Model of the segway:

The state variables are represented by:



And the output system is:



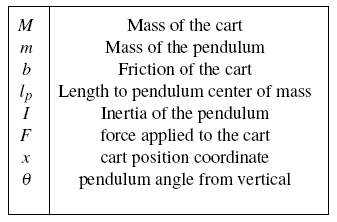


Table : System parameters

Our Segway model has the following characteristics:

|  |  |
| --- | --- |
| M | 38 Kg |
| m | 1 Kg |
| b | 0.1 N/m/sec |
| lp | 0.8 m |
| I | 0.006 Kg.m2 |

**NOTE:** The C matrix is 2 by 4, because both the cart’s position and the pendulum’s position are part of the output. For the state-space design problem we will be controlling a multi-output system so we will be observing the cart’s position from the first row and the pendulum’s with the second row.

We will analyze both systems separately and check if performance and estimation can be performed.

# Theoretical Review:

## Mathematical model of a system:

ODE:A Single Input Single Output system can be represented by an nth order linear ordinary differential equation:

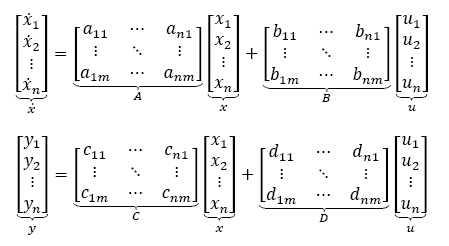


Transfer Function: To find the transfer function, first take the Laplace Transform of the differential equation (with zero initial conditions).  Recall that differentiation in the time domain is equivalent to multiplication by "s" in the Laplace domain. For the general case of an nth order differential equation with m derivatives of the input

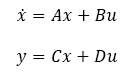


State Space Model: Linear time-invariant system theory (LTI) system theory has direct applications in circuits, signal processing, control theory, and other technical areas. For a LTI system of order n, and with r inputs, a set of n coupled first order linear differential equations with constant coefficients. Then a state-space model is a structured form or representation of a set of differential equations. State-space models (SSM) are very useful in Control theory and design. The differential equations are converted in matrices and vectors.

The vector form is the following:



A general linear State-space model may be written on the following general form:



A general block diagram of an nth order system is given below:

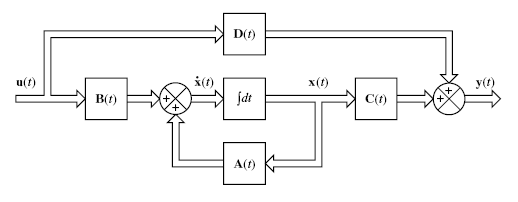
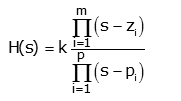


Figure : Block Diagram of a nth order system

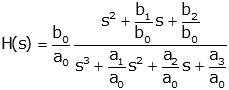
Pole/Zero: In the general case of a transfer function with an mth order numerator and an nth order denominator, the transfer function can be represented as:

****

For example consider the transfer function:

http://lpsa.swarthmore.edu/Representations/ZPK/img3.gif

If we rewrite this in a standard form such that the highest order term of the numerator and denominator are unity.



This is just a constant term (b0/a0) multiplied by a ratio of polynomials that can be factored.

http://lpsa.swarthmore.edu/Representations/ZPK/img7.gif

## Testing the model:

After deriving the mathematical model of a system, the system should be tested and investigated: Stability, Controllability, Observability, Delectability & Stabilizability are the fundamental tests for a system.

### Stability:

A system asymptotically stable if and only if every eigenvalue of A has strictly negative real part, that is: real{poles}<0

### Controllability:

Controllability is an important property of a [control system](http://en.wikipedia.org/wiki/Control_system), and the controllability property plays a crucial role in many control problems, such as stabilization of [unstable systems](http://en.wikipedia.org/wiki/BIBO_stability) by feedback, or optimal control.

Definition 1: If the input u(t) has access to the ith state variable, then this state variable is said to be controllable. If all state variables are controllable, then the system is said to be fully controllable. If one or more state variables are uncontrollable, then the system is said to be uncontrollable.

Definition 2: Given a Multiple Input Multiple Output (MIMO) state space model (SSM), then the SSM is controllable if and only if :

Rank [C] = n

Where C = [ B AB A2B… An-1B ] : Controllability Matrix

For Single Input Single Output (SISO) state space models (the controllability matrix is a square matrix in this case), the SSM is controllable if and only if:

Det (C) ≠ 0

Definition 3: Popov-Belevitch-Hartus Test (PBH Test)

PBH test states that the ith state variable is controllable if and only if:

ⱳiT B ≠ 0

Where ⱳi is the ith left eigenvector

### Observability:

Definition 1: If the output has access to the ith state variable, then this state variable is said to be observable. If all state variables are observable, then the system is said to be fully observable. If one or more state variables are unobservable, then the system is said to be unobservable.

Definition 2: Given a Multiple Input Multiple Output (MIMO) state space model (SSM), then the SSM is observable if and only if:

Rank [O] = n

Where O = [ C CA CA2 … CAn-1 ]T

: Observability Matrix

For Single Input Single Output (SISO) state space models (the observability matrix is a square matrix in this case), the SSM is observable if and only if:

Det (O) ≠ 0

Definition 3: Popov-Belevitch-Hartus Test (PBH Test)

PBH test states that the ith state variable is observable if and only if:

Cνi ≠ 0

Where νi is the ith right eigenvector

### Stabilizability:

If an unstable mode is controllable, then this mode is said to be stabilizable.

If all unstable modes are controllable, then the system is said to be stabilizable.

### Detectability:

If an unstable mode is observable, then this mode is said to be detectable.

If all unstable modes are observable, then the system is said to be detectable.

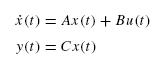
The system status: changing its performance or doing estimation can be assessed once the system is tested the table below describe the status of the system and what can be done to it.

|  |  |  |  |
| --- | --- | --- | --- |
| **Stability** | **Controllability** | **Observability** | **Status** |
| 1 | 0 | 0 | Keep |
| 1 | 0 | 1 | Keep and Estimate |
| 1 | 1 | 0 | Performance can be changed |
| 1 | 1 | 1 | Performance & Estimation |
| 0 | 0 | 0 | Redesign |
| 0 | 0 | 1 | Redesign |
| 0 | 1 | 0 | Stabilizable but not detectable |
| 0 | 1 | 1 | Stabilizable & detectable |

Table : System Status

## State Feedback Control (SFC) Law:

For a linear time-invariant state equation



This represents the open-loop system or plant to be controlled. Our focus is on the application of state feedback control laws of the form

**

The goal is achieving desired performance characteristics for the closed-loop state equation



The state feedback control law to include a gain matrix multiplying the reference input.

The state feedback control law features a constant state feedback gain matrix *K* of dimension *m* × *n* and a new external reference input *r(t)* necessarily having the same dimension *m* × 1 as the open-loop input *u(t)*.

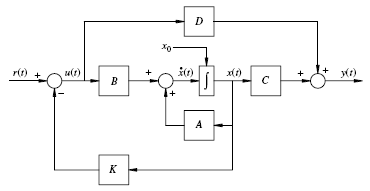


Figure : closed loop system block diagram

To change the performance of the system different method can be used so that the desired characteristic polynomial can be found by:

### Direct pole placement (DPP)

### Time domain design criteria (TDDC)

### Frequency domain design criteria (FDDC)

In DPP we can choose directly the desired poles to the system such that

where is the ith pole in a desired location.

In TDDC we specify the overshoot ‘Mp’ and the settling time ‘ts’ and then we compute

Where and Mp =

In FDDC are as well found and the desired characteristic polynomial is computed.

Finding the SFC gain K can be calculated by **Ackermann’s formula**

Note that the system should be controllable.

The compensated system is done by replacing the Matrix A in the SSM by Ak = A-BK;

And a pre filter ‘p’ is placed before the closed loop such a unit step response would have a zero steady state error ess=0. And Gcl =

## State Estimation:

An observer is a dynamic system Ŝ the purpose of which is to estimate the state of another dynamic system S using only the measured input and output of the latter. If the order Ŝ is equal to the order of S, then the observer is said to be of “Full-order”; if the order of Ŝ is less than the order of S, then the observer is said to be of “Reduced-order”.

## Full Order Observer:

A full-order observer accomplishes its purpose by calculating the “residual”, which is the difference between the measured output and the corresponding quantity generated by a model of S synthesized in the observer. The residual, multiplied by a “gain” is used as an input to a model of S. If the gain is chosen appropriately, the observer Ŝ will be an asymptotically stable dynamic system and the estimation error will converge to zero.

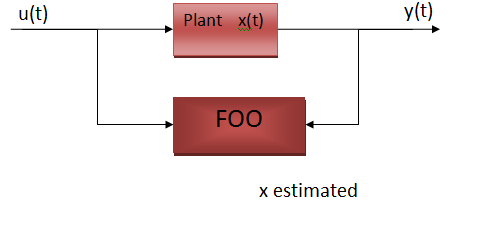


Figure : FOO Block Diagram

### Design Procedure:

1. Find the actual characteristic polynomial:

∆a(s)=|SI-AL|=|SI-A+LC|=0

1. Obtain the desired characteristic polynomial:

∆d(s) = Sn+dn-1Sn-1+…+d1S+d0. ∆d(s) can be found only by direct pole placement.

1. Equate ∆a(s) to ∆d(s), then solve for L or use modified Ackerman formula to find L:

L=∆d(A) O-1 .

The augmented system of the FOO: where T

y = [C 0]

### FOO with SFC:

The augmented system of the FOO with SFC becomes: where T

y = [C 0]

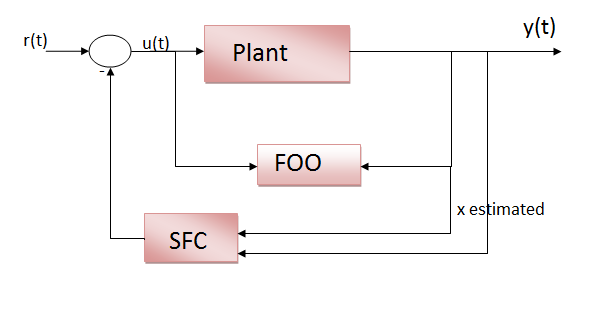


Figure : FOO & SFC Block Diagram

## Reduced Order Observer:

A reduced order observer for a dynamic system S is a dynamic system of order q = n-m, where n is the order of S and m is the number of observations. In addition to being more parsimonious of state variables, the reduced order observer may exhibit performance superior to that of the full-order observer, particularly in a closed-loop control system.

=+u

y = C

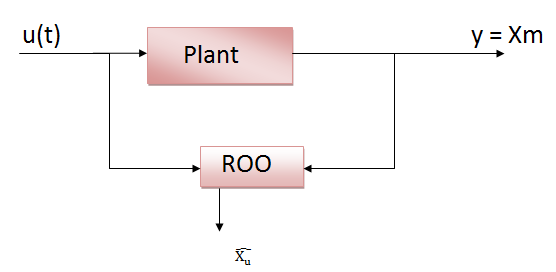


Figure : ROO Block Diagram

### ROO with SFC:

The augmented system of the ROO with SFC becomes: where T

y =

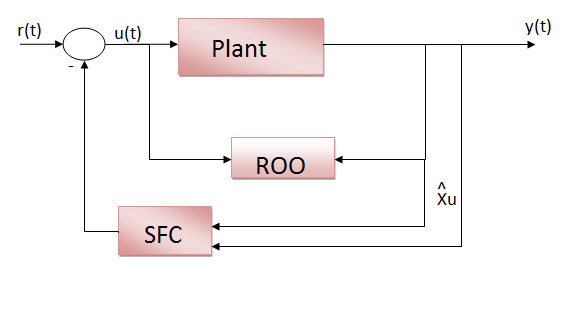


Figure : ROO and SFC Block Diagram

# Program Simulation of the system using Matlab:

We have used MATLAB to test, simulate and investigate the system. Two options are available a classical Matlab code where the user is prompt to enter the system parameters in text command form and a Graphical User Interface (GUI) method that allows the user to interact with the system using images rather than text commands.

The GUI interface is of the form shown in the figure below

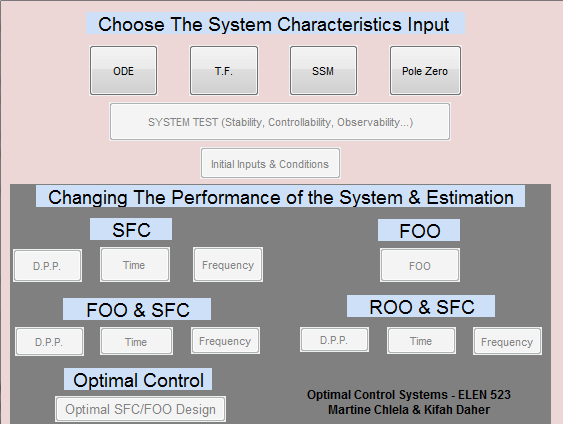


Figure : GUI figure based display

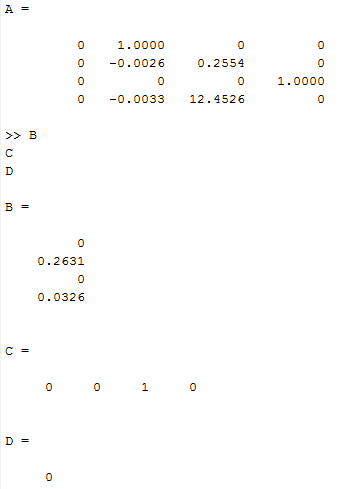
The user start inputting the system’s characteristics with a method of his choice then it is allowed to test the system so the performance and estimation pushbuttons are enabled since performance and estimation are based on the result test of stability, controllability and observavbility of the system as summarized in table 2.

As said before we will be observing the cart’s position from the first row of output and the pendulum’s with the second row separately.

## Pendulum position Analysis:

We will consider the second row of C matrix to investigate the pendulum position. So the user enters the system’s parameters for the chosen Segway model. For the C matrix the second row is only entered which allow us to study each output separately. C = [ 0 0 1 0 ]

The general output of the system is shown below representing the A, B, C & D matrix for the chosen system.



Once the user test the system by pressing the ‘System test’ pushbutton, the test is performed and the poles are shown as well the pole zero plot of the system.

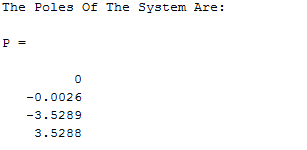
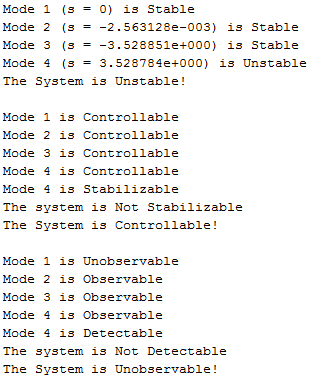


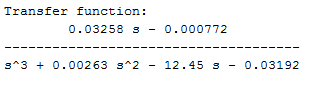
Figure : Pole zero plot of the system

As we can notice from the pole/zero plot we have two poles on the left hand side of the S plan and one to the right and side what make the system unstable.

Then in the Matlab Workspace the system test results are displayed as shown below; each mode is investigated alone and the overall system status is represented.



The program will display the transfer function of the system:



As it is shown above the each mode stability, controllability, observability, detectability and stabilizability are tested, and the overall system status is displayed. Since the system is Unobservable no estimation can be done, the program will display a warning noticing the user in case he tried to do estimation.

We will consider that an (SFC) State Feedback Controller is to be designed:

### State Feedback Controller design with direct pole placement (DPP):

The user must enter the desired poles location in order to design the SFC controller;

The desired poles given by the user are in the left hand side of the jw axis in the poles/zeros plan in order to make the system more stable they are the following:

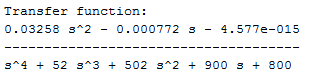
Then initial conditions are to be entered by the user in order to proceed in the design, in the initial Conditions pushbutton the user is asked to enter the simulation final time, the input type and the initial conditions for the state variables:

The user enters ‘10 sec’ for simulation final time and ‘unit step response’ for the input type. He is asked to enter the initial conditions.

We consider the following initial conditions: x0 = [0 0 -2 1]

Running the SFC controller design the program display the gain factor K and the new transfer function of the system:

The new transfer function is:



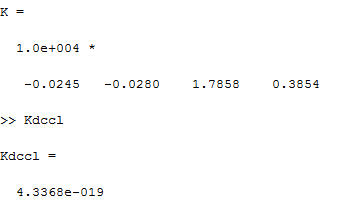


Figure 9 represents the simulated system for the pendulum position; it shows the uncompensated system that is unstable obviously and the compensated system via SFC controller with direct poles placement (DPP) which has a better performance.



Figure : SFC – DPP for the pendulum position analysis

In figure 9 the uncompensated system is plotted. It shows the instability of the inverted pendulum’s angular position without a controller. At t= 9sec the output of the uncompensated system deviates from a certain stable point or position.

Once the state feedback control is performed, the output becomes stable as shown at the bottom of figure 9. The pendulum starts from an angular position of -2 as entered in the initial condition then stabilize after a small overshoot. The cart will move in both direction until the pendulum is stable at zero position from vertical. This stability is due to the negative values (real part), hence stable position of the poles chosen by the user. The output of the compensated system contains a small overshoot at first, but then directly settles (at t=2.5 sec) to maintain a stable position. This position is the angular position of the inverted pendulum.

### State Feedback Controller design with Time Domain Design Criteria (TDDC):

As a second scenario the system is simulated using time domain design criteria TDDC where the user input the overshoot ‘Mp = 0.2’ and the settling time ‘ts = 5 sec’. The new poles of the system are shown below.

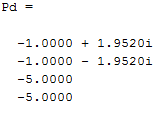


Figure 10 represents the simulated system for the pendulum position; it shows the uncompensated system that is unstable obviously and the compensated system via SFC controller using time domain design criteria (TDDC) which has a better performance.

The new transfer function of the compensated system via TDDC is:

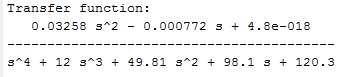




Figure : SFC – TDDC for the pendulum position analysis

In figure 10 the uncompensated system is plotted. It shows the instability of the inverted pendulum’s angular position without a controller. At t= 9sec the output of the uncompensated system deviates from a certain stable point or position (same behavior as when direct pole placement was applied).

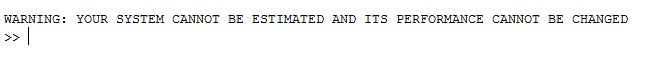
Once the state feedback control is performed, the output becomes stable as shown at the bottom of figure 9. As before the pendulum start from an angular position of -2 as per initial condition then oscillate while the cart trying to return the pendulum to the vertical and at the end it reaches the vertical position. The output of the compensated system contains a small overshoot at first (the intensity of this overshoot is due to the initial given value of Mp chosen by the user), but then directly settles (at t=4 sec) to maintain a stable position. This position is the angular position of the inverted pendulum, and the settling time depends on the value of ts given the user. Note that for a smaller value of maximum overshoot and settling time, the system could have had better performance.

Since the system is Unobservable no estimation can be done, and once the user try to simulate one of the estimation method by pressing the corresponding pushbutton a warning pop up.

### FOO:



### FOO & SFC:



### ROO & SFC :



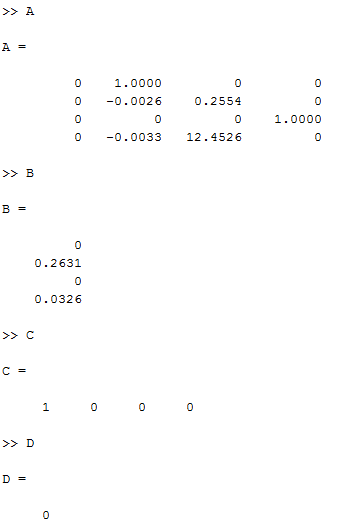
### Optimal SFC/FOO Design:

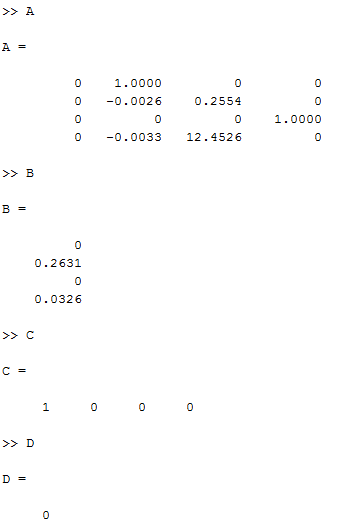


## Cart position Analysis:

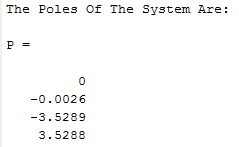
We will consider the first row of C matrix to investigate and observe the cart position. So the user enters the system’s parameters for the chosen Segway model. For the C matrix the first row is only entered. C = [ 1 0 0 0 ]

The general output of the system is shown below representing the A, B, C & D matrix for the chosen system.





Once the user test the system by pressing the ‘System test’ pushbutton, the test is performed and the poles are shown as well the pole zero plot of the system.



The transfer function of the system is:

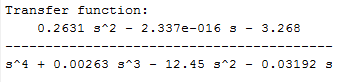
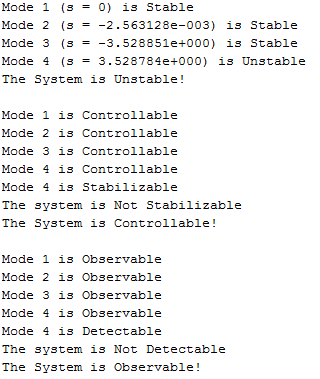




Figure : poles/zeros plot

In the Matlab Workspace the system test results are displayed as shown below; each mode is investigated alone and the overall system is represented.



The Matlab code that we wrote displayed that the overall system is unstable. This can be clearly noticed from the pz plot which shows the presence of a pole at the right hand side, that is its real part is positive, hence this mode is unstable, yielding to an unstable system.

### State Feedback Controller design with direct pole placement (DPP):

The user must enter the desired poles location in order to design the SFC controller;

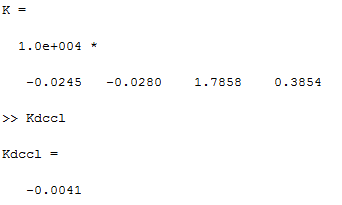
Here the system is controllable and an SFC design can be done without problem.

The desired poles given by the user are in the left hand side of the jw axis in the poles/zeros plan in order to make the system more stable they are the following:

The user enters ‘10 sec’ for simulation final time and ‘unit step response’ for the input type. He is asked to enter the initial conditions.

We consider the following initial conditions: x0 = [-1 0 -1 0]

Running the SFC controller design the program display the gain factor K and the transfer function:

****

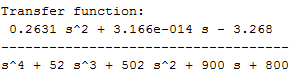
****

Figure 12 represents the simulated system; it shows the uncompensated system that is unstable obviously and the compensated system via SFC controller using Direct Pole Placement (DPP) which has a better performance.



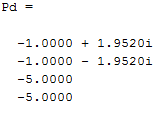
Figure : SFC – DPP for the cart position

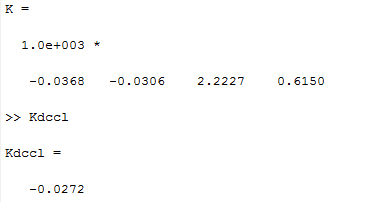
In figure 12 the uncompensated system is plotted. It shows the instability of the cart’s position without a controller. At t= 9sec the output of the uncompensated system deviates from a certain stable point or position.

Once the state feedback control is performed, the output becomes stable as shown at the bottom of figure 12. The cart start from a position of -2 as indicated in the initial condition start moving in order to stabilize the pendulum which cause a simple overshoot then directly goes to a stable position. The output of the compensated system contains a negligible overshoot at first, but then directly settles (at t=4 sec) to maintain a stable position. This position is the horizontal position of the cart which could have been predicted since the poles that we chose have negative real parts, hence are stable, making the system stable.

### State Feedback Controller design with Time Domain Design Criteria (TDDC):

As a second scenario the system is simulated using time domain design criteria TDDC where the user input the overshoot ‘Mp = 0.2’ and the settling time ‘ts = 5 sec’. The new poles of the system, the gain factor K and the new transfer function are shown below.





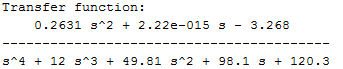


Figure 13 represents the simulated system for the cart position; it shows the uncompensated system that is unstable obviously and the compensated system via SFC controller using time domain design criteria (TDDC) which has a better performance.



Figure : SFC –TDDC for the cart position

In figure 13 the uncompensated system is plotted. It shows the instability of the cart’s position without a controller. At t= 9sec the output of the uncompensated system deviates from a certain stable point or position.

Once the state feedback control is performed, the output becomes stable as shown at the bottom of figure 13. The cart will change direction from -1 and keep oscillating until stabilizing the pendulum on the vertical position. The output of the compensated system contains a small overshoot at first, but then directly settles (at t=4.5 sec) to maintain a stable position. This position is the horizontal position of the cart which was stabilized in way that is depending on the value of Mp and ts given by the user. Certainly, decreasing the maximum peak and the settling time will yield to better performance and faster attainment of the stable position.

### Full Order Observer design:

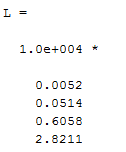
Another scenario where designing a FOO for the system. The user enters the desired poles (they in the left hand side of the jw axis in the poles/zeros plan in order to make the system more stable) they are the following:

The user enters ‘10 sec’ for simulation final time and ‘unit step response’ for the input type. He is asked to enter the initial conditions.

We consider the following initial conditions: x0 = [-1 0 -1 0] and the estimated initial condition xho = [1 0 -1 0]

Figure 14 represents the four states of the designed model versus the estimated ones.

The system solves for L and it is as following:



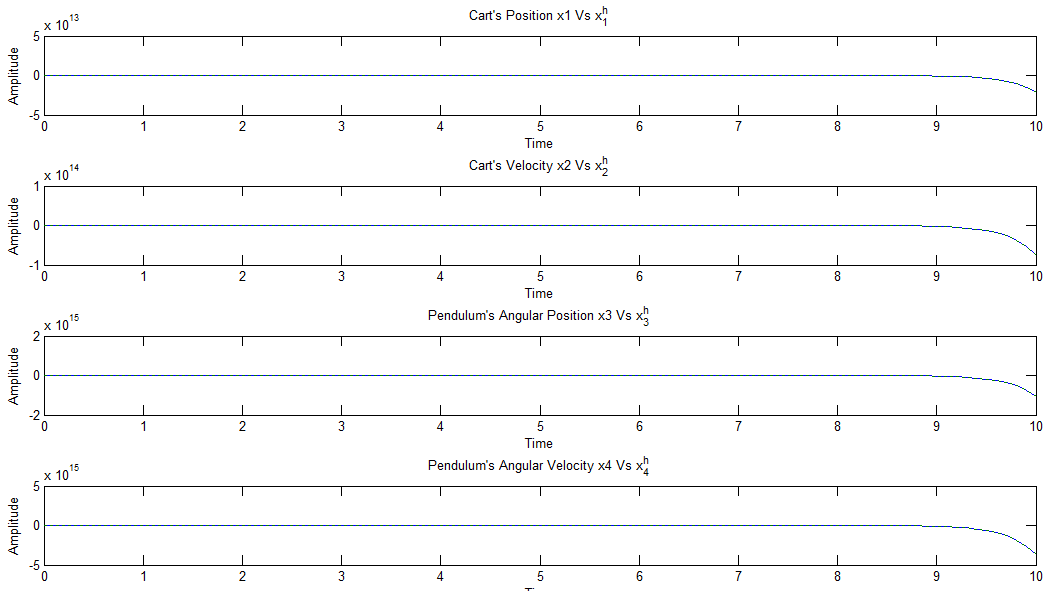


Figure : FOO design

As shown in figure 14, the states of the system cannot be stabilized but only estimation is performed. This is due to the presence of positive real valued poles shown in the pz plot. What is more, we can notice that the estimated state variables are confused with the original state variables. That is, the estimation error directly tended to zero. In order to achieve better performance of the system, a state feedback controller should be integrated.

### Full Order Observer design with State Feedback Control FOO & SFC using DPP:

Another scenario where designing a FOO for the system. The user enters the desired poles (they in the left hand side of the jw axis in the poles/zeros plan in order to make the system more stable) they are the following:

The user enters ‘10 sec’ for simulation final time and ‘unit step response’ for the input type. He is asked to enter the initial conditions.

We consider the following initial conditions: x0 = [-1 0 -1 0] and the estimated initial condition xho = [-1 -2 1 0]

The system solves for L and the gain K and are as following:

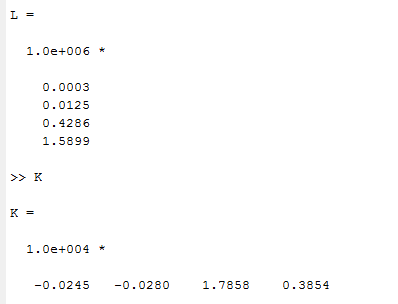


Figure 15 represents the four states of the designed model versus the estimated ones.

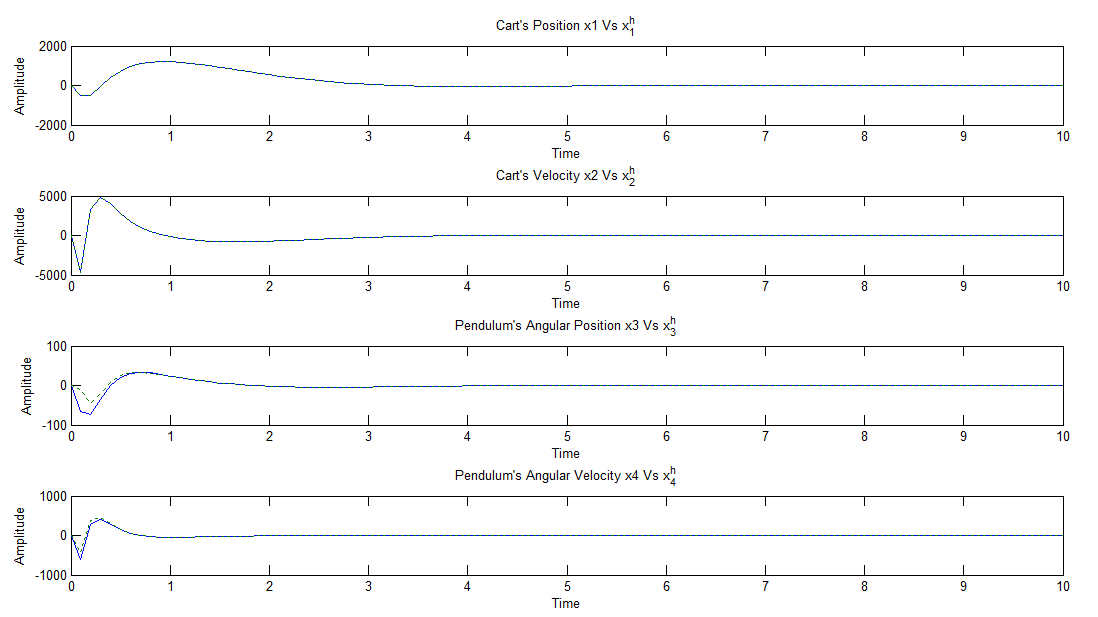


Figure : FOO & SFC with DPP

As stated previously, the addition of a state feedback controller will yield to better results hence a stable system. This is clearly shown in figure 15, where a full-order observer was implemented with the presence of a state feedback controller. The system’s state variables reached stability after a certain overshoot. Furthermore, we can notice the trajectory of the estimated state variables which is approximately the same as the one which the original state variables followed. That is, the estimation error is very small.

Figure 16 represents the actuating signal and the output response for the FOO & SFC designed controller using DPP.



Figure : Actuating signal and output response for the designed FOO & SFC (DPP)

At the top of figure 16, the actuating signal or the so called error signal is shown. This signal shows the difference between the estimated and the reference signal, which is negligible after the system reached stability; this yields better estimation.

We can observe in the bottom of figure 16 the output response of the cart, y, which is equal to x1 which is nothing but the cart’s horizontal position. We can clearly observe the improved performance of this output once the state feedback controller and the full-order observer were added to the system. Once the desired negative real valued (stable) poles were placed, the system reached stability, hence better performance. The horizontal position of the cart became more stable, hence the Segway trajectory more secure.

### Full Order Observer design with State Feedback Control FOO & SFC using TDDC:

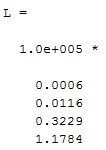
As a second scenario the system with FOO & SFC controller is simulated using time domain design criteria TDDC where the user input the overshoot ‘Mp = 0.2’ and the settling time ‘ts = 5 sec’. The system solves for L and it is as below:  


Figure 17 represents the four states of the designed model versus the estimated ones.

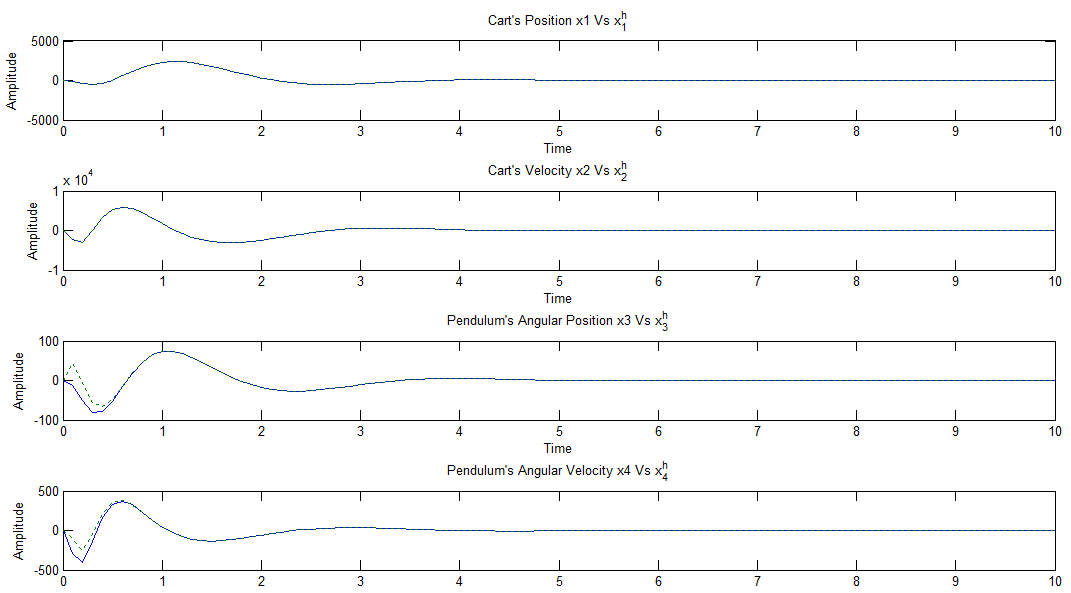


Figure : FOO & SFC with TDDC

Figure 18 represents the actuating signal and the output response for the FOO & SFC designed controller using TDDC.



Figure : Actuating signal and output response for the designed FOO & SFC (TDDC)

At the top of figure 18, the actuating signal or the so called error signal is shown. This signal shows the difference between the estimated and the reference signal, which is negligible after the system reached stability; this yields better estimation.

We can observe in the bottom of figure 18 the output response of the cart, y, which is equal to x1 which is nothing but the cart’s horizontal position. We can clearly observe the improved performance of this output once the state feedback controller and the full-order observer were added to the system. The horizontal position of the cart became more stable, hence the segway trajectory more secure. Note that, if the values of the maximum overshoot and settling time chosen by the user were decreased, better performance and faster attainment of stability will result.

### Reduced Order Observer design with State Feedback Control ROO & SFC using DPP:

In another scenario the system with ROO & SFC controller is designed and simulated using Direct Pole Placement (DPP) where the user enters the desired poles (they in the left hand side of the jw axis in the poles/zeros plan in order to make the system more stable) they are the following:

As well the user enters the ROO desired poles, considering the first state measured, three unmeasured states remains so the user enter three desired ROO poles :

The user enters ‘10 sec’ for simulation final time and ‘unit step response’ for the input type. He is asked to enter the initial conditions.

We consider the following initial conditions: x0 = [-1 0 -1 0] and the estimated initial condition xoh = [-1 -2 1 ] which are three corresponding to the unmeasured states.

The program will tell the user that the first state is considered measured and the remaining ones are unmeasured.

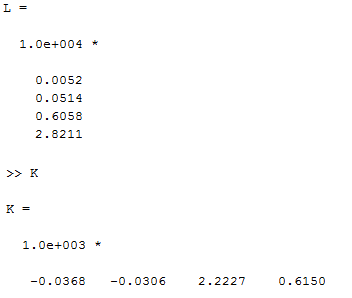
The system solves for L and the SFC gain K. they as following:

Figure 19 represents the four states of the designed model versus the estimated ones.

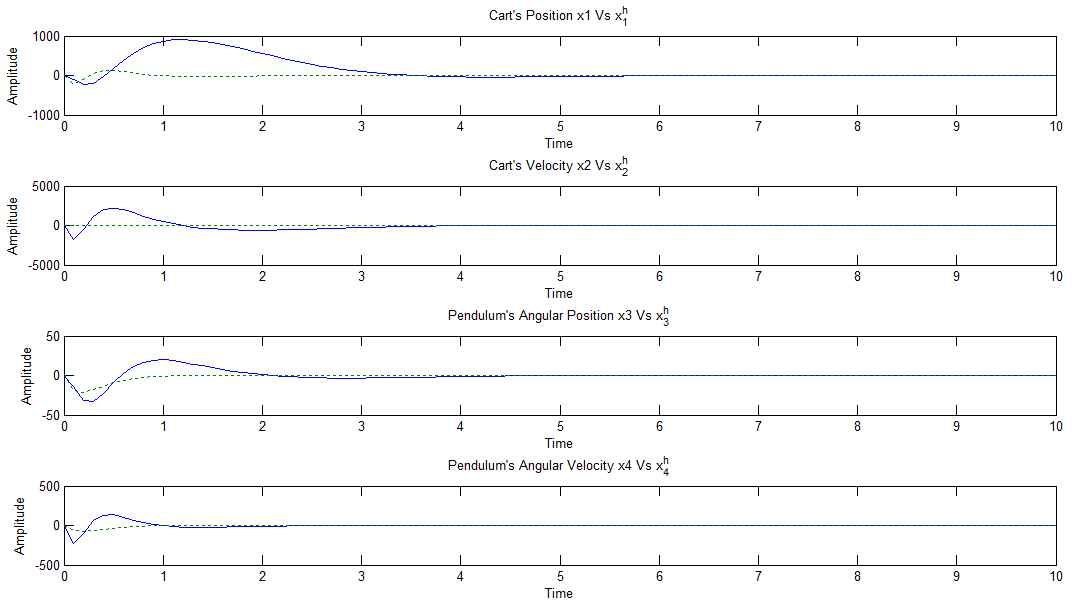


Figure : ROO & SFC (DPP)

As stated previously, the addition of a state feedback controller will yield to better results hence stable system. This is clearly shown in figure 19, where a reduced-order observer was implemented with the presence of a state feedback controller. The system’s state variables reached stability after a certain overshoot. Furthermore, we can notice the trajectory of the estimated state variables follow the original state variables. In the reduced-order observer the estimation error is larger than in full order observer which is logical.

Figure 20 represents the actuating signal and the output response for the ROO & SFC designed controller using DPP.



Figure : Actuating signal and output response for the designed ROO & SFC (DPP)

At the top of figure 20, the actuating signal or the so called error signal is shown. This signal shows the difference between the estimated and the reference signal, which is negligible after the system reached stability. However, once this actuating signal is compared to the one obtained when the estimation was performed using a full order observer, delayed attainment of zero estimation error is observed.

We can observe in the bottom of figure 20 the output response of the cart, y, which is equal to x1 which is nothing but the cart’s horizontal position. We can clearly observe the improved performance of this output once the state feedback controller and the reduced-order observer were added to the system. The horizontal position of the cart became more stable, hence the segway trajectory more secure. Comparing this signal to the one obtained when a full order observer (also in DPP) was used reaching stability was delayed.

### Reduced Order Observer design with State Feedback Control ROO & SFC using TDDC:

As a second scenario, the system with ROO & SFC controller is designed and simulated using time domain design criteria TDDC where the user input the overshoot ‘Mp = 0.2’ and the settling time ‘ts = 5 sec’. The same ROO poles are assigned [ -5 -5 -5]

Figure 20 represents the four states of the designed model versus the estimated ones.

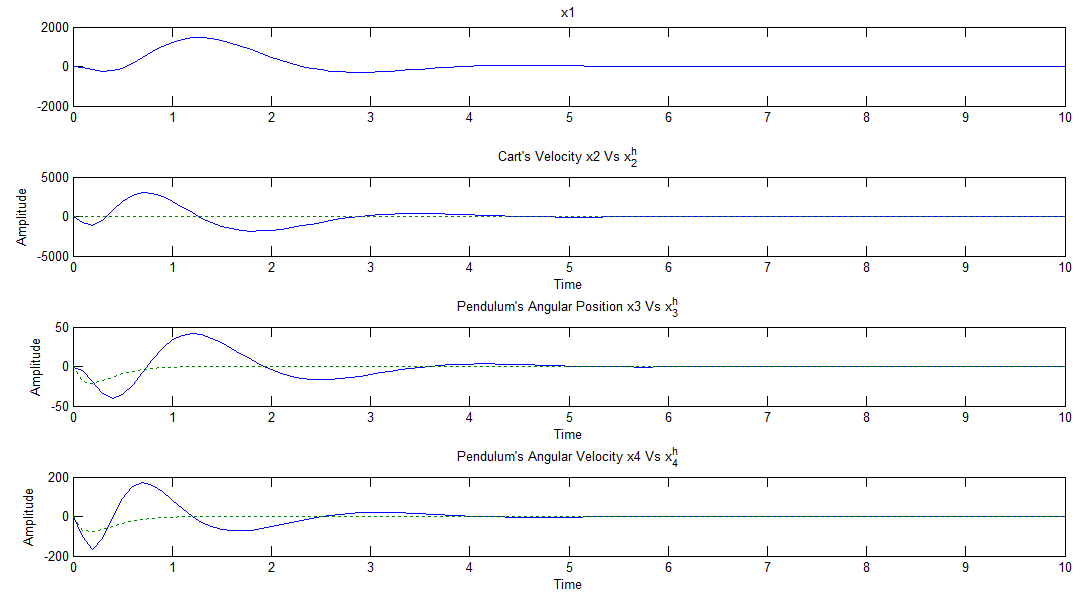


Figure : ROO & SFC using TDDC

Figure 21 shows the estimated and the original state variables when time domain design criteria was used. When comparing these outputs to the ones obtained by direct pole placement, we notice that the states reached stable operating points only when the settling time (given by the user to be 5sec) was elapsed. However, when these signals are compared to the ones obtained by the full-order observer estimation, we notice that steady state was reached faster in the latter estimation technique. Better performance can certainly be achieved if the settling time is reduced.



Figure : Actuating signal and output response for the designed ROO & SFC (TDDC)

Figure 22 highlights the fact that a zero estimation error is achieved after a relatively large period of time (in comparison with the FOO&SFC design); this is illustrated by the oscillations viewed in the figure. However, the system will finally reach this desired negligible error. What is more, when observing the output response, one can clearly notice the stable operating point reached after 5sec (which is the settling time entered by the user). In addition to that, smaller maximum peak and settling time will certainly yield better performance.

### Optimal SFC/FOO Design:

Another scenario, the system with optimal FOO/SFC controller is designed and simulated.

Figure 21 represents the uncompensated system for the four states, Figure 22 represents the compensated four variables versus the estimated ones and Figure 23 shows the output of the compensated system and the optimal effort. The given initial conditions by the user are   
xo = [-1 0 -1 0]

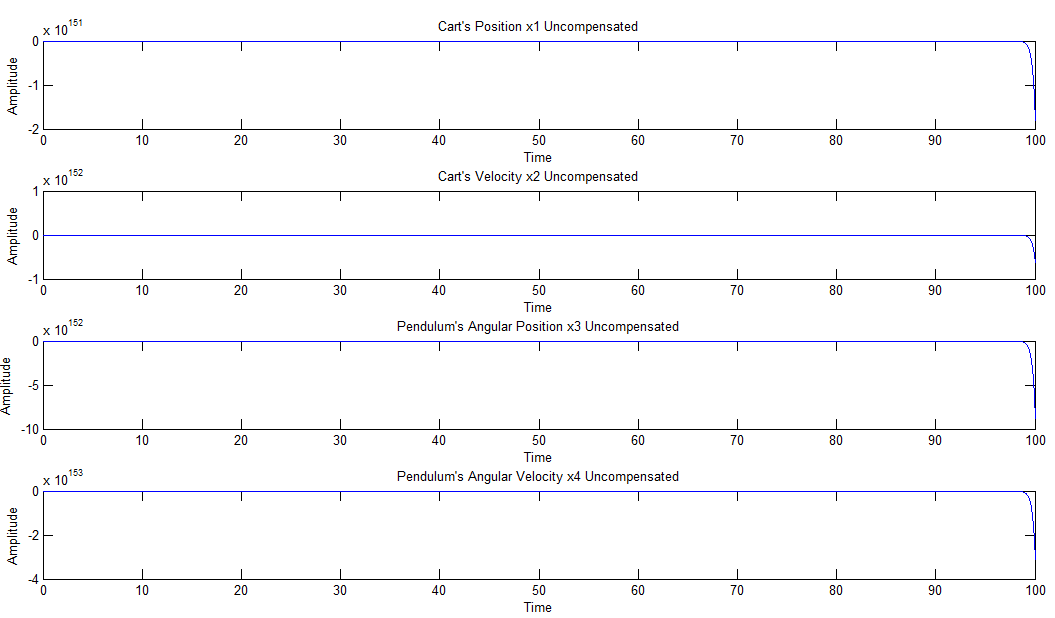


Figure : the uncompensated state variables for Optimal SFC/FOO Design

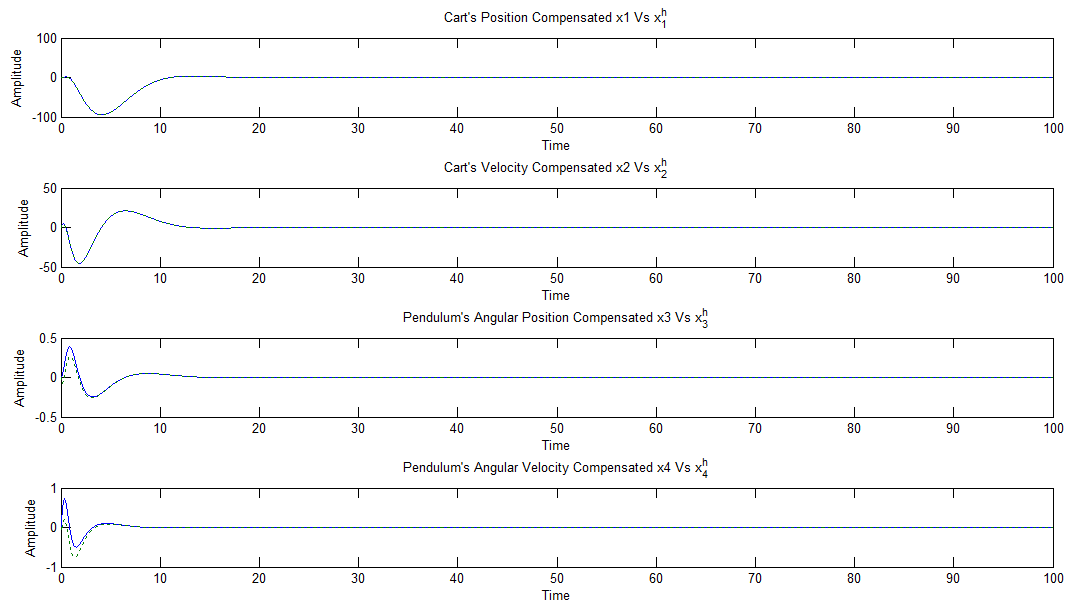


Figure : the compensated state variables for Optimal SFC/FOO Design

Figure 24 shows how the pendulum’s and the cart position are stabilized after a certain time. The cart will oscillate to stabilize the pendulum. The pendulum will goes in both direction until reaching and stabilizing on the vertical position.



Figure : the compensated system output & the optimal effort

# Conclusion:

Control theory is in engineering uses mathematics to deal with the behavior of dynamical systems with inputs. The external input of a system is the reference that the output variables of a system need to follow. A controller is designed to obtain the desired effect on the output of the system by manipulating the inputs. The usual objective is to calculate solutions for the proper action the controller will do that result in system stability.

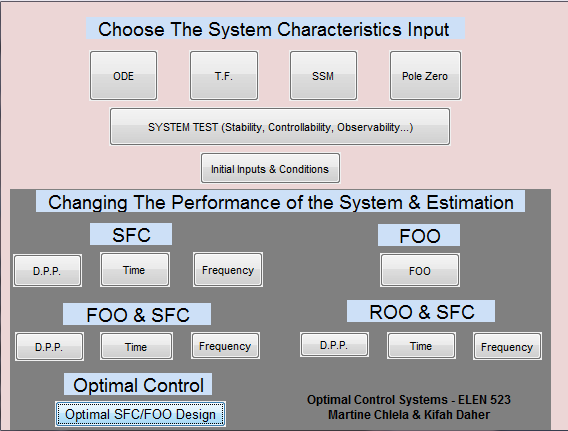
Optimal control finds a proper control law for a given system with optimality criterions taken into consideration. An optimal control is represented by a set of differential equations that describe the paths of the variables and minimize the cost functional.

Different type of controllers has been investigated in this project such as State Feedback Control with different criterions such as Direct Pole Placement, Time Domain Design Criteria and Frequency Domain Design Criteria that can improve the performance of the system if it is controllable. If the system is observable estimation can be done. Full Order Observer and Reduced Order Observer controllers mixed with the SFC can improve the performance of the system with an estimation of the results.

Controlling the Segway was similar of the classical inverted pendulum model. With the designed controllers the system is stable. The pendulum will stabilize in the vertical direction after a certain time. The cart will change direction and oscillate until stabilizing the pendulum. When the system is stabilized, all the cart’s position and the pendulum’s position are in the desired location.

# Appendix:

## Matlab GUI code:



function varargout = OptimalGUI(varargin)

% OPTIMALGUI MATLAB code for OptimalGUI.fig

% OPTIMALGUI, by itself, creates a new OPTIMALGUI or raises the existing

% singleton\*.

%

% H = OPTIMALGUI returns the handle to a new OPTIMALGUI or the handle to

% the existing singleton\*.

%

% OPTIMALGUI('CALLBACK',hObject,eventData,handles,...) calls the local

% function named CALLBACK in OPTIMALGUI.M with the given input arguments.

%

% OPTIMALGUI('Property','Value',...) creates a new OPTIMALGUI or raises the

% existing singleton\*. Starting from the left, property value pairs are

% applied to the GUI before OptimalGUI\_OpeningFcn gets called. An

% unrecognized property name or invalid value makes property application

% stop. All inputs are passed to OptimalGUI\_OpeningFcn via varargin.

%

% \*See GUI Options on GUIDE's Tools menu. Choose "GUI allows only one

% instance to run (singleton)".

%

% See also: GUIDE, GUIDATA, GUIHANDLES

% Edit the above text to modify the response to help OptimalGUI

% Last Modified by GUIDE v2.5 18-Jan-2013 22:14:38

% Begin initialization code - DO NOT EDIT

gui\_Singleton = 1;

gui\_State = struct('gui\_Name', mfilename, ...

'gui\_Singleton', gui\_Singleton, ...

'gui\_OpeningFcn', @OptimalGUI\_OpeningFcn, ...

'gui\_OutputFcn', @OptimalGUI\_OutputFcn, ...

'gui\_LayoutFcn', [] , ...

'gui\_Callback', []);

if nargin && ischar(varargin{1})

gui\_State.gui\_Callback = str2func(varargin{1});

end

if nargout

[varargout{1:nargout}] = gui\_mainfcn(gui\_State, varargin{:});

else

gui\_mainfcn(gui\_State, varargin{:});

end

% End initialization code - DO NOT EDIT

% --- Executes just before OptimalGUI is made visible.

function OptimalGUI\_OpeningFcn(hObject, eventdata, handles, varargin)

% This function has no output args, see OutputFcn.

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% varargin command line arguments to OptimalGUI (see VARARGIN)

% Choose default command line output for OptimalGUI

handles.output = hObject;

% Update handles structure

guidata(hObject, handles);

% UIWAIT makes OptimalGUI wait for user response (see UIRESUME)

% uiwait(handles.figure1);

% --- Outputs from this function are returned to the command line.

function varargout = OptimalGUI\_OutputFcn(hObject, eventdata, handles)

% varargout cell array for returning output args (see VARARGOUT);

% hObject handle to figure

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Get default command line output from handles structure

varargout{1} = handles.output;

% --- Executes on button press in pushbutton1.

function pushbutton1\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton1 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

set(handles.pushbutton21, 'Enable', 'on');

clc

clear all

count1 =0;

disp ('Enter the output parameters of the ODE starting with the lowest order coefficient,when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count1=count1+1;

end

y=cellfun(@str2num,b(1:count1))

count2 =0;

disp ('Enter the input parameters of the ODE starting with the lowest order coefficient, when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count2=count2+1;

end

u=cellfun(@str2num,b(1:count2))

[A,B,C,D] = TF2SS(u,y);

save();

% --- Executes on button press in pushbutton2.

function pushbutton2\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton2 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

set(handles.pushbutton21, 'Enable', 'on');

clc

clear all

count1 =0;

disp ('Enter the denominator parameters of the Transfer Function starting with the lowest order coefficient,when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count1=count1+1;

end

y=cellfun(@str2num,b(1:count1))

count2 =0;

disp ('Enter the numerator parameters of the Transfer Function starting with the lowest order coefficient, when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count2=count2+1;

end

u=cellfun(@str2num,b(1:count2))

% G=tf(u,y)

[A,B,C,D] = TF2SS(u,y);

save();

% --- Executes on button press in pushbutton3.

function pushbutton3\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton3 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

set(handles.pushbutton21, 'Enable', 'on');

clc

clear all

disp ('Enter the size "n" of A such that A:n\*n')

n=input('');

A=zeros(n);

disp('Enter the A matrix coefficients row by row')

for i=1:n

for j=1:n

A(i,j)=input('');

end

end

disp ('Enter the column size "r" of B such that B:n\*r')

r=input('');

B=zeros(n,r);

disp('Enter the B matrix coefficients row by row')

for i=1:n

for j=1:r

B(i,j)=input('');

end

end

disp ('Enter the row size "m" of C such that C:m\*n')

m=input('');

C=zeros(m,n);

disp('Enter the C matrix coefficients row by row')

for i=1:m

for j=1:n

C(i,j)=input('');

end

end

D=zeros(m,r);

disp('Enter the D matrix coefficients row by row D:m\*r')

for i=1:m

for j=1:r

D(i,j)=input('');

end

end

A

B

C

D

save();

% --- Executes on button press in pushbutton4.

function pushbutton4\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton4 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

set(handles.pushbutton21, 'Enable', 'on');

clc

clear all

count1 =0;

disp ('Enter the zeros,when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count1=count1+1;

end

z=cellfun(@str2num,b(1:count1))

count2 =0;

disp ('Enter the poles, when done press q')

for i=1:100

b{i}=input('','s');

if b{i}=='q'

break

end

count2=count2+1;

end

p=cellfun(@str2num,b(1:count2))

disp ('Enter the gain factor k')

k=input('')

[A,B,C,D] = ZP2SS(z,p,k);

save();

% --- Executes on button press in pushbutton5.

function pushbutton5\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton5 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO == 0

fprintf('WARNING: YOUR SYSTEM IS UNCONTROLLABLE!! YOU CANNOT CHANGE ITS PERFORMANCE!!\n')

return;

end

disp('Enter the desired poles in complex "a+bi" form:')

for i=1:length(P)

a=input('Enter the real part "a" parameter: \n');

b=input('Enter the imaginary part "b" parameter: \n');

Pd(i)=complex(a,b);

end

Pd

% State Feedback Control: Ackerman Formula

K=acker(A,B,Pd);

% The Compensated System

Ak = A-B\*K;

Gcl = ss(Ak,B,C,D);

Kdccl = -C/Ak\*B; % Pre-Filter N=1/Kdccl

if Kdccl<1

Kdccl=1;

end

Gcl0sse = Gcl/Kdccl;

tf(Gcl0sse)

% Simulation

figure

% lsim(G,':k',Gcl,'-.k',Gcl0sse,'k',r,t,xo')

subplot(2,1,1);lsim(G,'b',r,t,xo')

str1= sprintf('%s Uncompensated system',sp);

title(str1)

subplot(2,1,2);lsim(Gcl0sse,'b',r,t,xo')

str2= sprintf('%s Compensated system with zero steady state error',sp);

title(str2)

save();

% --- Executes on button press in pushbutton6.

function pushbutton6\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton6 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO == 0

fprintf('WARNING: YOUR SYSTEM IS UNCONTROLLABLE!! YOU CANNOT CHANGE ITS PERFORMANCE!!\n')

return;

end

Mp = input('Enter the Overshoot factor Mp:\n');

ts = input('Enter the Settling time ts:\n');

x = log(Mp)^2;

zeta = sqrt(x/(pi^2+x));

wn = 5/(zeta\*ts);

Do=[1 2\*zeta\*wn wn^2];

Pd = roots(Do); %Dominant Poles

if max(size(Pd))<max(size(A))

for q=1:(max(size(A))-size(Pd))

Pnd(q)=5\*real(Pd(q));

end

Pd=[Pd;Pnd']

end

% State Feedback Control: Ackerman Formula

K=acker(A,B,Pd);

% The Compensated System

Ak = A-B\*K;

Gcl = ss(Ak,B,C,D);

Kdccl = -C/Ak\*B; % Pre-Filter N=1/Kdccl

if Kdccl<1

Kdccl=1;

end

Gcl0sse = Gcl/Kdccl;

tf(Gcl0sse)

% Simulation

figure

% lsim(G,':k',Gcl,'-.k',Gcl0sse,'k',r,t,xo')

subplot(2,1,1);lsim(G,'b',r,t,xo')

str1= sprintf('%s Uncompensated system',sp);

title(str1)

subplot(2,1,2);lsim(Gcl0sse,'b',r,t,xo')

str2= sprintf('%s Compensated system with zero steady state error',sp);

title(str2)

save()

% --- Executes on button press in pushbutton7.

function pushbutton7\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton7 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO == 0

fprintf('WARNING: YOUR SYSTEM IS UNCONTROLLABLE!! YOU CANNOT CHANGE ITS PERFORMANCE!!\n')

return;

end

zeta = input('Enter the dapmping factor :\n');

wn = input('Enter the natural frequency wn \n');

Do=[1 2\*zeta\*wn wn^2];

Pd = roots(Do); %Dominant Poles

if max(size(Pd))<max(size(A))

for q=1:(max(size(A))-size(Pd))

Pnd(q)=5\*real(Pd(q));

end

Pd=[Pd;Pnd']

end

% State Feedback Control: Ackerman Formula

K=acker(A,B,Pd);

% The Compensated System

Ak = A-B\*K;

Gcl = ss(Ak,B,C,D);

Kdccl = -C/Ak\*B; % Pre-Filter N=1/Kdccl

if Kdccl<1

Kdccl=1;

end

Gcl0sse = Gcl/Kdccl;

% Simulation

figure

subplot(2,1,1);lsim(G,'b',r,t,xo')

str1 = sprintf('%s Uncompensated system',sp);

title(str1)

subplot(2,1,2);lsim(Gcl0sse,'b',r,t,xo')

str2= sprintf('%s Compensated system with zero steady state error',sp);

title(str2)

% lsim(G,':k',Gcl,'-.k',Gcl0sse,'k',r,t,xo')

% --- Executes on button press in pushbutton8.

function pushbutton8\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton8 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if (St==1&&CO==0&&Ob==1)||(CO==1&&Ob==1)

ma=max(size(A));

% FOO

disp('Enter the initial estimations xho one by one followed by return:')

for i=1:ma

xho(i)=input('');

end

xho=xho';

disp('Enter the closed loop desired poles in a+bj form one by one followed by return')

for i=1:ma

P(i)=input('');

end

% Ackerman Formula

L = acker(A',C',P)'

% Augmented System

AL = [A zeros(max(size(A)));L\*C A-L\*C];

BL = [B;B];

CL = [C zeros(1,ma)];

Ga = ss(AL,BL,CL,0);

% Simulation

xto = [xo;xho];

[y,t,xk] = lsim(Ga,r,t,xto);

figure

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xk(:,a),t,xk(:,a+ma),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED!!\n')

return;

end

save()

% --- Executes on button press in pushbutton9.

function pushbutton9\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton9 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% FOO & SFC

disp('Enter the inital estimations xho one by one followed by return:')

for j=1:ma

xho(j)=input('');

end

xho=xho';

disp('Enter the closed loop desired poles in complex "a+bi" form:')

for i=1:ma

a=input('Enter the real part "a" parameter: \n');

b=input('Enter the imaginary part "b" parameter: \n');

Pk(i)=complex(a,b);

end

Pk=Pk';

% State Feedback Control

K = acker(A,B,Pk);

Ak = A-B\*K;

Ck = C-D\*K;

P = 1/(D-Ck/Ak\*B);

Bk = P\*B;

Dk = P\*D;

% Full Order Observer

Pl = 5\*real(Pk);

L = acker(A',C',Pl)' % Ackerman Formula

% SFC and FOO

Al = A-B\*K-L\*C;

Bl = L;

Cl = -K;

Alk = [A -B\*K;L\*C Al-L\*D\*K];

Clk = [C -D\*K];

Blk = [L\*D+B;B];

Dlk = D;

xtlko = [xo;xho];

P = -1/(Clk/Alk\*Blk+Dlk);

Gcllk = ss(Alk,P\*Blk,Clk,P\*Dlk);

[ycllk,t,xcllk] = lsim(Gcllk,r,t,xtlko);

for j=1:ma

x(:,j) = xcllk(:,j);

xh(:,j) = xcllk(:,j+ma);

%xh(j)=xh(j)'

end

Xh = xh';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,x(:,a),t,xh(:,a),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ycllk), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

save()

% --- Executes on button press in pushbutton10.

function pushbutton10\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton10 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% FOO & SFC

disp('Enter the initial estimations xho one by one followed by return:')

for j=1:ma

xho(j)=input('');

end

xho=xho';

Mp = input('Enter the Overshoot factor Mp:\n');

ts = input('Enter the Settling time ts:\n');

x = log(Mp)^2;

zeta = sqrt(x/(pi^2+x));

wn = 5/(zeta\*ts);

Do=[1 2\*zeta\*wn wn^2];

Pk = roots(Do); %Dominant Poles

if max(size(Pk))<ma

for q=1:ma-max(size(Pk))

Pnk(q)=5\*real(Pk(1));

end

Pk=[Pk;Pnk']

end

Pk=Pk';

% State Feedback Control

K = acker(A,B,Pk);

Ak = A-B\*K;

Ck = C-D\*K;

P = 1/(D-Ck/Ak\*B);

Bk = P\*B;

Dk = P\*D;

% Full Order Observer

Pl = 5\*real(Pk);

L = acker(A',C',Pl)' % Ackerman Formula

% SFC and FOO

Al = A-B\*K-L\*C;

Bl = L;

Cl = -K;

Alk = [A -B\*K;L\*C Al-L\*D\*K];

Clk = [C -D\*K];

Blk = [L\*D+B;B];

Dlk = D;

xtlko = [xo;xho];

P = -1/(Clk/Alk\*Blk+Dlk);

Gcllk = ss(Alk,P\*Blk,Clk,P\*Dlk);

[ycllk,t,xcllk] = lsim(Gcllk,r,t,xtlko);

Xh = (xcllk(:,1:ma))';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xcllk(:,a),t,xcllk(:,a+ma),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ycllk), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

% --- Executes on button press in pushbutton11.

function pushbutton11\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton11 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% FOO & SFC

disp('Enter the inital estimations xho one by one followed by return:')

for j=1:ma

xho(j)=input('');

end

xho=xho';

zeta = input('Enter the dapmping factor :\n');

wn = input('Enter the natural frequency wn \n');

Do=[1 2\*zeta\*wn wn^2];

Pk = roots(Do); %Dominant Poles

if max(size(Pk))<ma

Pn=5\*real(Pk(1));

Pk=[Pk;Pnk];

end

Pk=Pk';

% State Feedback Control

K = acker(A,B,Pk);

Ak = A-B\*K;

Ck = C-D\*K;

P = 1/(D-Ck/Ak\*B);

Bk = P\*B;

Dk = P\*D;

% Full Order Observer

Pl = 5\*real(Pk);

L = acker(A',C',Pl)' % Ackerman Formula

% SFC and FOO

Al = A-B\*K-L\*C;

Bl = L;

Cl = -K;

Alk = [A -B\*K;L\*C Al-L\*D\*K];

Clk = [C -D\*K];

Blk = [L\*D+B;B];

Dlk = D;

xtlko = [xo;xho];

P = -1/(Clk/Alk\*Blk+Dlk);

Gcllk = ss(Alk,P\*Blk,Clk,P\*Dlk);

[ycllk,t,xcllk] = lsim(Gcllk,r,t,xtlko);

Xh = (xcllk(:,1:ma))';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xcllk(:,a),t,xcllk(:,a+ma),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ycllk), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

% --- Executes on button press in pushbutton12.

function pushbutton12\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton12 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% ROO & SFC

disp('The first state is considered measured and the remaing ones are unmeasured')

disp('Enter the initial estimations xho one by one followed by return:')

for j=1:ma-1

xoh(j)=input('');

end

xoh=xoh';

disp('Enter the closed loop desired poles in complex "a+bi" form:')

for i=1:ma

a=input('Enter the real part "a" parameter: \n');

b=input('Enter the imaginary part "b" parameter: \n');

Pk(i)=complex(a,b);

end

Pk=Pk';

disp('Enter the desired ROO poles one by one followed by return')

for j=1:ma-1

Pd(j)=input('');

end

Pd=Pd';

% ROO

Nm = [1];

Nu = [2:ma];

m = length(Nm);

u = length(Nu);

n = m + u;

Amm = A(Nm,Nm);

Amu = A(Nm,Nu);

Aum = A(Nu,Nm);

Auu = A(Nu,Nu);

L = acker(Auu',Amu',Pd)';

xot = xo(2:ma) - xoh;

% SFC

K = acker(A,B,Pk);

Ku = K(Nu);

Km = K(Nm);

% Augmented System

Aa = [A-B\*K B\*Ku;zeros(u,n) Auu-L\*Amu];

Ba = [B;zeros(size(B(Nu)))];

Ca = [C zeros(size(B(Nu)))'];

Ga = ss(Aa,Ba,Ca,0);

xoa = [xo;xot];

% Simulation

[ya,t,xa] = lsim(Ga,r,t,xoa);

Xh = (xa(:,1:ma))';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

subplot(ma,1,1), plot(t,xa(:,1)), title('x1')

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xa(:,a),t,xa(:,a+ma-1),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ya), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

save()

% --- Executes on button press in pushbutton13.

function pushbutton13\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton13 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% ROO & SFC

disp('The first state is considered measured and the remaing ones are unmeasured')

disp('Enter the initial estimations xho one by one followed by return:')

for j=1:ma-1

xoh(j)=input('');

end

xoh=xoh';

Mp = input('Enter the Overshoot factor Mp:\n');

ts = input('Enter the Settling time ts:\n');

x = log(Mp)^2;

zeta = sqrt(x/(pi^2+x));

wn = 5/(zeta\*ts);

Do=[1 2\*zeta\*wn wn^2];

Pk = roots(Do); %Dominant Poles

if max(size(Pk))<ma

for q=1:ma-max(size(Pk))

Pnk(q)=5\*real(Pk(q));

end

Pk=[Pk;Pnk'];

end

Pk=Pk';

disp('Enter the desired ROO poles one by one followed by return:')

for j=1:ma-1

Pd(j)=input('');

end

Pd=Pd';

% ROO

Nm = [1];

Nu = [2:ma];

m = length(Nm);

u = length(Nu);

n = m + u;

Amm = A(Nm,Nm);

Amu = A(Nm,Nu);

Aum = A(Nu,Nm);

Auu = A(Nu,Nu);

L = acker(Auu',Amu',Pd)';

xot = xo(2:ma) - xoh;

% SFC

K = acker(A,B,Pk);

Ku = K(Nu);

Km = K(Nm);

% Augmented System

Aa = [A-B\*K B\*Ku;zeros(u,n) Auu-L\*Amu];

Ba = [B;zeros(size(B(Nu)))];

Ca = [C zeros(size(B(Nu)))'];

Ga = ss(Aa,Ba,Ca,0);

xoa = [xo;xot];

% Simulation

[ya,t,xa] = lsim(Ga,r,t,xoa);

Xh = (xa(:,1:ma))';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

subplot(ma,1,1), plot(t,xa(:,1)), title('x1')

for a=2:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xa(:,a),t,xa(:,a+ma-1),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ya), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

% --- Executes on button press in pushbutton14.

function pushbutton14\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton14 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load()

if CO==1&&Ob==1

ma=max(size(A));

% ROO & SFC

disp('The first state is considered measured and the remaing ones are unmeasured')

disp('Enter the initial estimations xho one by one followed by return:')

for j=1:ma-1

xoh(j)=input('');

end

xoh=xoh';

zeta = input('Enter the dapmping factor :\n');

wn = input('Enter the natural frequency wn \n');

Do=[1 2\*zeta\*wn wn^2];

Pk = roots(Do); %Dominant Poles

if max(size(Pk))<ma

for q=1:ma-max(size(Pk))

Pnk(q)=5\*real(Pk(q));

end

Pk=[Pk;Pnk'];

end

Pk=Pk';

disp('Enter the desired ROO poles one by one followed by return:')

for j=1:ma-1

Pd(j)=input('');

end

Pd=Pd';

% ROO

Nm = [1];

Nu = [2:ma];

m = length(Nm);

u = length(Nu);

n = m + u;

Amm = A(Nm,Nm);

Amu = A(Nm,Nu);

Aum = A(Nu,Nm);

Auu = A(Nu,Nu);

L = acker(Auu',Amu',Pd)';

xot = xo(2:ma) - xoh;

% SFC

K = acker(A,B,Pk);

Ku = K(Nu);

Km = K(Nm);

% Augmented System

Aa = [A-B\*K B\*Ku;zeros(u,n) Auu-L\*Amu];

Ba = [B;zeros(size(B(Nu)))];

Ca = [C zeros(size(B(Nu)))'];

Ga = ss(Aa,Ba,Ca,0);

xoa = [xo;xot];

% Simulation

[ya,t,xa] = lsim(Ga,r,t,xoa);

Xh = (xa(:,1:ma))';

% The Actuating Signal

u = -K\*Xh +r;

% Plotting

figure

subplot(ma,1,1), plot(t,xa(:,1)), title('x1')

for a=2:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,xa(:,a),t,xa(:,a+ma-1),':')

str = sprintf('%s x%d Vs x\_%d^h',stv,a,a);

title(str);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,u), str1=sprintf('The Actuating Signal for %s',sp);

title(str1)

subplot(212), plot(t,ya), str2=sprintf('The Output Response for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

% --- Executes on button press in pushbutton21.

function pushbutton21\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton21 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

% Finding System's Poles to assess stability

%data = daqread('OptimalGUI')

load()

set(handles.pushbutton5, 'Enable', 'on');

set(handles.pushbutton6, 'Enable', 'on');

set(handles.pushbutton7, 'Enable', 'on');

set(handles.pushbutton8, 'Enable', 'on');

set(handles.pushbutton9, 'Enable', 'on');

set(handles.pushbutton10, 'Enable', 'on');

set(handles.pushbutton11, 'Enable', 'on');

set(handles.pushbutton12, 'Enable', 'on');

set(handles.pushbutton13, 'Enable', 'on');

set(handles.pushbutton14, 'Enable', 'on');

set(handles.pushbutton15, 'Enable', 'on');

set(handles.pushbutton22, 'Enable', 'on');

P = eig(A);

disp('The Poles Of The System Are: ')

P

% Stability Test

for q=1:length(P)

s=0;

if real(P(q)) > 0

s=s+1;

fprintf('Mode %d (s = %d) is Unstable\n', q, P(q))

else

fprintf('Mode %d (s = %d) is Stable\n', q , P(q))

end

end

if s==0

fprintf('The System is Stable!\n\n')

St=1;

else fprintf('The System is Unstable!\n\n')

St=0;

end

ma=max(size(A));

% Contrability Test

e=0;

[lV,lD] = eig(A');

[rV,rD] = eig(A);

for i=1:max(size(lV))

test=(lV(:,i))'\*B;

if test==0

fprintf('Mode %d is Uncontrollable\n', i)

if real(P(i)) > 0

fprintf('Mode %d is Not Stabilizable\n',i)

end

else

fprintf('Mode %d is Controllable\n', i)

if real(P(i)) > 0

fprintf('Mode %d is Stabilizable\n',i)

e=e+1;

end

end

end

if e==max(size(rV))

fprintf('The system is Stabilizable\n')

else

fprintf('The system is Not Stabilizable\n')

end

Co=ctrb(A,B);

if rank(Co)~= ma

fprintf('The System is Uncontrollable!\n\n')

CO=0;

else fprintf('The System is Controllable!\n\n')

CO=1;

end

% Observability Test

d=0;

for i=1:max(size(rV))

test=C\*rV(:,i);

if test==0

fprintf('Mode %d is Unobservable\n', i)

if real(P(i)) > 0

fprintf('Mode %d is Not Detectable\n',i)

end

else

fprintf('Mode %d is Observable\n', i)

if real(P(i)) > 0

fprintf('Mode %d is Detectable\n',i)

d=d+1;

end

end

end

if d==max(size(rV))

fprintf('The system is Detectable\n')

else

fprintf('The system is Not Detectable\n')

end

O = obsv(A,C);

if rank(O) ~= ma

fprintf('The System is Unobservable!\n\n')

Ob=0;

else fprintf('The System is Observable!\n\n')

Ob=1;

end

%Redesign and bad system

if St==1&&CO==0&&Ob==0

fprintf('WARNING: CANNOT DESIGN A CONTROLLER FOR YOUR SYSTEM!!\n')

return;

end

if St==0&&CO==0

fprintf('WARNING: YOUR SYSTEM MUST BE REDESIGNED!!\n')

return;

end

save();

% --- Executes on button press in pushbutton22.

function pushbutton22\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton22 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load();

G=ss(A,B,C,D);

tf(G)

if C(1)==1

sp = sprintf('Cart''s Position');

else

sp = sprintf('Pendulum''s Angular Position');

end

% pole and zero plot

figure

pzplot(G),grid on

%initial iputs

ft = input('\nEnter the simulation final time: \n');

r = input('\nEnter the type of input "1" for "0 input" and "2" for "Unit Step function": \n');

t=0:0.1:ft;

if r==1

r=0\*t;

else if r==2

r=stepfun(t,0);

end

end

disp('Enter Initial Conditions Xo one by one followed by return: ')

for i=1:max(size(A));

xo(i)=input('');

end

xo=xo';

save();

% --- Executes on button press in pushbutton15.

function pushbutton15\_Callback(hObject, eventdata, handles)

% hObject handle to pushbutton15 (see GCBO)

% eventdata reserved - to be defined in a future version of MATLAB

% handles structure with handles and user data (see GUIDATA)

load();

if CO==1&&Ob==1

ma=max(size(A));

% Optimal SFC/FOO Design

[y,t,x] = lsim(G,r,t,xo);

% Optimal SFC

Q = C'\*C;

R=1;

[K,P] = lqr(A,B,Q,R);

% Optimal FOO

[L,PL] = lqr(A',C',Q,R);

L=L';

PL=PL';

Akl = [A -B\*K;L\*C A-B\*K-L\*C];

Gkl = ss(Akl, zeros(max(size(Akl)),1), [C zeros(1,ma)],[])

[ykl,tkl,xkl] = lsim(Gkl,r,t,[xo;-xo]);

u= -K\*xkl(:,ma+1:2\*ma)';

% Plotting

figure

for a=1:ma

if a==1

stv=('Cart''s Position');

else if a==2

stv=('Cart''s Velocity');

else

if a==3

stv=('Pendulum''s Angular Position');

else if a==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,a), plot(t,x(:,a)),

str1 = sprintf('%s x%d Uncompensated',stv,a);

title(str1);

ylabel ('Amplitude')

xlabel('Time')

end

figure

for b=1:ma

if b==1

stv=('Cart''s Position');

else if b==2

stv=('Cart''s Velocity');

else

if b==3

stv=('Pendulum''s Angular Position');

else if b==4

stv=('Pendulum''s Angular Velocity');

end

end

end

end

subplot(ma,1,b), plot(t,xkl(:,b),t,xkl(:,b+ma),':')

str2 = sprintf('%s Compensated x%d Vs x\_%d^h',stv,b,b);

title(str2);

ylabel ('Amplitude')

xlabel('Time')

end

figure

subplot(211), plot(t,ykl), str1=sprintf('Compensated System output y for %s',sp);

title(str1)

subplot(212), plot(t,u), str2=sprintf('Optimal Control effort u^\* for %s',sp);

title(str2)

else

fprintf('WARNING: YOUR SYSTEM CANNOT BE ESTIMATED AND ITS PERFORMANCE CANNOT BE CHANGED\n')

return;

end

save()

# Table of figures

[Figure 1: Block Diagram of a nth order system 9](#_Toc346467500)

[Figure 2: closed loop system block diagram 15](#_Toc346467501)

[Figure 3: FOO Block Diagram 17](#_Toc346467502)

[Figure 4: FOO & SFC Block Diagram 19](#_Toc346467503)

[Figure 5: ROO Block Diagram 20](#_Toc346467504)

[Figure 6: ROO and SFC Block Diagram 20](#_Toc346467505)

[Figure 7: GUI figure based display 21](#_Toc346467506)

[Figure 8: Pole zero plot of the system 23](#_Toc346467507)

[Figure 9: SFC – DPP for the pendulum position analysis 26](#_Toc346467508)

[Figure 10: SFC – TDDC for the pendulum position analysis 28](#_Toc346467509)

[Figure 11: poles/zeros plot 31](#_Toc346467510)

[Figure 12: SFC – DPP for the cart position 34](#_Toc346467511)

[Figure 13: SFC –TDDC for the cart position 36](#_Toc346467512)

[Figure 14: FOO design 38](#_Toc346467513)

[Figure 15: FOO & SFC with DPP 40](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467514)

[Figure 16: Actuating signal and output response for the designed FOO & SFC (DPP) 41](#_Toc346467515)

[Figure 17: FOO & SFC with TDDC 42](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467516)

[Figure 18: Actuating signal and output response for the designed FOO & SFC (TDDC) 43](#_Toc346467517)

[Figure 19: ROO & SFC (DPP) 45](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467518)

[Figure 20: Actuating signal and output response for the designed ROO & SFC (DPP) 46](#_Toc346467519)

[Figure 21: ROO & SFC using TDDC 48](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467520)

[Figure 22: Actuating signal and output response for the designed ROO & SFC (TDDC) 49](#_Toc346467521)

[Figure 23: the uncompensated state variables for Optimal SFC/FOO Design 50](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467522)

[Figure 24: the compensated state variables for Optimal SFC/FOO Design 51](file:///C:\Users\kif\Desktop\Optimal%20Control\project%20segway.docx#_Toc346467523)

[Figure 25: the compensated system output & the optimal effort 52](#_Toc346467524)

# References:

* Robert L. Williams II and Douglas A. Lawrence, *Linear State-Space Control Systems,* John Wiley, 2007
* William L. Brogan, *Modern Control Theory*, Prentice Hall, Third edition
* Frank L.Lewis, Darren M.Dawson, Chaouki T.Abdallah, *Robot Manipulator Control Theory and Practice*, Marcel Dekker Inc, Second edition.
* <http://www.avantcar.si/en/rentacar/segway-rental>
* Sethi, S. P., and Thompson, G. L., 2000. *Optimal Control Theory: Applications to Management Science and Economics*, 2nd edition, Springer
* <http://www.utdallas.edu/~sethi/OPRE7320presentation.html>