ECE 414 - Take Home Test

Mohammed Al-Sayegh

April 22, 2019

In this report five technical methods of design for controllers for G(s) plant are investigated. The plant itself is given to us in a form of randomly generated outputs from the function *ece414planttf.p* set by the date of our birthday, in this case the 28th of December. There will be 100 plant G(s) where the nominal plant must be located to design each controller. Robust system that can manage to control the 100 plant of G(s).

1. Root Locus

The first technique is root locus. By investigation the option of the PID type controller we decide which best fit controller is to be designed. Because of the number zero's in the plant only types PID and PIDF are feasible to be designed. A PI controller was chosen as candidate based on what was given as nominal plants.

2. pidtune and pidtuner

In this part, only the candidates PI and PIDF were considered for design on the bases of the possibility of PID that can achieved goal specification. A pidTuner use to generate the PID type plant. While pidTuner is used to open GUI interface and control the parameters of the PI and PIDF controller.

3. pidtune with ***pidsearch.m***

In this section, *pidsearch.m* is used to tone a *pidTune* control plant. It helped to optimize the baseline of the PID controller to control. The overshoot chosen to be the best to be control.

4. Unity Feedback Linear Algebra design

In this part of the report, a ***stepitae*** and ***stepshape*** were used to generate a D(s) plant to control the nominal G(s) plant. With the help of the ***lamdesign*** function, the result of the generated controller plan was generated and check if it met the unity feedback deign limitation.

5. Two Parameter Linear Algebraic Controller Design

For two parameters LAM, ***lamdesign*** function was used. A plant generated using ***steplqr,*** the real roots of the plant used as vector to pass ***lamdesign*** to generate H(s) and F(s) plants.

1. Introduction

This report has two main objectives. The first objective is to design five proper controllers with each controller using different design techniques. The first controller uses the root locus technique. The second controller is a PID type controller that uses either the *pidtune* or *pidTuner* functions in MATLAB. The third controller is a PID type controller designed using the *pidsearch* function. The fourth controller uses the unity feedback linear algebra design technique. Lastly, the fifth controller uses the two-parameter linear algebraic design technique. The second objective is to choose the best fit proper controller that meets all listed design goals in Table 1.

Table 1: Proper controllers design goals

Parameter	**Design Goals**
Unit step input	Zero percent steady state error
Overshoot	Less than ten percent
Step response	may not go in the wrong direction first
Phase margin	greater than forty-five degrees
Settling time	Minimize
Peak control effort	Minimize
Peak sensitivity	Minimize

The design approach for each of the five proper controllers is described below. In each section of the proper controllers, the criteria that met and failed the design goals are described. The results of the best fit controller are shown in the design evaluation section for proper controllers that met the design goals.

2. Finding the Nominal Plant

In this section, the process of searching for the nominal plant is described. The nominal plant is the plant that can be used as a reference to describe the system characteristics. A provided function called *ece414planttf.p* was used to base the system controller design on. The function returns 100 random variations of the plant G(s) by varying the alpha value for a given day and month. The date passed to the function is December 28th, and the alpha range passed was 1:100.
In this design, the 28th used for the day and 12 is set as the month. The alpha is varied from 1 to 100. The nominal plant gain is the average of the gain of the 100 plants variation. The location of the nominal plant can be seen in Figure 1.

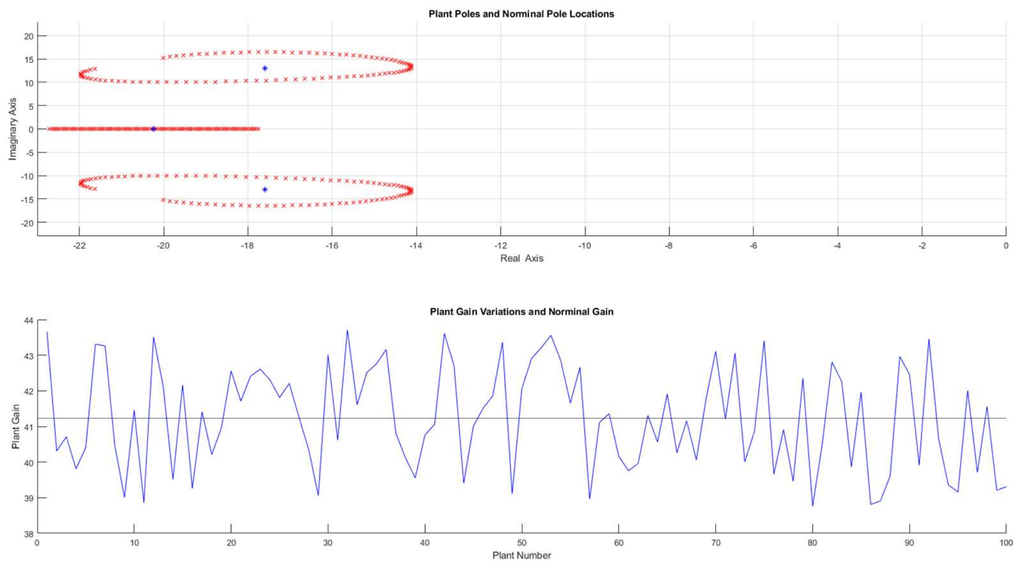


Figure 1: Nominal Plant location

The result of the gain is found by taking the average of all gain plants representing the gain of the nominal plant which can be seen in Figure 1. Using the found average zero's location of the plant and the average gain of all possible random variation, the nominal plant is found to be as the following:

$$G(s) = \frac{41.234}{(s+20.23)\,(s^2 + 35.18s + 477.1)} - (1)$$

Equation (1) references plant G(s) that can be used to build a robust control system.

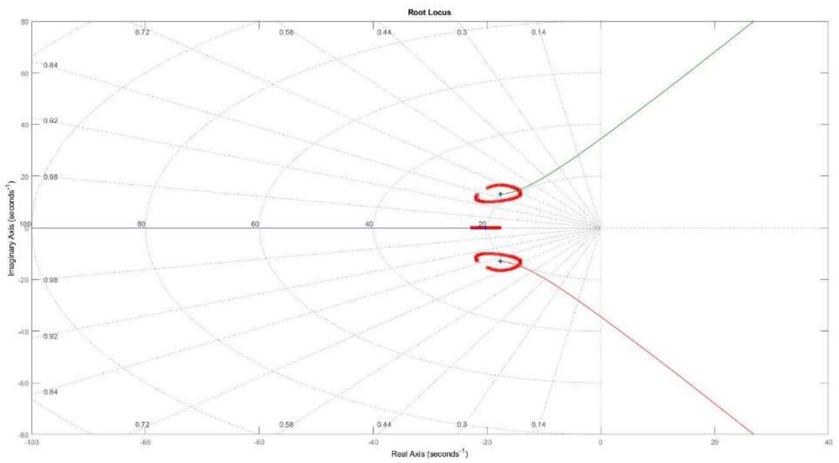


Figure 1: Root locus of the nominal plant centered in the middle of for a normal distribution of Alpha

3. PID Controller design with Root Locus Technique

In this section, we are designing a PID type (e.g., P, PD, PI, or PID) controller using the Root locus technique. There are multiple variations of PID type. However, there is only one case out of the listed types that is possible to implement. A 'P' type, also known as a proportional control, is completely governed by the G(s) plane. It is not possible to achieve the design goal of a zero steady state error when analyzing other PID types such as 'D' base type base (PD and PID). It is important to see the 'D' term effect on the controller. When the number of zeros is greater than the number of poles, it implies that the effect is measured before the cause occurred, this is the case for an ideal PD or PID for the 'D' part of the controller. So, it is not physically realizable to achieve a zero steady state error unless a derivative filter is introduced in its denominator, thus making the number of zeros equal to number of poles. Such systems are called PD + Filter (PDF) and PID + Filter (PIDF). The same goes for PDF as it goes for PD and PID. The 'D' term will have high influence on the controller. So that leave us with PI and PIDF controllers.

The control plant of the PIDF is expressed as follows:

$$D(s) = K_D \frac{(S+Z_1)(S+Z_2)}{S(S+P)} \text{ -- } (3)$$

Where the terms to control PIDF will be K_D, Z_1, P, and Z_2. By investigating the PIDF controller plant, it can be seen it requires four terms to control the nominal plant. It will be a difficult task to achieve a balanced controller relying only on root locus to change K_D, Z_1, P, and Z_2. This leaves the last PID type left as an option which is 'PI'. A PI controller can be described as follows:

$$D(s) = K_P + \frac{K_I}{S} = \frac{(K_P \times S) + K_I}{S} = \frac{K_P\left(S + \frac{K_I}{K_P}\right)}{S}$$

$$\therefore D(s) = K_P \times \frac{S+Z}{S} \text{ -- } (4)$$

3.1. Method

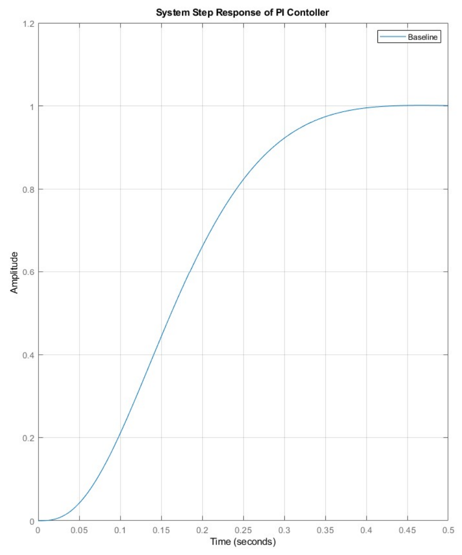
The PI controllers can be altered. It has two terms unlike the PIDF, so by choosing the Z term and varying K_P, the PI controller can be calibrated to meet the design goals. The two terms must be manipulated in order to achieve the system design goal.

3.2. Implementations Result

The values of K_P and Z were altered incrementally until a balance was achieved. The values that were found met all specifications of the design goals at K_P equals 78.7 and Ki equals 1360. The root locus expression found was:

$$D(s) = K_P + \frac{K_I}{s} = 78.7 + \frac{1360}{s} - (5)$$

Figure 2 shows the step response and control effort of the PI controller.



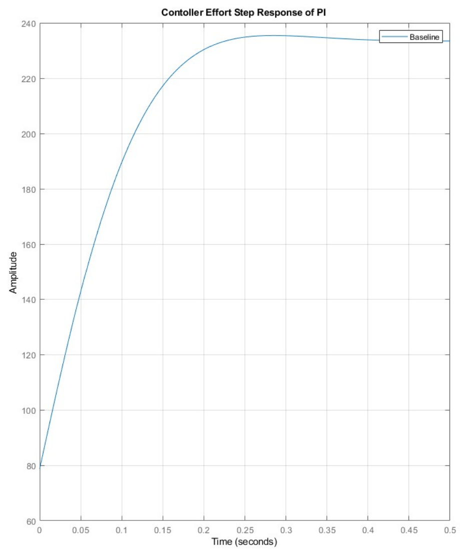


Figure 2: Step response and Control effort of the PI controller.

In the Table 2, a list of controller information.

Table 2: Systems info for PI

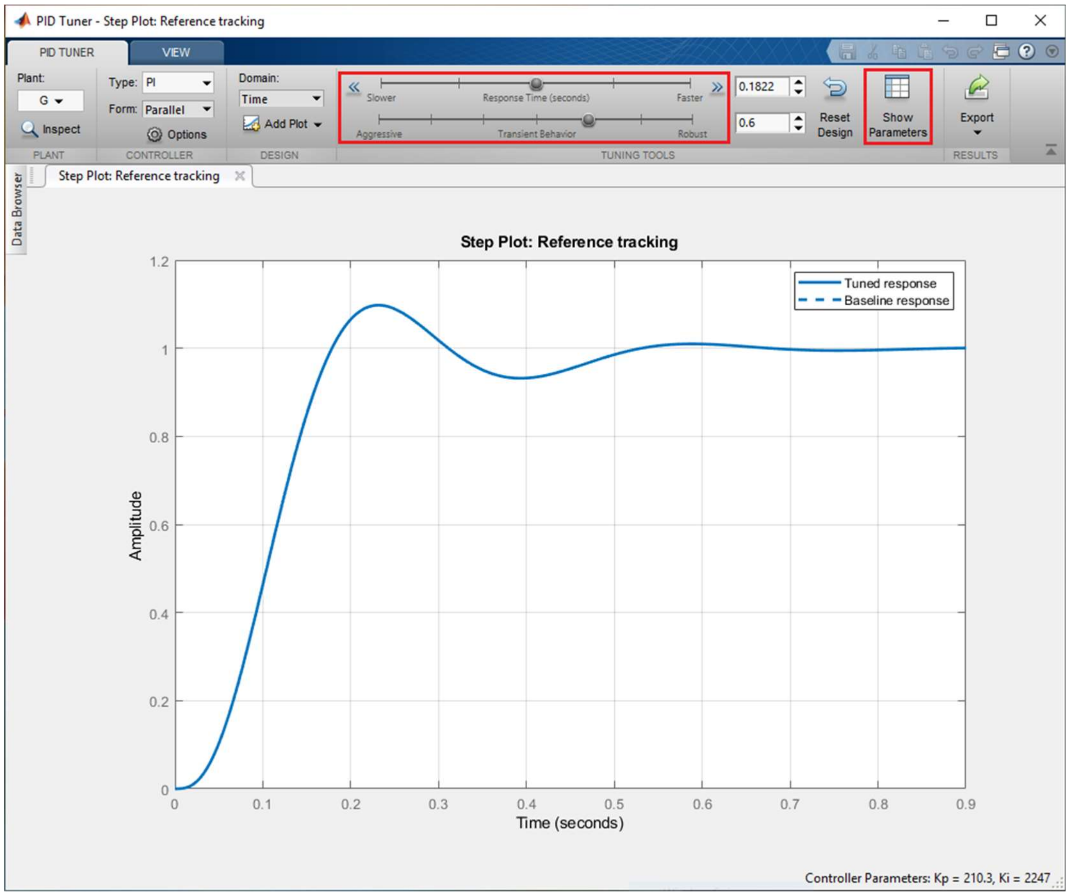
Info type	**PI Controller**
Rise Time	0.2148
Settling Time	0.3602
Settling Min	0.9016
Settling Max	1.0011
Overshoot	0.1141
Undershoot	0
Peak	1.0011
Peak Time	0.4661
Umax	235.4305
Ess_{Step}	0
Ess_{Ramp}	0.1720
Gain margin in dB	15.9155
Phase margin in degrees	68.0027
Gain crossover frequency in rad/s	23.2454
Phase crossover frequency in rad/s	5.7605
Vector margin	0.7271
Vector margin frequency	14.8168
Peak sensitivity	1.3754

4. PID Controller design with *pidTune* and *pidTuner*

In this section, *pidTune* and *pidTuner* GUI interface was used to calibrate the PID type controller parameter manually. Only the PI and PIDF type controllers were considered as proper candidates. The reasoning of only considering the aforementioned controllers has already been established in the root locus technique section.

4.1. Method

MATLAB *pidTune* is used to create a PID model of PI and PIDF type. The controller terms of Kp, Ki, and Kd were generated through the processes of creating a PID model. Terms have a baseline that are given by *pidTune*. By using *getallspecs.m* and 'Show Parameters' from the *pidTuner* window as shown in Figure 3, the parameter of the closed loop of the controller can be derived. To achieve better results, the two systems were altered by using 'Response Time' and 'Transient Behavior' from the design window of *pidTuner*.


PID Tuner - Step Plot: Reference tracking

Plant: G
Type: PI
Form: Parallel
Domain: Time
Response Time (seconds): 0.1822
Transient Behavior: 0.6

Controller Parameters: Kp = 210.3, Ki = 2247

Figure 3. ***pidTuner*** **window**

4.2. Implementations Result

After creating model for PI and PIDF, the parameter of the controller will be decided base on the *pidTuner*. The result of the alteration of the system is listed in this subsection. In Table 3, the terms K_P, K_D and K_I for the tuned controller are shown.

Table 3: Systems info for PI and PIDF

PID Type Terms	PI Controller	PIDF Controller
K_P	110.2561	470.4864
K_I	1.5274e+03	3.9427e+03
K_D	-	13.9307

While in Table 4, the parameters associated with the step response of PI and PIDF controllers using *getallspecs.m* are shown.

Table 4: Systems info for PI and PIDF

Info type	**PI Controller**	**PIDF Controller**
Rise Time	0.1820	0.0721
Settling Time	0.3115	0.3158
Settling Min	0.9022	0.9247
Settling Max	0.9996	1.0483
Overshoot	0	4.8280
Undershoot	0	0
Peak	0.9996	1.0483
Peak Time	0.8720	0.1398
Umax	237.0692	3.1841e+04
Ess_{Step}	0	0
Ess_{Ramp}	0.1533	0.0594
Gain margin in dB	14.9450	38.6952
Phase margin in degrees	69.0015	62.0023
Gain crossover frequency in rad/s	25.1524	223.2869
Phase crossover frequency in rad/s	6.6663	19.7035
Vector margin	0.7052	0.6990
Vector margin frequency	16.3174	32.6687
Peak sensitivity	1.4181	1.4306

After evaluating the step response of the two-systems using MATLAB. A plot of the step response of the PI and PIDF system shown in Figure 3. The control plant of the PI controller can be expressed as follows:

$$D_{PI}(s) = K_P + K_i \times \frac{1}{S} = 110 + \frac{1.53e3}{S} \quad -(5)$$

While the control plant expression for PIDF is as follows:

$$D_{PIDF}(s) = K_P + K_i \times \frac{1}{S} + K_i \times \frac{S}{T_f S + 1} = 470 + \frac{3.94e3}{S} + \frac{13.9 \times S}{0.000444 \times S + 1} - (6)$$

The evaluation of the step responses of the PI and PIDF systems using MATLAB are shown in Figure 4.

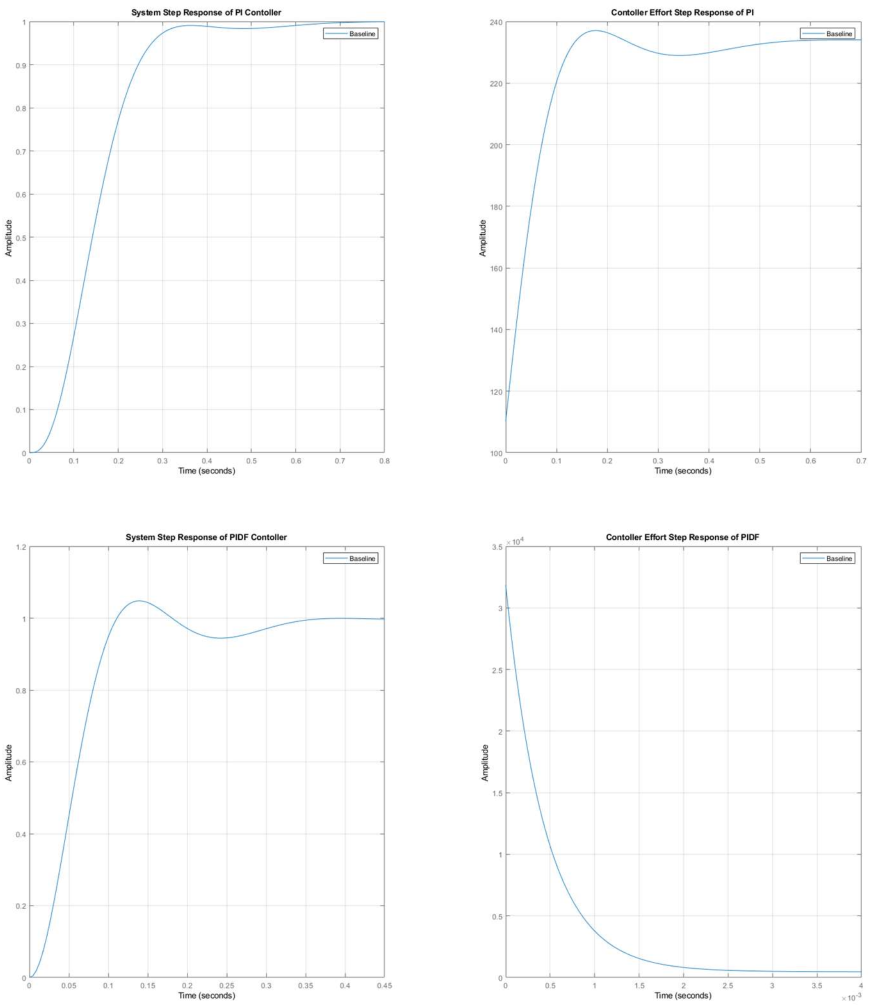


Figure 4: The step response and controller effort for the PI and PIDF type contoller using *pidTuner*

By investigating the listed results from Table 3, both the PI and PIDF met the design goal. There are three design goals, each where one controller is preferable to the other. If we look at settling time, the PIDF had the least amount of the settling timing. While observing the peak control effort and sensitivity, the PI controller had the lowest magnitude. This meets more design goals than the PIDF controller. In this design technique, the PI controller will be a better candidate as the best control option.

5. PID Controller Design with *pidsearch.m*

In this section, *pidsearch.m* is used to tone a *pidTune* control plant. It helped to optimize the baseline of the PID controller to control the following criteria such as:

- 'ITAE' is the integral of time and the absolute error.
- 'ISTE' is the integral of the square of $t \times |y_{final} - y|$.
- 'ISE' is the integral of the squared error.
- 'RISE' is the integral of the squared error starting at the rise time Tr.
- 'OS' is the step response overshoot.
- 'Ts' is the step response settling time.
- 'OSTs' is the product of the overshoot and the settling time.
- 'UTs' is the product of the initial control effort U and the settling time.

By looking at the listed available options for tuning, only one criterion is limited numerically for the design goals which is the overshoot, other criteria are preferred to be as low as possible. Therefore, the *pidshearch.m* is set to be based on the step response overshoot.

5.1. Method

A function called *specs_table.m* uses *pidTuner.m* and *pidsearch.m* to create an excel file that includes all controller information in a table. It receives the plant G(s) of which the design decision is based on the output result of the look up table. The system which had a control effort that is not infinity and had zero steady state error is then printed. There were only two systems which match these two conditions, the PI and PIDF controllers.

5.2. Implementations Result

Table 5: Systems info for PI and PIDF

Info type	**PI Controller**	**PIDF Controller**
Rise Time	0.1088	0.0710
Settling Time	0.4908	0.3156
Settling Min	0.9120	0.9231
Settling Max	1.0976	1.0672
Overshoot	9.7624	6.7243
Undershoot	0	0
Peak	1.0976	1.0672
Peak Time	0.2309	0.1403
Umax	310.6495	3.0872e+04
Ess_{Step}	0	0
Ess_{Ramp}	-0.1042	-0.0571
Gain margin in dB	11.1148	38.4646
Phase margin in degrees	60	60.0003
Gain crossover frequency in rad/s	27.1423	217.0343
Phase crossover frequency in rad/s	10.9777	19.7044
Vector margin	0.5840	0.6884
Vector margin frequency	18.7948	31.8196
Peak sensitivity	1.7124	1.4527

The two evaluated proper systems matched all design goals listed in Table 1. The control plant expressions for PI and PIDF after tuning with the *pidsearch.m* function are shown in Equations 7 and 8.

$$D_{PI}(s) = \frac{237.86 \times (S + 6.68)}{S} \quad - (7)$$

$$D_{PIDF}(s) = \frac{29666 \times (S + 16.87)(S + 15.87)}{s(s + 2474)} \quad - (8)$$

The closed loop step response and control effort of the D_{PI} and D_{PIDF} can be seen for nominal plant in Figure 5.

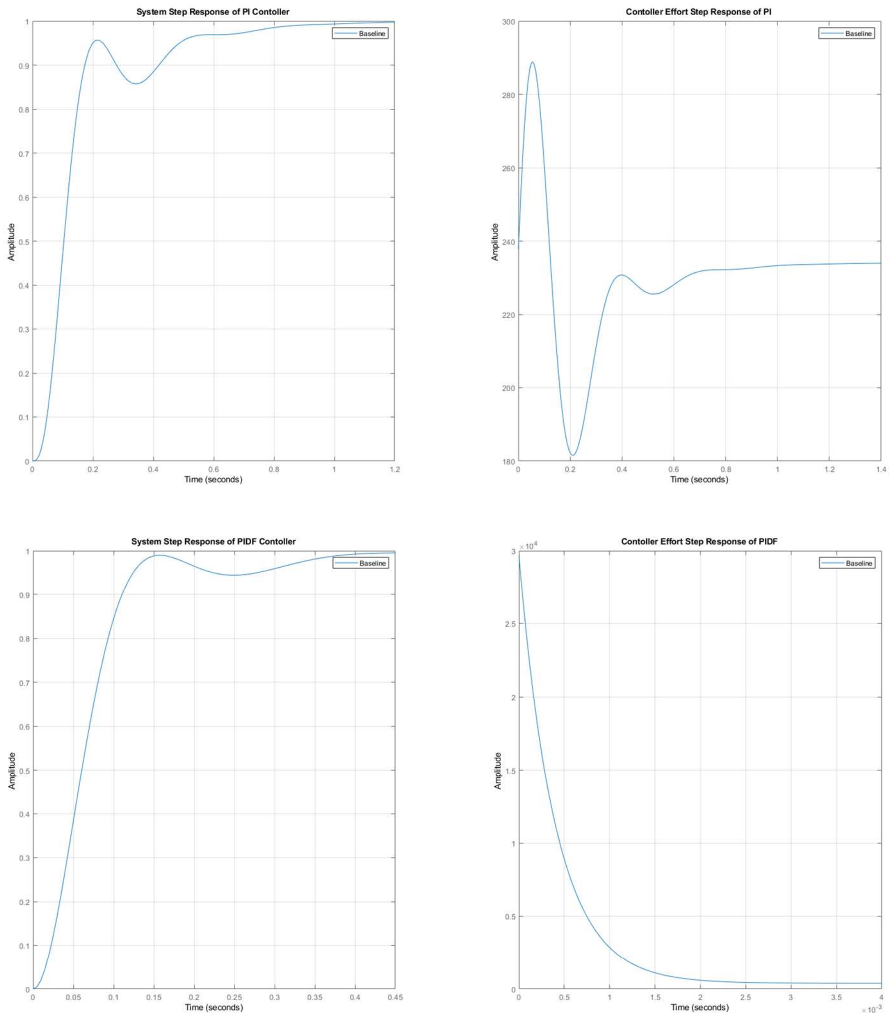


Figure 5: The step response and controller effort for PI and PIDF type Contoller using *pidsearch*

By investigating the collecting result in Table 3 and by looking at both system responses in Figure 3, we can see similar results from the *pidTuner* technique. However, this time PIDF controller achieves more goals than PI controller. So, let's see what is different here, a PI controller has less control effort in general in both cases. However, if we look at bottom of Table 5, the sensitivity of PI increases to be greater than the PIDF, while the PIDF controller still maintains a relatively consistent settling time and has a decrease in rise time.

6. Controller with a Unity Feedback Linear Algebra design

Here we are introduced to a different approach to design a controller called the outward approach. With this kind of approach, a unity feedback configuration used for simplicity of the design.

6.1. Method

In Figure 6, a block diagram of unity feedback LAM design shown.

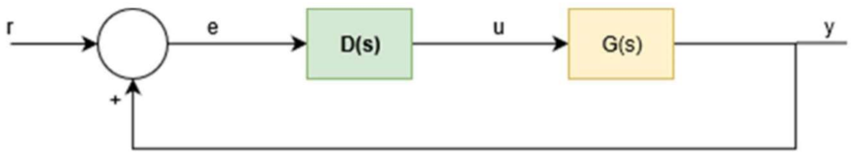


Figure 6: Block diagram of closed-loop of control for Unity Feedback

The *lamdesign.m* function used to generate a unity LAM system. For design process both *stepitae* and *stepshape.m* used to generate D_0(s). In this case, for any set of specifications there will be an ultimate choice for a LAM. However, we are still limited in terms of the controller sensitivity to plant G(S) where the desired outcome is to achieve the minimum amount of sensitivity. We are using *stepitae* and *stepshape.m* to generate desired pole for given step response specification. As design procedure, we are fallowing C.T. Chen procedure [1].

a) $\deg D_0(s) - \deg N_0 \geq \deg D(s) - \deg N$ were ploe − zero excess inquality
b) Retainment of nonmini closemum-phase zeros, that can be achieved by all closed right-hand plane (RHP) zero's of N(s) are retained in $N_0(s)$
c) $D_0(s)$ is Hurwitz (stability of $G_0(s)$and $T(s)$ requires both $D_0(s)$ and $D_1(s)$ to be Hurwitz)

All design procedure restrictions will be handled by the LAM design tool *stepitae.m, stepshape.m and lamdesign.m* by Professor Hanselman [2]. Where it will indicate when the control design is off of the parameters.

6.2. Implementations Result

By passsing the plant G(s) to the *specs_table.m* function which is a written function that calls *stepitae.m* and *stepshape.m* to create a table in excel for possible configuration using both methods. For *stepitae,* it returns a 6th order system having a overshoot value of zero and a 0.55 second settling time to within 10% tolerance. The same is done with *stepshape* and only with *stepshape.m* were design goals met. Only one of the unity systems, using *stepshape,* meets the design goal as seen in Table 4.

Table 4: Systems info of a Unity Feedback Linear Algebra design using *stepitae* and *stepshape.m*

Info type	**Unity LAM with *stepitae.m***	**Unity LAM with *stepshape.m***
Rise Time	0.2890	0.2136
Settling Time	0.6919	0.3836
Settling Min	-164.4212	0.9043
Settling Max	-144.6159	0.9995
Overshoot	3.0160	0
Undershoot	0	0
Peak	164.4212	0.9995
Peak Time	0.6163	0.5962
Umax	3.9019e+04	234.1833
Ess $_{\text{Step}}$	160.6074	0
Ess $_{\text{Ramp}}$	Inf	-0.1705
Gain margin in dB	0.0543	14.9558
Phase margin in degrees	-0.4077	68.7622
Gain crossover frequency in rad/s	0	23.2454
Phase crossover frequency in rad/s	5.0740	5.8026
Vector margin	0.0062	0.7292
Vector margin frequency	0	15.5337
Peak sensitivity	160.6074	1.3713

For a unity LAM with *stepitae.m*, the peak sensitivity and control effort is higher than the other candidate that is generated using the *stepshape.m* controller. It does not meet the design goal of zero percent steady state error. By using *stepshape.m* where the overshoot can be controlled, it results with a better control design.

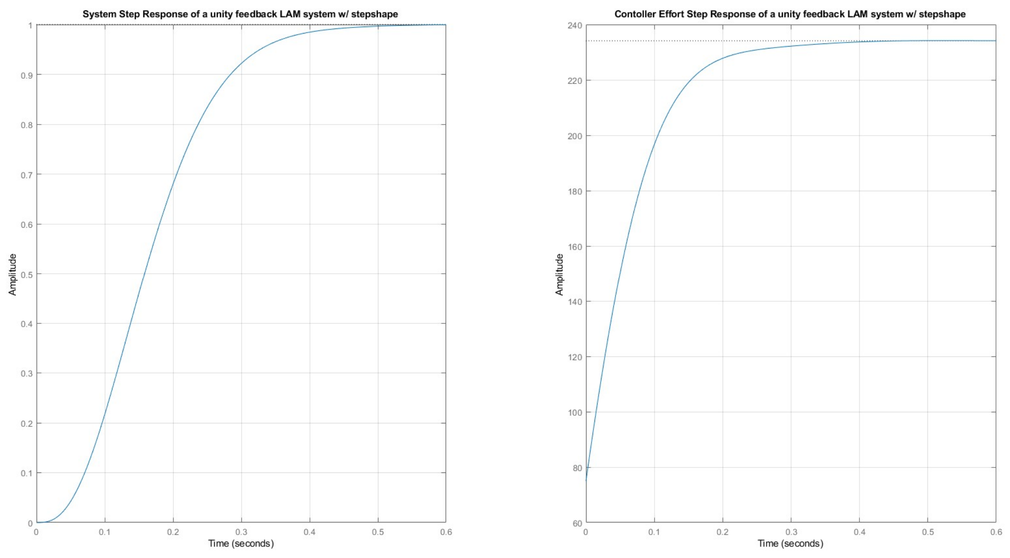


Figure 7: The step response and controller effort for Unity LAM with stepshape.m

7. Two Parameter Linear Algebraic Design Controller

In this section, the two-parameter configuration as the fifth technique are introduced. Same as unity feedback, it is an outward design approach. Although the unity feedback configuration can be used to achieved arbitrary pole placement, it generally cannot be used to achieve model matching.

7.1. Method

There are several configurations of a two-degree freedom system. They are defined by the numerator and denominator of the T(S). *steplqr.m* used to return the LQR optimum prototype system T(s). While *lamdesign.m* offers the following design configuration as shown in Figure 8 where F(S) is the forward controller, prefilter, or input signal shaper, and H(S) is the feedback controller.

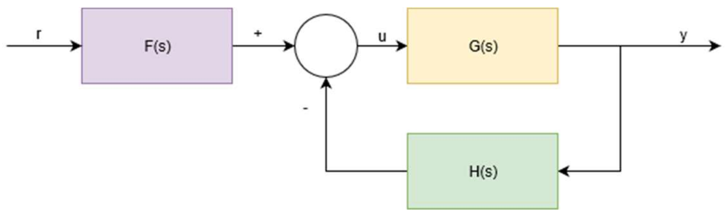


Figure 8: Block diagram of closed-loop of control for Unity Feedback

7.2. Implementations Result

lamdesign and *steplqr* are used to design a two parameters controller. The tool is the same from the unity feedback LAM controller. However, another function called *steplqr.m* is used to help design the controller. *Steplqr.m* is a function that receives the parameters of the desired settling time in seconds, the settling time percentage, and plant G(s). It returns the closed loop transfer function in the form of zpk. Through testing the set up for settling time was found to be best at 4 seconds. The expression of the T(S) is:

$$T(s) = \frac{9727.9}{(S + 20.21)(S + 1.01)(S^2 + 35.17\,S + 476.7)}$$

lamdesign.m is expecting a vector of polynomial root locations that are chosen from the design process on a needed basis. So, a real negative vector used. The result is shown in Table 5.

Table 5: Systems info for two parameter LAM controller

Info type	**Two Parameter LAM with *lamdesign.m***
Rise Time	2.1786
Settling Time	3.9999
Settling Min	0.9000
Settling Max	0.9985
Overshoot	0
Undershoot	0
Peak	0.9985
Peak Time	6.5568
Umax	235.9200
Ess $_{\text{Step}}$	0
Ess $_{\text{Ramp}}$	-1.1137
Gain margin in dB (Gm)	40.8000
Phase margin in degrees (Pm)	88.5298
Gain crossover frequency in rad/s	68.7842
Phase crossover frequency in rad/s	0.9831
Vector margin	0.9782
Vector margin frequency	20.2135
Peak sensitivity	1.0223

The controller met all design goals. The close loop step response and control effort of the two parameter LAM can be seen for the nominal plant in Figure 9.

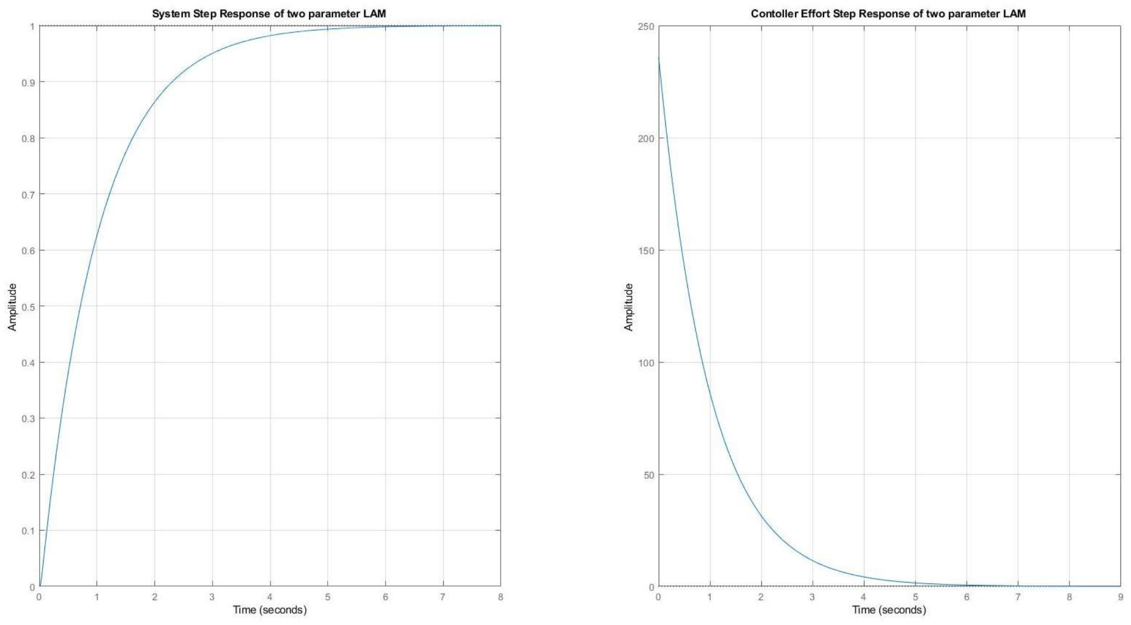


Figure 9: The step response and controller effort for Two Parameter Linear Algebraic Controller

8. Conclusion

The first objective was to design five proper controllers with each controller using different design techniques (root-locus, *pidTune*/*pidTuner*, *pidsearch.m,* unity LAM, two-parameter LAM). While the second objective was meeting all listed design goals in Table 1 in the introduction section. The results of the five controllers that achieved the two main objectives are shown in Table 6.

Table 6: Systems info for five type controller

Info type	PI Controller With root-locus	PI Controller With *pidTune/pidTuner*	PIDF Controller With *pidsearch.m*	Unity LAM with *stepshape.m*	Two Parameter LAM with *lamdesign.m*
Rise Time	0.2148	0.1820	0.0710	0.2136	2.1786
Settling Time	0.3602	0.3115	0.3156	0.3836	3.9999
Settling Min	0.9016	0.9022	0.9231	0.9043	0.9000
Settling Max	1.0011	0.9996	1.0672	0.9995	0.9985
Overshoot	0.1141	0	6.7243	0	0
Undershoot	0	0	0	0	0
Peak	1.0011	0.9996	1.0672	0.9995	0.9985
Peak Time	0.4661	0.8720	0.1403	0.5962	6.5568
Umax	235.4305	237.0692	3.0872e+04	234.1833	235.9200
Ess $_{Step}$	0	0	0	0	0
Ess $_{Ramp}$	0.1720	0.1533	-0.0571	-0.1705	-1.1137
Gain margin in dB	15.9155	14.9450	38.4646	14.9558	40.8000
Phase margin in degrees	68.0027	69.0015	60.0003	68.7622	88.5298
Gain crossover frequency in rad/s	23.2454	25.1524	217.0343	23.2454	68.7842
Phase crossover frequency in rad/s	5.7605	6.6663	19.7044	5.8026	0.9831
Vector margin	0.7271	0.7052	0.6884	0.7292	0.9782
Vector margin frequency	14.8168	16.3174	31.8196	15.5337	20.2135
Peak sensitivity	1.3754	1.4181	1.4527	1.3713	1.0223

By looking at the five controllers that listed in Table 6 it can be seen there are advantages and disadvantages for each controller. The performance of a controller is based on the rise time, settling time, overshoot and steady state error. As all listed controllers have zero steady state error, other design goals such as settling time, peak control effort and peak sensitivity will be the base for an engineering judgment. This means the PIDF with *pidsearch.m* and the two-parameter LAM controllers will be ruled out due the high control effort for PIDF controller and high settling time for two-parameter LAM controller. That leave us with PI with root-locus, PI *pidTune*/*pidTuner* and unity LAM base controllers.

By looking at the PI *pidTune*/*pidTuner* based controller and unity LAM based controller it can be seen both have zero overshoot which makes them both preferable to the PI with a root-locus base design. However, the PI *pidTune*/*pidTuner* is the best choice because of the lower settling time.

9. Reference

- Reference: C.T. Chen, Analog and Digital Control System Deisgn: Transfer Function, State Space, and Algebraic Methods, Saunders College Publishing, ISBN: 0-03-094070-2, 1993.
- D.C. Hanselman, University of Maine, Orono, ME 04469. Mastering MATLAB 7, for getallspecs.m, pidsearch.m, stepshape.m, stepitae.m and lamdesign.m

Appendix MATLAB CODE

The written code can be found in the following github repository:

https://github.com/mohammedalsayegh/ECE414