Date received:

SCHOOL OF ENGINEERING

COVER SHEET FOR CONTINUOUSLY ASSESSED WORK

Course Code ……………………EE501T………………

SECTION 1: Student to complete

SURNAME/FAMILY NAME: …………Leputa………………………………………..

FIRST NAME: …Mateusz…………………………..

ID Number: …………51337195…………………….

Date submitted: ………19/11/2018…………………

Please:

- *Read the statement on "Cheating" and definition of "Plagiarism" contained over page. The full Code of Practice on Student Discipline, Appendix 5.15 of the Academic Quality Handbook is at:* https://www.abdn.ac.uk/infohub/study/student-discipline.php
- *attach this Cover Sheet, completed and signed to the work being submitted*

SECTION 2: Confirmation of Authorship

The acceptance of your work is subject to your signature on the following declaration:

I confirm that I have read, understood and will abide by the University statement on cheating and plagiarism defined over the page and that this submitted work is my own and where the work of others is used it is clearly identified and referenced. I understand that the School of Engineering reserves the right to use this submitted work in the detection of plagiarism.

Signed: ____________________Mateusz Leputa__________________________

Date:____________19/11/2018____________________________

Note: Work submitted for continuous assessment will not be marked without a completed Cover Sheet. Such work will be deemed 'late' until a completed cover Sheet is submitted and will be subject to the published penalty for late submission.

Cheating in any assessment, whether formative or summative, can result in disciplinary action being taken under the University's Code of Practice on Student Discipline. For these purposes "Cheating" includes:

(a) Possession in an examination of material or electronic device which has not been

authorised in writing by the relevant Course Co-ordinator. Students whose first language is not English may, however, refer to a dictionary where this is approved by the Head of the School responsible for the examination;

(b) Copying from another student in an examination;

(c) Removing an examination book from an examination room;

(d) Impersonating another candidate in relation to any assessment;

(e) Permitting another person to impersonate oneself in relation to any assessment;

(f) Paying or otherwise rewarding another person for writing or preparing work to be

Submitted for assessment;

(g) Colluding with another person in the preparation or submission of work which is to be assessed. This does not apply to collaborative work authorised by the relevant course coordinator.

(h) Plagiarism. Plagiarism is the use, without adequate acknowledgment, of the intellectual work of another person in work submitted for assessment. A student cannot be found to have committed plagiarism where it can be shown that the student has taken all reasonable care to avoid representing the work of others as his or her own.

Contents

Introduction

The aim of this assignment was to design a control system for a mass spring damper system shown in *fig.1:*

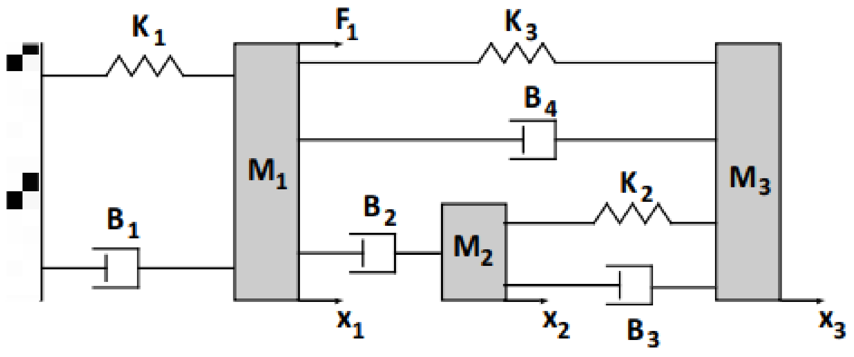


Figure 1

Where $m_1 = m_2 = m_3 = 0.01$ kg, $k_1 = 1\times10^4$ N/m, $k_2 = 1\times10^5$ N/m, $k_3 = 1\times10^6$ N/m, $b_1 = 0.1$ Ns/m, $b_2 = b_3 = b_4 = 0.4$ Ns/m and $F_1 = 5.6$ kN.

To obtain an analytical solution the system was derived using a general mass spring damper system state space representation, which can be obtained as follows.

$$F = m\ddot{x} + b\dot{x} + kx \text{ (1)}$$

To obtain the MSD equilibrium equation, the equilibrium equation for each of the masses needs to be found, hence:

$$F_1 = m_1\ddot{x_1} - b_4(\dot{x_3} - \dot{x}_1) - b_2(\dot{x_2} - \dot{x}_1) + b_1\dot{x}_1 + k_1x_1 - k_3(x_3 - x_1) \text{ (2)}$$

$$0 = m_2\ddot{x_2} - k_2(x_3 - x_2) - b_3(\dot{x}_3 - \dot{x}_2) + b_2(\dot{x}_2 - \dot{x}_1) \text{ (3)}$$

$$0 = m_3\ddot{x_3} + b_4(\dot{x_3} - \dot{x}_1) + b_3(\dot{x_3} - \dot{x}_2) + k_3(x_3 - x_1) + k_2(x_3 - x_2) \text{(4)}$$

Now in matrix form (1):

$$F = \begin{bmatrix} m_1 & 0 & 0 \\ 0 & m_2 & 0 \\ 0 & 0 & m_3 \end{bmatrix}\ddot{x} + \begin{bmatrix} b_4 + b_2 + b_1 & -b_2 & -b_4 \\ -b_2 & b_2 + b_3 & -b_3 \\ -b_4 & -b_3 & b_4 + b_3 \end{bmatrix}\dot{x} + \begin{bmatrix} k_1 + k_3 & 0 & -k_3 \\ 0 & k_2 & -k_2 \\ -k_3 & -k_2 & k_3 + k_2 \end{bmatrix}x \text{ (5)}$$

And F is the input vector:

$$F = \begin{bmatrix} F_1 \\ 0 \\ 0 \end{bmatrix} \text{ (6)}$$

Now convert to state space. Define state variables:

$$x_1 = \dot{x} \;\; x_2 = x \;\; y = x \;\; F = u \text{ (7)}$$

Rearrange equation (1):

$$\dot{x}_1 = \ddot{x} = -\frac{b}{m}\dot{x} - \frac{k}{m}x + \frac{1}{m}F \text{ (8)}$$

Now Substitute the matrixes form (5):

$$\begin{bmatrix} x_1 \\ x_2 \end{bmatrix} = \begin{bmatrix} -\frac{b}{m} & -\frac{k}{m} \\ 1 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} + \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} u$$

$$y = \begin{bmatrix} 0 & 1 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \end{bmatrix} (10)$$

Hence the state space equations are:

$$\dot{x} = \begin{bmatrix} \frac{-(b_4+b_2+b_1)}{m_1} & \frac{b_2}{m_2} & \frac{b_4}{m_3} & -\frac{k_1+k_3}{m_1} & 0 & \frac{k_3}{m_3} \\ \frac{b_2}{m_1} & \frac{-(b_2+b_3)}{m_2} & \frac{b_3}{m_3} & 0 & -\frac{k_2}{m_2} & \frac{k_2}{m_3} \\ \frac{b_4}{m_1} & \frac{b_3}{m_2} & \frac{-(b_3+b_4)}{m_3} & \frac{k_3}{m_1} & \frac{k_2}{m_2} & \frac{-k_3-k_2}{m_3} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} & 0 & 0 \\ 0 & \frac{1}{m_2} & 0 \\ 0 & 0 & \frac{1}{m_3} \\ 0 & 0 & 0 \\ 0 & 0 & 0 \\ 0 & 0 & 0 \end{bmatrix} u \ (9)$$

$$y = \begin{bmatrix} 0 & 0 & 0 & F_1 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \\ 0 & 0 & 0 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} (10)$$

However, the system has 3 outputs and 3 inputs in this form, but it will be assumed that only input one and output 2 can be measured and controlled hence the matrixes can be reduced to:

$$\dot{x} = \begin{bmatrix} \frac{-(b_4+b_2+b_1)}{m_1} & \frac{b_2}{m_2} & \frac{b_4}{m_3} & -\frac{k_1+k_3}{m_1} & 0 & \frac{k_3}{m_3} \\ \frac{b_2}{m_1} & \frac{-(b_2+b_3)}{m_2} & \frac{b_3}{m_3} & 0 & -\frac{k_2}{m_2} & \frac{k_2}{m_3} \\ \frac{b_4}{m_1} & \frac{b_3}{m_2} & \frac{-(b_3+b_4)}{m_3} & \frac{k_3}{m_1} & \frac{k_2}{m_2} & \frac{-k_3-k_2}{m_3} \\ 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \\ 0 & 0 & 1 & 0 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} + \begin{bmatrix} \frac{1}{m_1} \\ 0 \\ 0 \\ 0 \\ 0 \\ 0 \end{bmatrix} u \ (11)$$

$$y = \begin{bmatrix} 0 & 0 & 0 & F_1 & 0 & 0 \end{bmatrix} \begin{bmatrix} x_1 \\ x_2 \\ x_3 \\ \dot{x}_1 \\ \dot{x}_2 \\ \dot{x}_3 \end{bmatrix} (12)$$

This works as there in no cross coupling in the C and B matrixes of the state space equation.

The system responds identically to the reference input-output provided as shown in (*fig.2*).

The input signal is a triangle wave of frequency 5 Hz and amplitude 1. The bode plot for the open loop system can be seen in (*fig.3*). It can immediately be observed that the problematic poles reside at 91.2 Hz, 612 Hz and 2280 Hz. Although the latter 2 have a gain of -12.4 dB and -16.5 dB respectively, whilst the first pole has a gain of 39.7 dB, hence that is the majority problematic dynamic in this system.

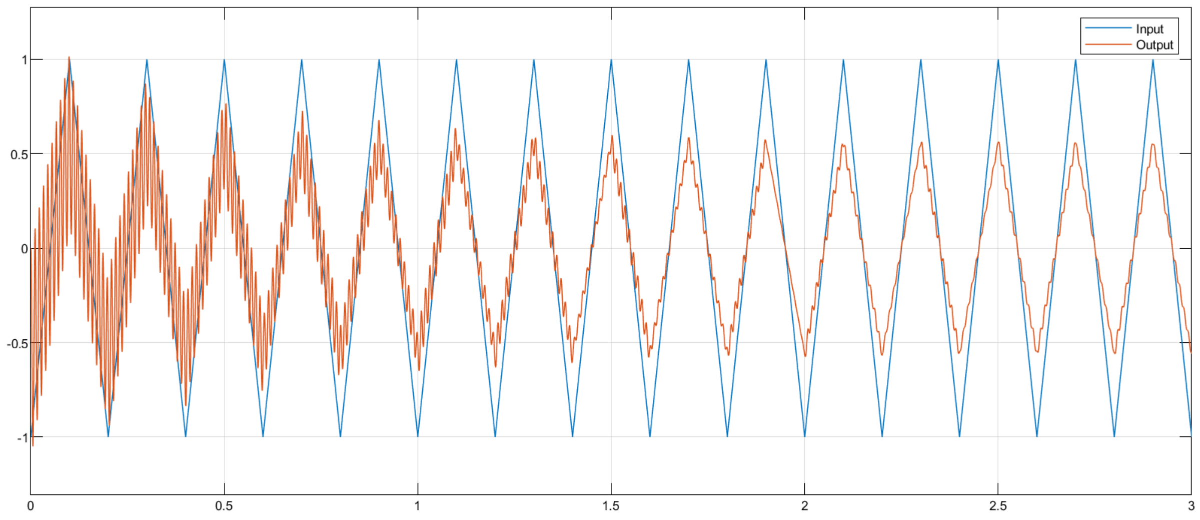


Figure 2 – open loop input-output behaviour for a 5 Hz triangle wave input

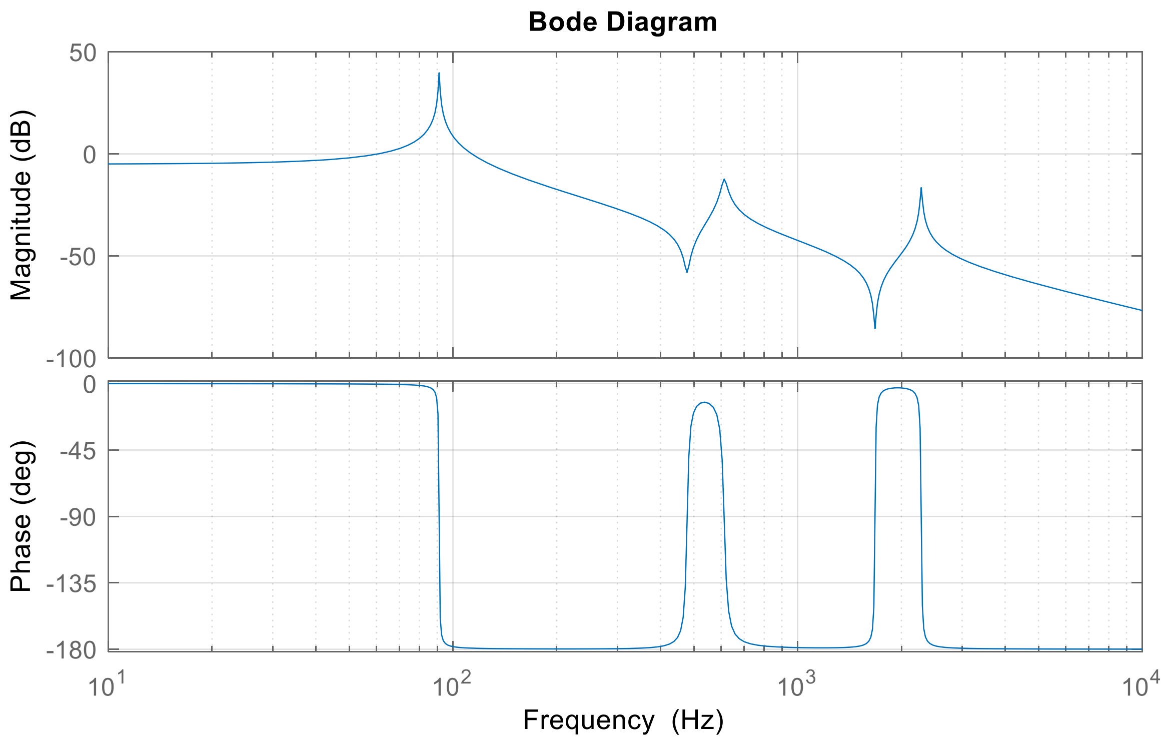


Figure 3 – open-loop bode plot of the system

The compensation and or control of these variables can be achieved through several means and control schemes. In this report LQR, IRC and Genetic Algorithms are used to obtain solutions that could be used to control the system. Since the example input to the system is a 5 Hz triangle wave let it be assumed that the system will operate within a similar bandwidth of interest. To verify the system is observable and controllable the MATLAB functions *rank(ctrb(A,B),10^-9)* and *rank(obsv(A,C),10^-9)*, to find controllability and observability respectively. Both values return as 6. Since this is a 6-state system it means this system is a minimal realisation, hence no transform matrix needs to be found and the system can remain as it is represented for control purposes. Note the additional 10^{-9} term, which defines the tolerance of the calculation, otherwise MATLAB would truncate the numbers within the matrix and return values that are wrong.

LQR

LQR is a method for optimal pole placement for a given weight matrixes *Q* and *R*. Depending on the application the R and Q must be weighed appropriately to the application hence some assumptions have to be made. First let the system states be analysed in the open-loop system. The matrixes *A, B, C* and *D* were implemented in a Simulink model as shown in the figure below:

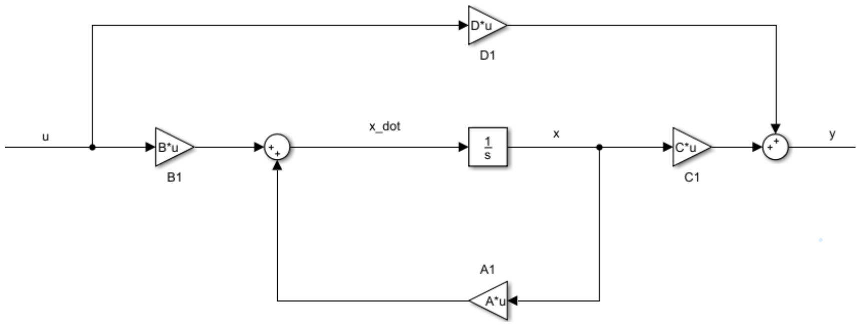


Figure 4 – Matlab model for the MSD system

The simulated states have the values presented in *fig.5* and *fig.6:*

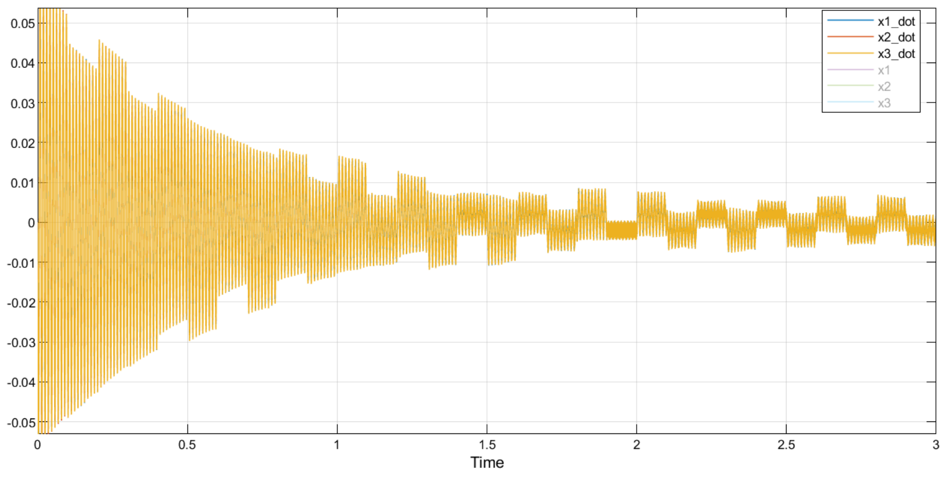


Figure 4– $\dot{x}_1$, $\dot{x}_2$, $\dot{x}_3$ *state space vectors*

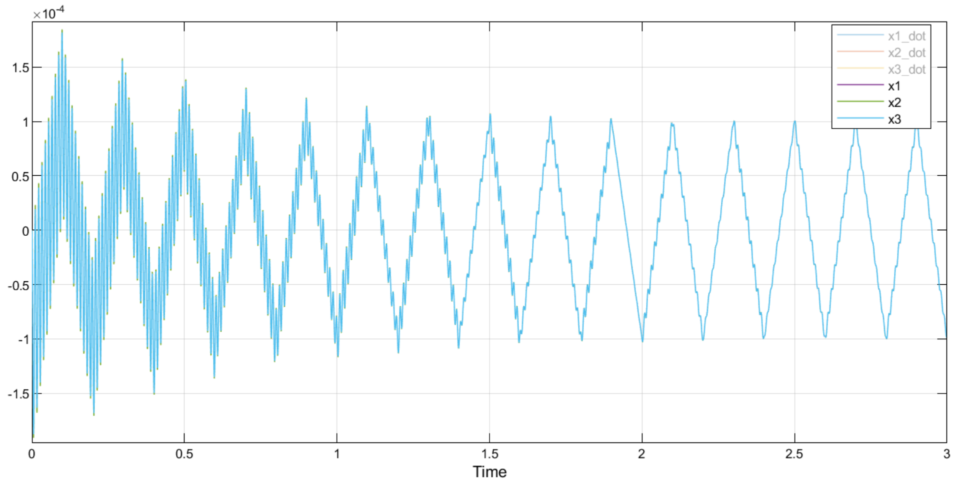


Figure 5– x_1, x_2, x_3 *state space vectors*

This provides an insight into what magnitudes the system operates at hence when controllers are introduced the controller effort can be quantified and compared to the original states of the system.

Setting the all the diagonal vales in a 6x6 Q matrix to 1, and the R value to 1, yields the system output for a triangle input, as shown in *fig.6 (LQR-1)*. As can be seen result is not satisfactory, with an over shoot of ~85%, a settling time of 0.13 seconds and a gain of ~1.78, the system needs to be adjusted further.

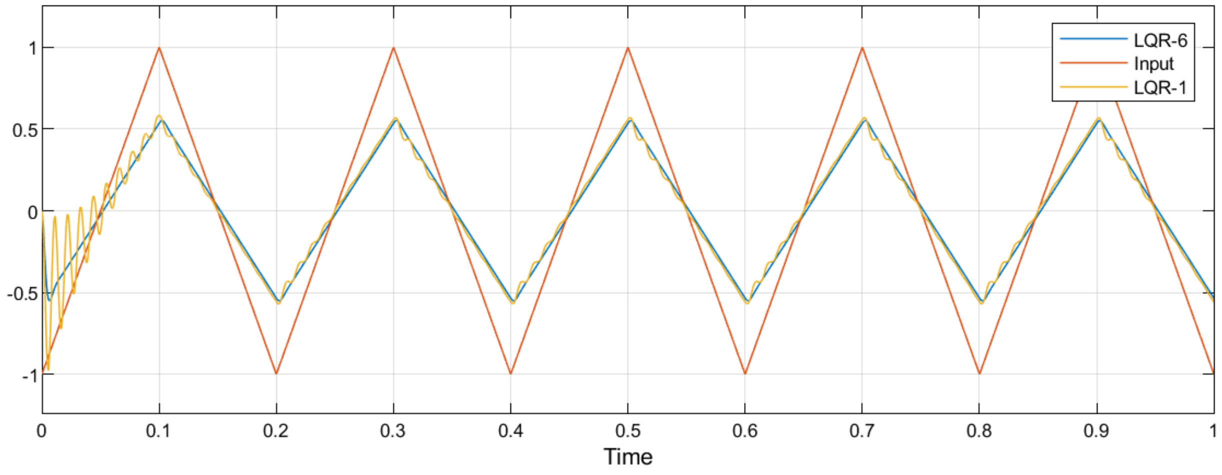


Figure 6 – comparison of different Q and R value results for LQR tuning.

For the first trace in *fig.6* (*LQR-6*) the values were adjusted according to the magnitudes of the errors that are permissible within the output. As demonstrated in [1, pp.190-195], a good "rule of thumb" is to attach large values to states that should remain small and values that should contribute to the cost function more should have smaller weight attached to them. Once a solution that works was found the magnitude of the individual elements were adjusted until an even better performance was found. As can be observed in *Table 1 LQR-2* Shows an improved performance as the input weight was made 100 times more important than other states, relatively. By decreasing the "importance" of the velocity states 100 times (*LQR - 3*), the system showed no overshoot and a worse settling time as well as more controller effort. Despite seemingly working better, the system was no longer able to track the reference as it became overdamped, effectively becoming a filter for the higher input frequencies. In *LQR – 4* increasing the *R* weight to 1 once again yielded results akin to *LQR-2*, however, this is expected as their relative magnitudes of variables in *Q* and *R* were almost identical with exception of the acceleration states, what was concluded is that the weights on the acceleration states can be of the same magnitude as the velocity states as shown in *LQR – 5*. Finally, the states related directly to the output one was adjusted along with the *R* weight, which improved the system performance drastically as can be seen in the table. However, the absolute maximum controller output was higher than the previous systems

System Designation	Required Reference gain N	Q diagonal values (on 6x6 matrix)	R	Overshoot (%)	Settling time (s)	Max controller output
Open-loop	1.78	N/A	N/A	98.29	2.38	N/A
LQR – 1	1.78	1,1,1,1,1,1	1	84.92	0.13	0.08
LQR – 2	1.78	1,1,1,1000,1000,1000	0.01	16.09	0.01	0.58
LQR – 3	1.78	100,100,100,1000,1000,1000	0.01	00.00	0.04	1.01
LQR – 4	1.78	100,100,100,1000,1000,1000	1	16.13	0.01	0.59
LQR – 5	1.78	100,100,100,100,100,100	1	16.13	0.01	0.59
LQR – 6	1.78	50,100,100,100,100,100	0.5	7.32	0.01	0.67

Table – 1– Q matrixes and resultant systems

Depending on the application *LQR – 4* or 5 might provide adequate system performance. But if the controller can drive the output at 0.58 of the of the input signal, (in this system input of 1 corresponds to the value of F_1) it should be able to use the gains form *LQR-6,* in which case the force max force the controller would need to output is 0.67, hence only about 15% higher. This however would mean that the system is always using more energy. It might also not be a consideration at all since the controller and input signal might be implemented on the same device, in which case the controller output is simply a modification to the signal.

With an observer designed using *lqr(A', C', B*(B'), 0.1)* in MATLAB, the system becomes:

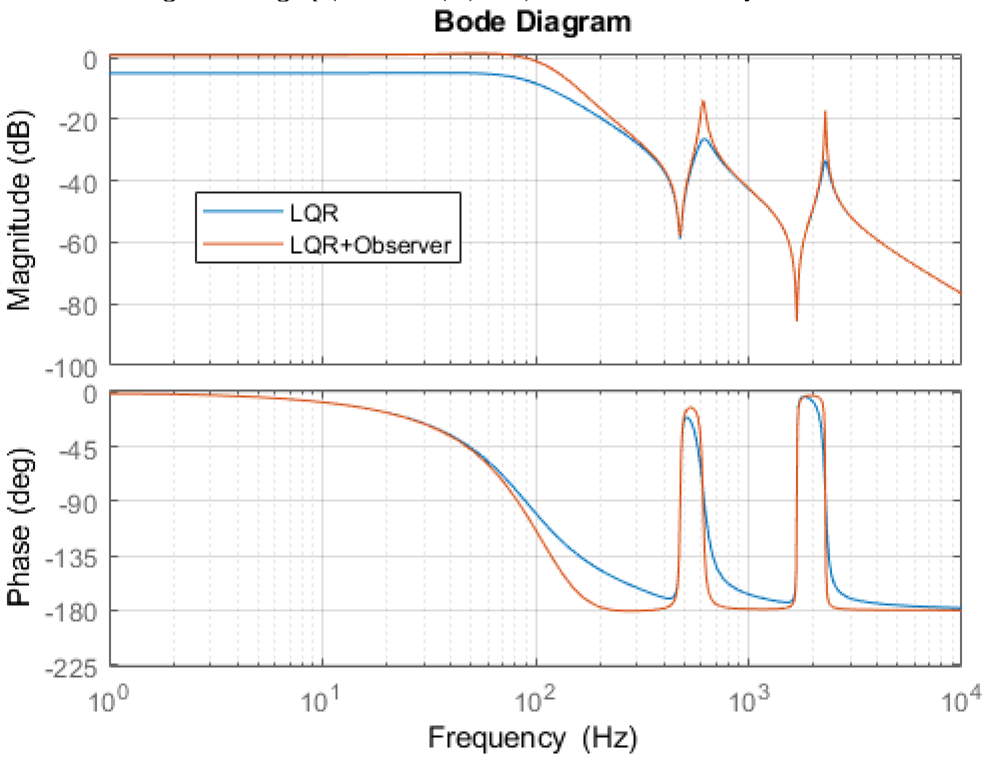


Figure – 7

The gains for *LQR – 6* were [19.432, -0.596,3.842,15499.184, -6322.524, -9176.629]

IRC

An integral resonant controller [3] is a frequency space (transfer function) method for controlling a system, it has the advantage of being simple in terms of calculations required, it only requires finding a *k* gain for an integral controller, which can be found using root locus or simply trial and error with educated guesses with respect to the resultant pole placement. The feed-through term *D* can be found using an analytical solution or simply by adjusting the value until a satisfactory performance is achieved.

This control scheme has the advantage of being robust to changes in the mass of the system but has an unnecessarily high gain at low frequency/DC input which requires an additional tracking controller.

The layout of a typical IRC controller (a), and its alternative, implementable layout is (b). Where the integrator on the feedback path has the equation shown in [4, pg.59] and its state space representation is shown in equations (13), (14).

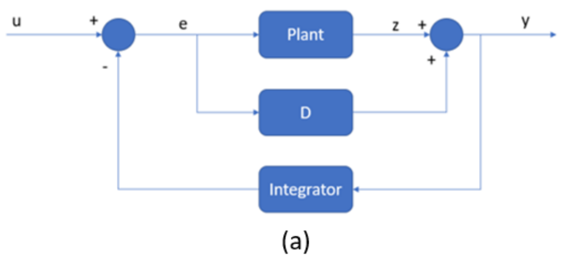
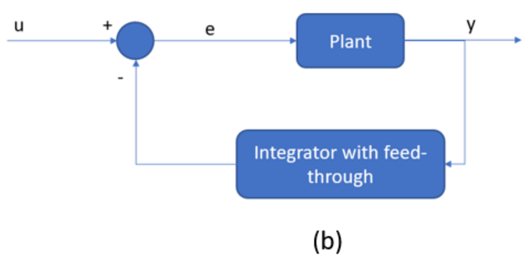


Figure 8 – (a) integrator and feedthrough (D) controller (b) alternative layout [4, pg.59]



State space equation for integrator with feed-through:

$$\dot{x} = [-(K \times D)]x + [1]u$$

$$y = [K]x + [0]u$$

The solution was found experimentally in process similar to the LQR tuning in the previous section. The table below presents performance of selected systems

System Designation	Required Reference gain N	K	D	Overshoot (%)	Settling time (s)	Max controller output
IRC – 1	0.79	-500	-1	22.60	0.028	1.236
IRC – 2	0.79	-600	-1	28.56	0.028	1.251
IRC – 3	0.79	-400	-1	15.61	0.032	1.212
IRC – 4	0.79	-300	-1	8.24	0.038	1.170
IRC – 5	0.63	-300	-0.8	0.00	0.048	1.906
IRC – 6	0.54	-600	-0.8	0.06	0.018	2.181
IRC – 7	0.57	-700	-0.8	1.22	0.018	2.216

Table 2 – tuning an integral resonant controller

The resultant IRC system bode plot can be seen below along the LQR resultant system:

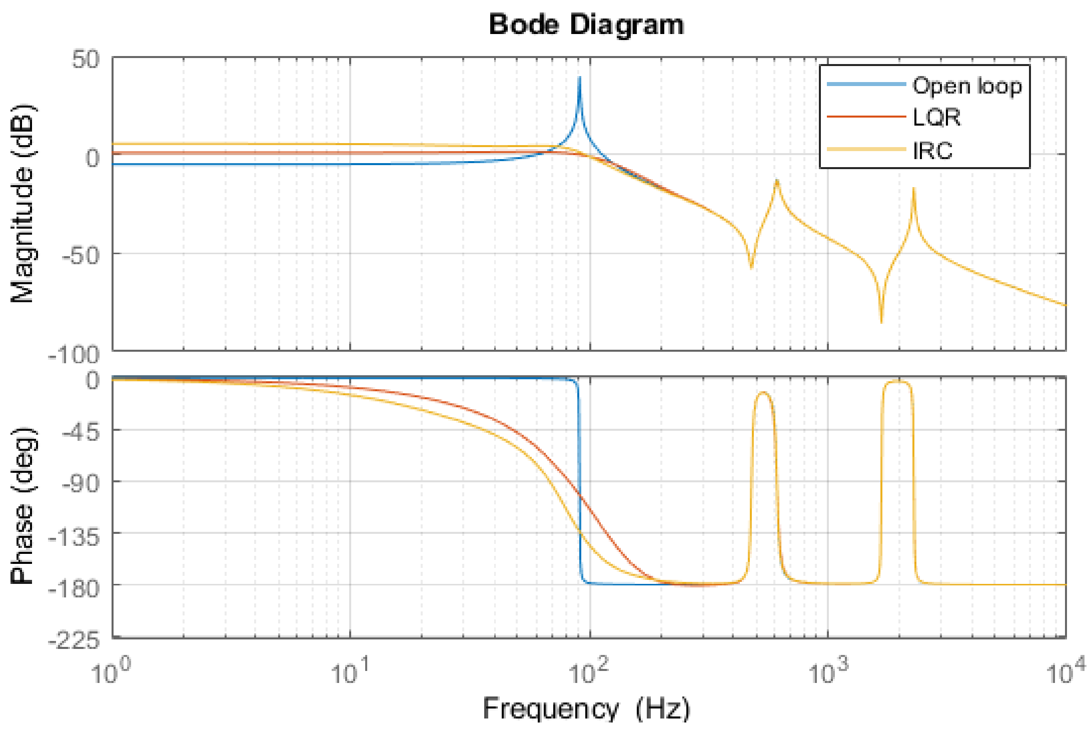


Figure 9 – Comparison of IRC and LQR control schemes

As can be observed both systems yield comparable and acceptable results. The however shows much more controller effort, 3-4 times as much in the case of *LQR-6* to *IRC-7*. This is possibly down to the LQR scheme taking full advantage of the state-space rather than a simple input to output relation like the IRC scheme. Nevertheless, the additional controller output signal might not be a concern as mentioned before in the case of the LQR, the controller output might be a modification the input value of the system from a digital controller.

Where IRC system is better over the is the easier computation, the LQR system requires an implementation of an observer, which needs to be designed and implemented on the target controller and it requires adding a gain to the reference signal. Hence whilst LQR might be the better solution for applications with a bigger computational budget, an IRC controller is better for cases where the computational power of the controller is limited. IRC can also be designed more easily with only 2 parameters to tune for the single output. In a MIMO system this might not be the case and LQR might be the computationally less expensive method.

Another measure and comparison that needs to be made is the robustness to system changes, with the original system and an observer used for the LQR the system performs as follows:

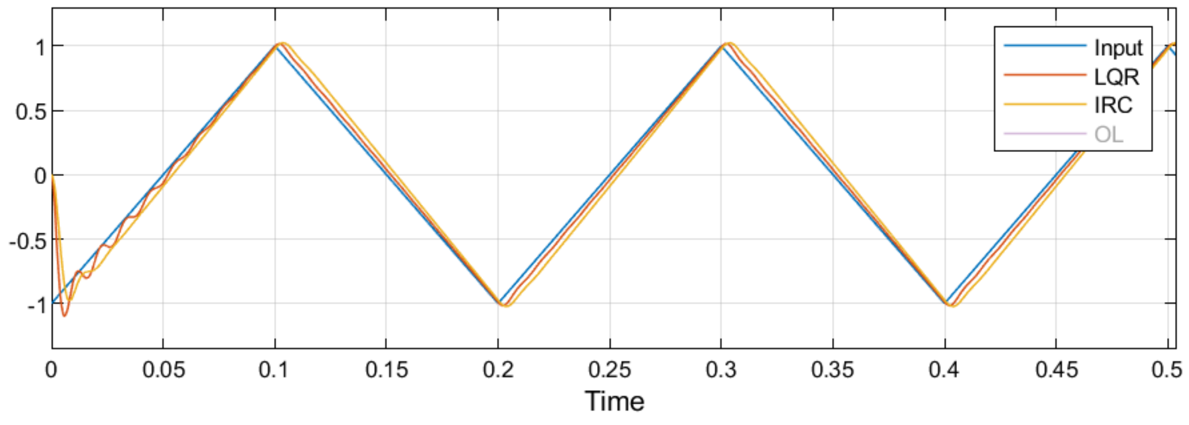


Figure 10 – Note the LQR system includes an observer

Now changing the parameters by 10%

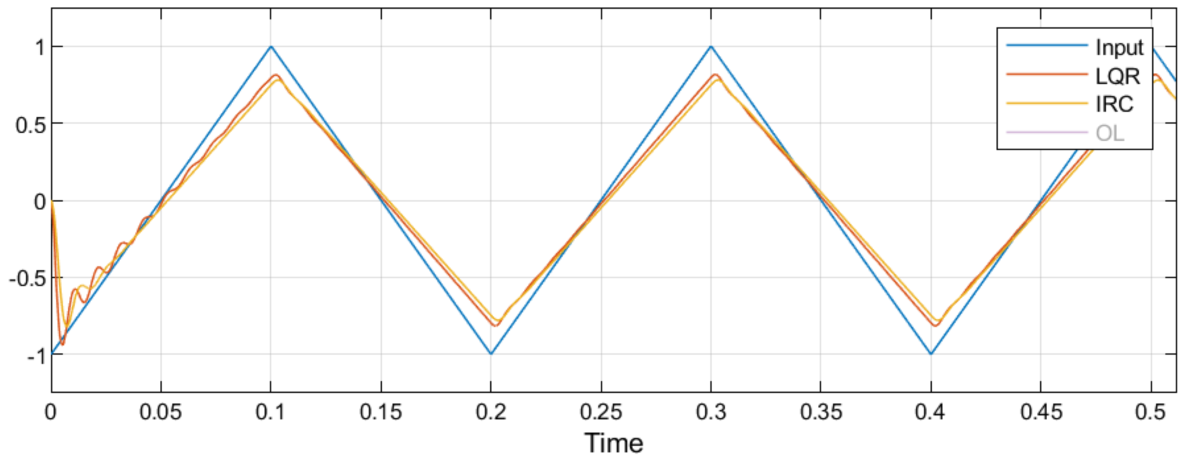


Figure 11

Now changing the parameters by 40%

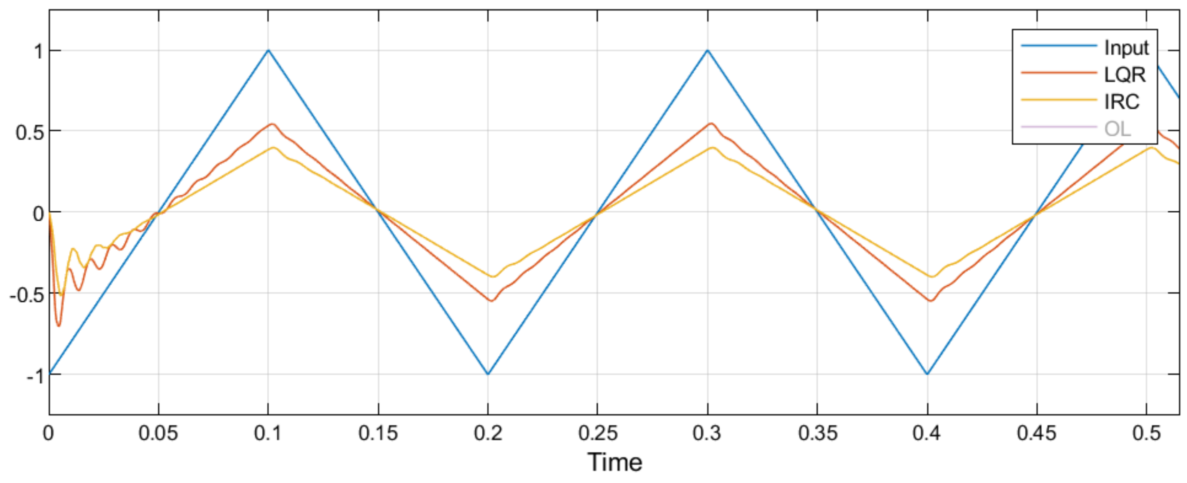


Figure 12

What can be concluded from *fig.10, fig.11* and *fig.12* is that as the systems variables diverge form the original specification, the system performance degrades, and additional gain changes are required. However, introducing additional gains might make the system might become unstable at the first resonant peak (~93 Hz), as the gain around that peak becomes more than 0 dB. As can be observed in *(fig .14)* the LQR phase margin changes significantly by additional -360 degrees, dues to the change in variables. Whilst it is an extreme edges case that the system variables might change by 40%, it does add to the reasons one might choose an IRC controller for the task

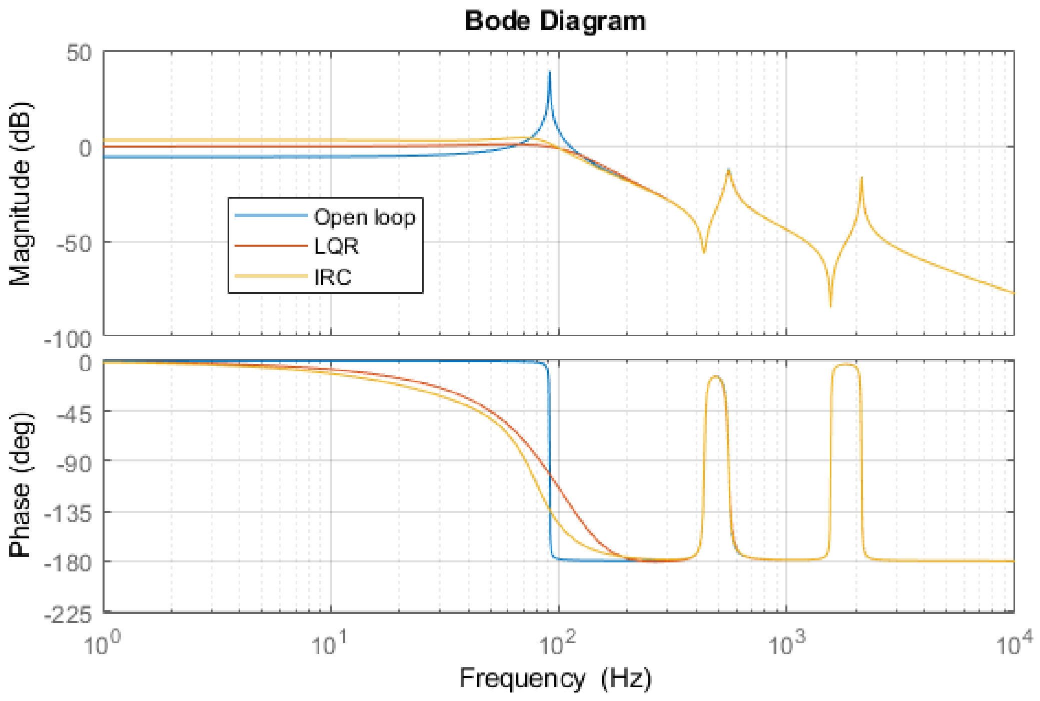


Figure- 13 system after 10 % variation in the system variables

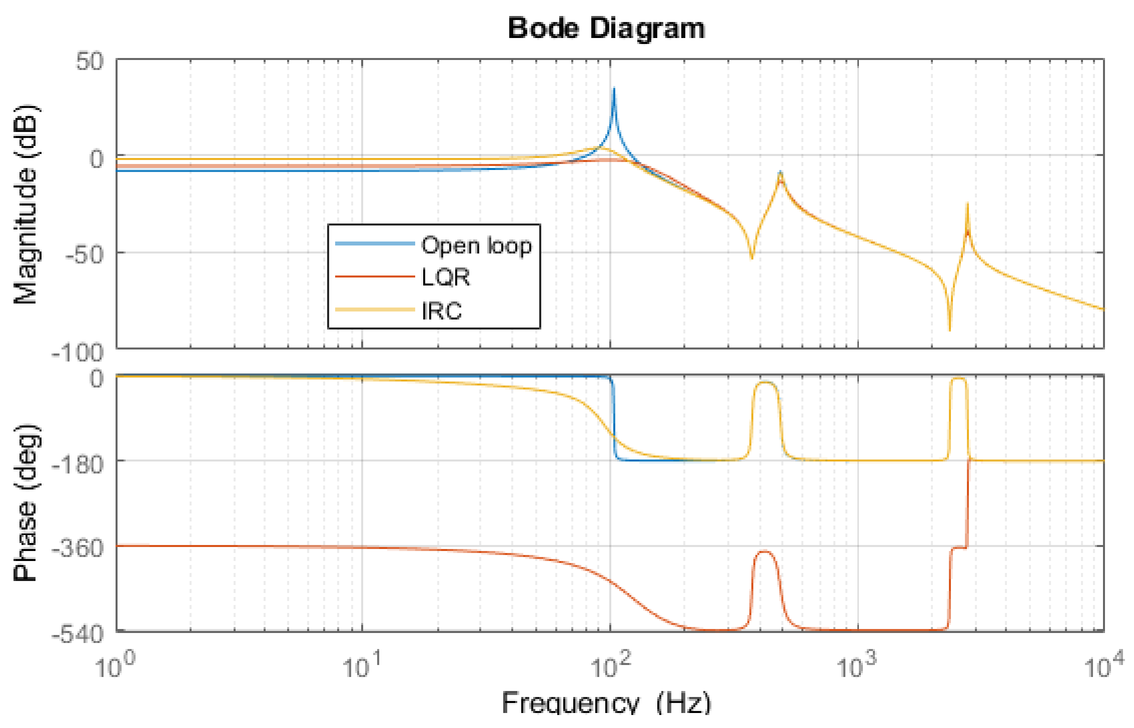


Figure -14 system after 40 % variation in the system variables

The system values were as follows for both for 10 % and 40 % respectively:

m_1 = m_2 = m_3 =0.011 kg, k_1 =1.1×10^4 N/m, k_2 =0.9×10^5 N/m, k_3 = 0.95×10^6 N/m, b_1= 0.11 Ns/m, b_2= b_3= b_4= 0.43 Ns/m and F_1= 5.6 kN.

m_1 = 0.014 kg, m_2 = 0.012 m_3 =0.006 kg, k_1 =1.4×10^4 N/m, k_2 =0.7×10^5 N/m, k_3 = 1.25×10^6 N/m, b_1= 0.15 Ns/m, b_2= *0.48* Ns/m ,b_3=, b_4= 0.27 Ns/m and F_1= 5.6 kN.

Genetic algorithm K gain optimization

A genetic algorithm, as the name suggests is written to imitate natural selection or evolution, at least superficially. A typical genetic algorithm operates as follows:

1. Generate a random population of *x* size.
2. Test viability with a suitable cost function.
3. Select a proportion of the best candidates and discard all others.
4. Creates a new population of *x-best candidates* members based on the best candidates using mutation and random cross-over. Keep the best candidates, this is known as an elitist generic algorithm feature.
5. Go back to step 2 and repeat until a set number of iterations have passed or the best candidate has satisfied a set criterion of the cost function.

A cross-over function randomly mixes equivalent parts of the population. This randomly looks for good combinations of variables within a population.

Mutation is a function of the cost and a random number within a normal distribution, having a random element with normal distribution allows for partially new candidates or “genes” to be introduced, whilst most of the mutations remain small. This effectively ends up being a random search of a multivariable space for local minima.

In the case of this code the following cross-over code was adopted:

$$K_{new} = K_{r1}[elements\ 1\ to\ random\ value]\ append\ with\ K_{r2}[element\ random\ value\ to\ 6] (14)$$

Where K_{r1} and K_{r2} are random *K* gains from population of best candidates. The result is then passed to the mutation function

$$K_{j,new\ mutated} = K_{j,new} + K_{j,new} \times m \times\ random\ no.normal\ distribution \times cost\ prev.gen\ (15)$$

Where K_j is the jth element of the *K* gain array, *m* is the mutation ratio, *cost prev. gen* is the lowest cost from previous generation and *random no.normal distribution* is a random number generated according to random distribution. This means as cost gets smaller, the changes that can be made to variables get smaller and the search narrows down the multi-dimensional space that its looking in. This however means that the code might get stuck in local minima, if the starting random variables do not reach or are discarded before they can find better set of gains

The cost function has to be defined appropriately to the desired performance, much like *Q* and *R*.

For the first attempt the cost function a MATLAB function was designed so that it takes in the desired signal, system input (in this case they are the same), and the state space function with gains itself. It then uses the *lsim()* command to simulate the output. The output is then subtracted from the reference and the average error for the data is found. The code snippet is shown in the *code 1.*

```
function cost_out = costgen(s,u,t,tar)
    cost_out =0;
    y = lsim(s,u,t);
    for i = 1:length(u)
       cost_out = cost_out + abs(tar(i)-y(i));
    end
    cost_out = cost_out/length(u);
end
```

Code - 1

As can be immediately gathered, running this for a population of 100 systems requires 100 simulations. Whilst Matlab parallel computing toolbox can be used to do multiple simulations at once, it is not the most efficient method, an example result of run of 500 iterations with a population of 100 (resulting in 50,000 *lsim()* simulations , takes approximately 10 minutes on an average home computer):

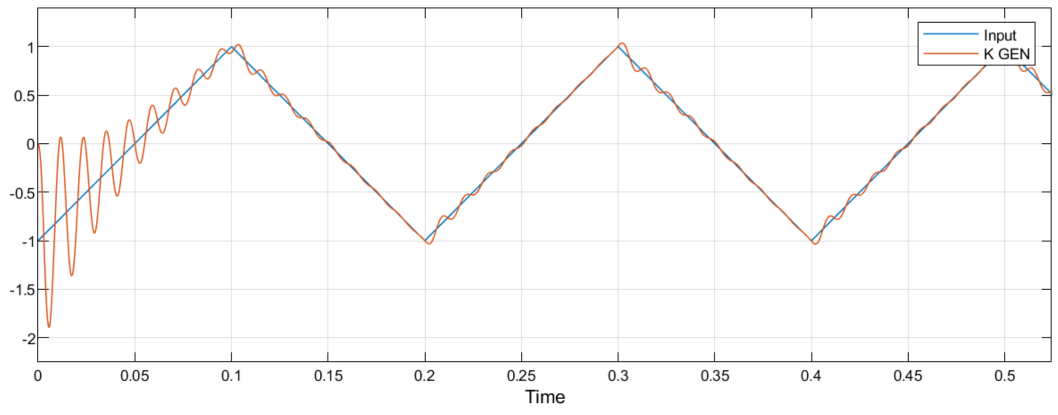


Figure – 15 – system output for generated gains, with observer

Where the *K* gains are [4.029, -0.685,0.088,23.373,97.582, -4597.176]. Whist the *K* gains for the *LQR-6* system were [19.432, -0.596, 3.842,15499.184, -6322.524, -9176.629]. Not that the generated gains requires a small input gain of 1.1, when it is reading the states directly from the state-space representation in simulink, with an observer this rises to 2.9. This might be a huge advantage in some cases

The best cost for each iteration was also saved as to demonstrate the progress that the code finds:

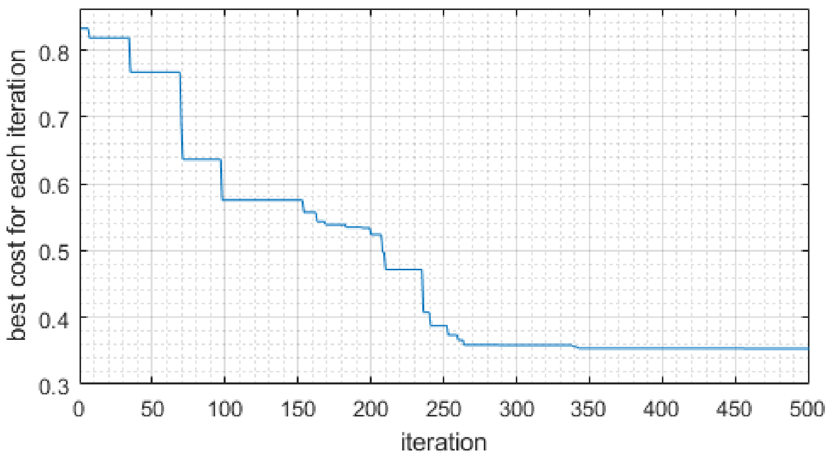


Figure – 16 – lowest cost for each iteration of the code

For reference the cost of *LQR-6* was 0.5698. As seen in the figure the generation plateaus after ~300, iterations, this was the case for multiple runs of the code. What can also be observed is that the cost gets stuck on a series of variables for up to 50 iterations before a new optimal set of *K* gains is found. It is entirely possible that the calculation will get stuck in a local minimum that will be much higher than what is possible.

Since the code was proven to work, the cost function needed optimization. Most systems characteristics can be accurately defined by its step response, MATLAB contains a *setp()* and *stepinfo()* function which yield all the important characteristics of a step response. In this case overshoot, undershoot, peak and settling time were the aspects that should be optimized for reference tracking.

The new cost function was therefore:

```
function cost_out = costgen1(s)
    cost_out =0;
    S1 = stepinfo(s,'SettlingTimeThreshold',0.02);
    cost_out = S1.SettlingTime +S1.Overshoot/100+S1.Undershoot/100+abs(1/2-
S1.Peak);
end
```

Code 2

Note the ½ term for the last term as we aim to have a reference gain of ~2, in similar to other 2 systems

These are the results of a 300 iteration 100 population run:

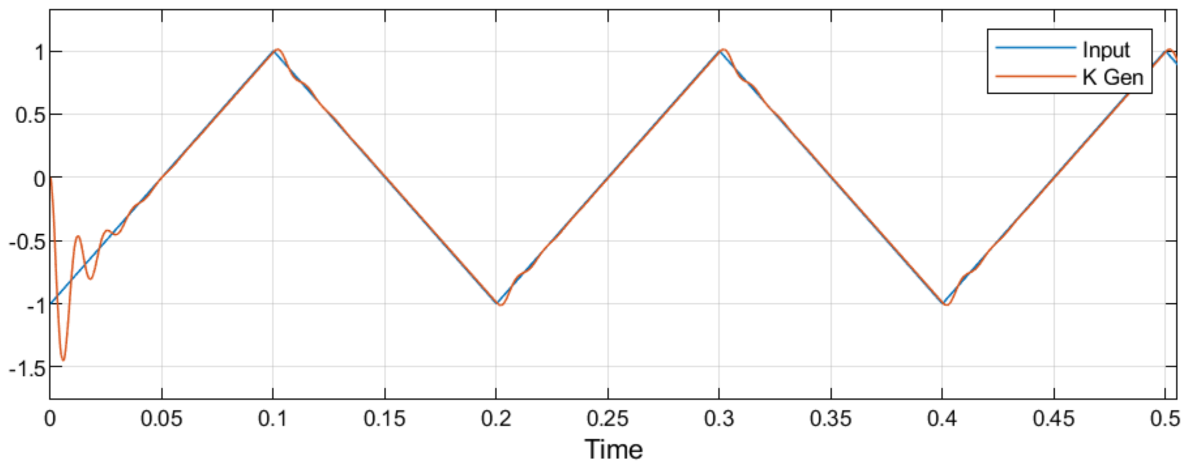


Figure – 17– test of genetically generated K gains with observer

Note that this system requires a small input gain reference gain of 1.01, with observer this once again rises to 2.22. The is almost similar performance to the previously generated system, with the difference in reference gain.

Below is the performance of two systems generated with the 2 different cost functions:

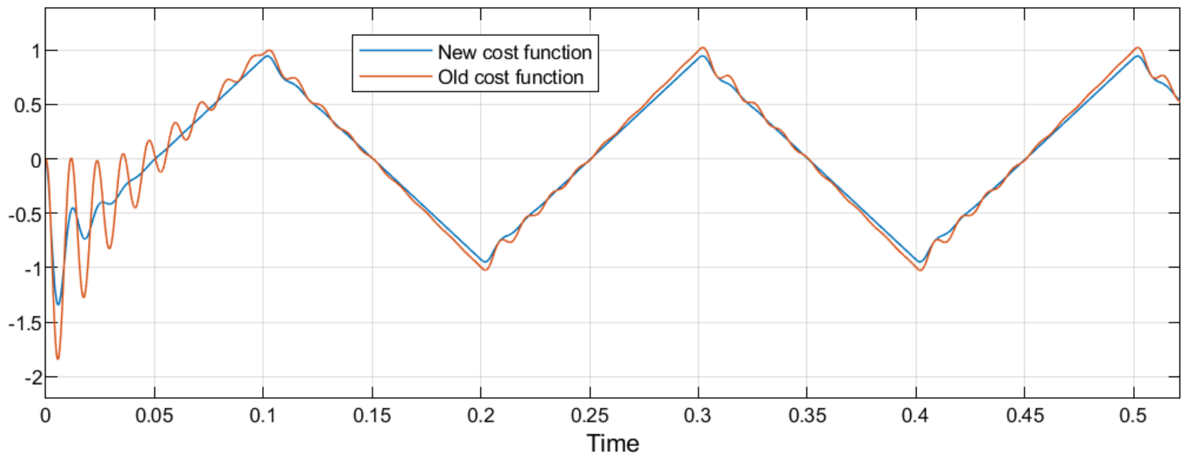


Figure – 18– cost functions results comparison

As can be seen in *fig.18* the new cost function yields a much better performance of the system.

The absolute maximum controller output for this system is 1.3, hence it is comparable to IRQ but has a performance comparable to the IRQ system.

K gain in this case was [59.135,1.450,-37.059,2207.885,728.998,-7331.429], therefore comparable in magnitude to those obtained using the LQR method for *LQR -6*.

Genetic algorithm IRC gain optimization

The genetic algorithm used in previous section was adapted to generate the *D* and *K* gain for the IRC control scheme. A typical cost curve can be seen below:

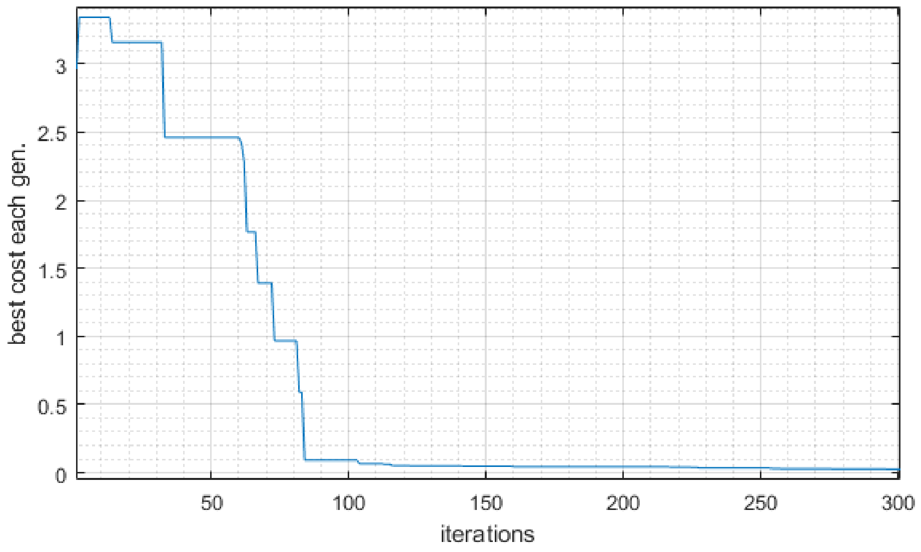


Figure – 19 – lowest cost for each iteration of the code

The decent was much sharper this time with plateaus of 20-30 generations rather than the 50 or more in the case of the state-feedback gains. Also, the solution at which the iterations occurs at less than a hundred iterations rather than 300 in the state-feedback gain problem. This is mostly due to this being a 2 variable problem rather than a 6-variable problem in state feedback form. The values for *k* and *D* in this case were -1022.5 and -0.7208 respectively and performed as follows when compared to manually tuned IRC.

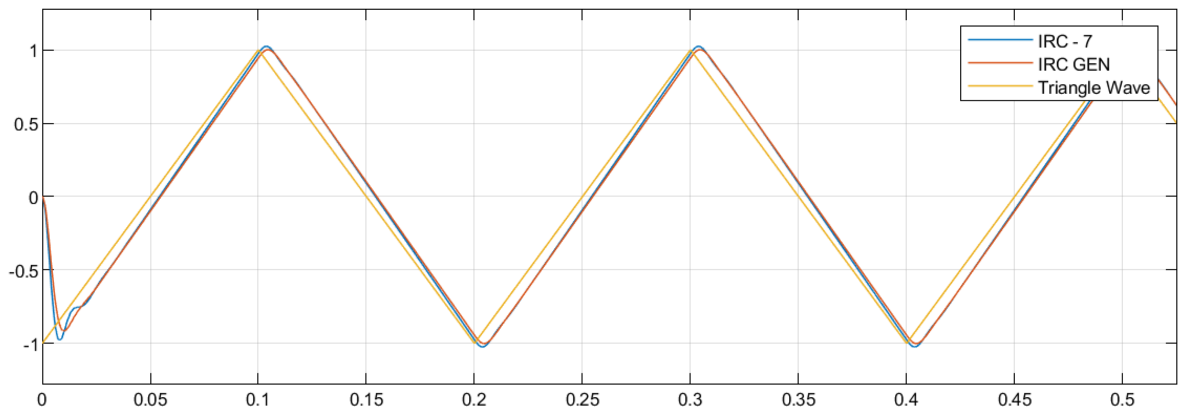


Figure – 20 – Generated vs manually tuned IRC

The systems perform similarly with the main difference being is the generated system requires an input gain of 0.42, whilst the manually tuned system requires a gain of 0.57. The algorithmically tuned system has less oscillations.

Conclusion and Evaluation

2 methods for tuning an LQR controller with an observer and an IRC controller were demonstrated. Manual experimental tuning and genetic algorithm tuning.

The IRC method shows a delay of 3.8 ms 4.5 ms for the manually tuned system and algorithmically tuned system respectively. Whilst LQR systems showed a delay of 2.1 ms for all 3 cases presented in the graph below. However, the LQR based methods tend to display more oscillations when the input changes.

In the case of the IRC, the manually tuned system performed better, but in the case of the LQR systems the genetically algorithmically tuned systems performed better, with less oscillations and smaller reference gain inputs necessary.

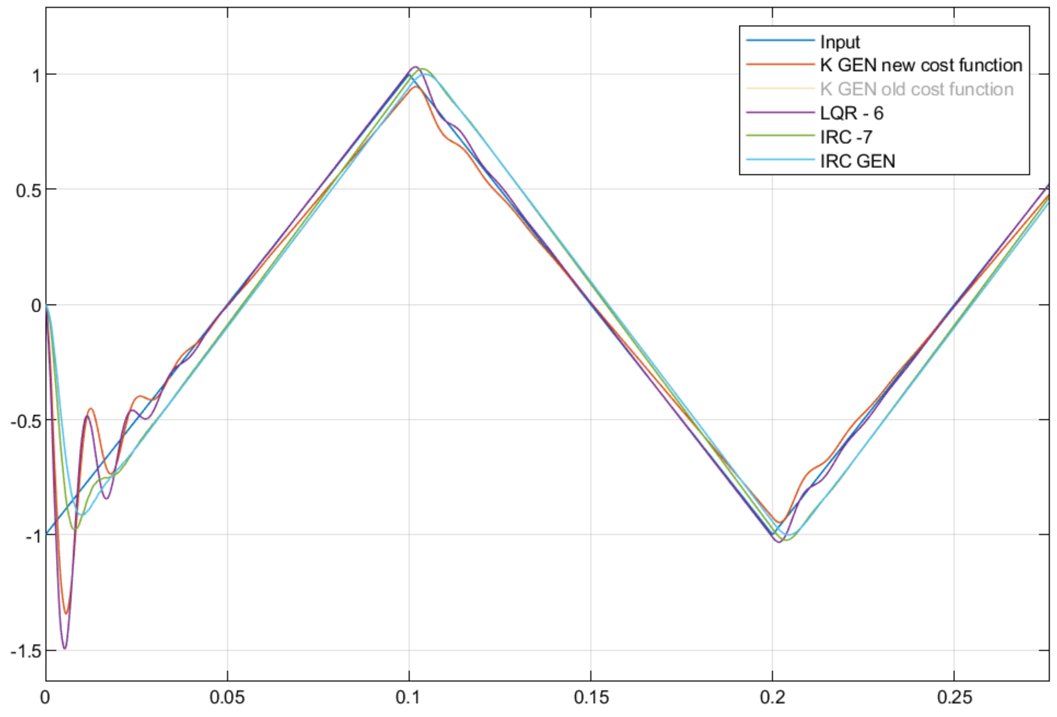


Figure – 21 – Comparison of All major discussed systems and tuning methods

Whilst the Genetic algorithm method may yield better results in some cases it has severe drawbacks, main one being the number of computations required for a system to reach satisfactory performance. However, they can be used on any layout and with an appropriately chosen cost function it can yield solutions to systems that are not possible otherwise, or that have not yet devised analytical methods for. Alternatively, it can be a useful tool to indicate solutions that one could be searching for.

References

[1] Åström, K. and Murray, R. (2008). *Feedback systems*. Princeton: Princeton University Press.

[2] Pereira, E, Moheimani, SOR & Aphale, SS 2008, 'Analog implementation of an integral resonant control scheme' *Smart Materials & Structures*, vol. 17, no. 6, 067001. DOI: 10.1088/0964-1726/17/6/067001

[3] E. Pereira, S. S. Aphale, V. Feliu and S. O. R. Moheimani, "Integral Resonant Control for Vibration Damping and Precise Tip-Positioning of a Single-Link Flexible Manipulator," in *IEEE/ASME Transactions on Mechatronics*, vol. 16, no. 2, pp. 232-240, April 2011. doi: 10.1109/TMECH.2009.2039713

[4] Díaz, I. M., Pereira, E. and Reynolds, P. (2012), Integral resonant control scheme for cancelling human-induced vibrations in light-weight pedestrian structures. Struct. Control Health Monit., 19: 55-69. doi:10.1002/stc.423

Appendix A – LQR generation code

```
close all; clear all; clc

N = 1
D = 0
m1=0.01;
m2=0.01;
m3=0.01;
k1=1e4;
k2=1e5;
k3=1e6;
b1=0.1;
b2=0.4;
b3=0.4;
b4=0.4;
F1=5.6e3;

m=[m1,0,0;0,m2,0;0,0,m3];
b=[b4+b2+b1,-b2,-b4;-b2,b2+b3,-b3;-b4,-b3,b3+b4];
k=[k1+k3,0,-k3;0,k2,-k2;-k3,-k2,k3+k2];

A = [-b*m^-1,-k*m^-1;eye(3),[0,0,0;0,0,0;0,0,0]];
B = [m^-1;[0,0,0;0,0,0;0,0,0]];
B = [B(:,1)];
C = [[0,0,0;0,0,0;0,0,0],F1*eye(3)];
C = C(1,:)
D=0;
sys1 = ss(A,B,C,D)
[L, P] =lqr(A', C', B*(B'), 0.01);
[K,P]=lqr(A,B,[50,0,0,0,0,0;0,100,0,0,0,0;0,0,100,0,0,0;0,0,0,100,0,0;0,0,0,0,100,0;0,0,0,0,0,100],0.5) %Check
Acl = (A-B*K)

sys2 = ss(Acl,1*B,C,D)

pop = 200;
kill_ratio=0.8;
mutation_rate = 0.9;
itterations = 300;

cost_base = costgen1(sys1)
cost_lqr = costgen1(sys2)

%initialize random k matrixes
K_gen =[]
a = -5*max(K);
b = 5*max(K);
parfor i=1:pop
    K_gen = [K_gen;((b-a).*rand(length(K),1) + a)'];
end
cost_history=[];
K_best_gen_j=[]
for j = 1:itterations
     cost_arr = [];
    parfor i=1:pop
        Acl = (A-B*K_gen(i,:));
        sys3 = ss(Acl,B,C,D);
        cost_arr = [cost_arr;costgen1(sys3)];
    end
    %rank by cost
    [cost_arr,I] = sort(cost_arr);
    cost_history = [cost_history, cost_arr(1)];
    K_best_gen_j = [K_best_gen_j;K_gen(I(1),:)];
    cost_history(j)
    j
    K_gen;
    K_gen_temp = [];
    for i=1:pop
        K_gen_temp(i,:) = K_gen(I(i),:);
    end
    K_gen = K_gen_temp;
```

```
%save best k
    K_gen_temp = [];
    parfor i=1:pop*kill_ratio
        K_gen_temp(i,:) = K_gen(i,:);
    end
%Crossover & Mutation
    parfor i=(pop*kill_ratio+1):pop
        r = randi(length(K)-1);
        r1 = randi(pop*kill_ratio);
        r2 = randi(pop*kill_ratio);
        K_temp = [[K_gen(r1,1:r-1)],[K_gen(r2,r:length(K))]];
        for z = 1:length(K_temp)
           K_temp(z)= K_temp(z)+mutation_rate*normrnd(0,1)*K_temp(z)*cost_history(j);
        end
        K_gen_temp = [K_gen_temp;K_temp];

    end
    K_gen = K_gen_temp;

end
    [cost_arr,I] = sort(cost_arr);
    cost_history = [cost_history, cost_arr(1)];
    K_best_gen_j = [K_best_gen_j;K_gen(I(1),:)];
        Acl = (A-B*K_best_gen_j(j,:));
        sys3 = ss(Acl,B,C,D);
```

Appendix B –IRC generation code

```
close all; clear all; clc
opts = bodeoptions('cstprefs');
opts.FreqUnits = 'Hz';
opts.PhaseUnits = 'deg';
N = 1
D = 0
m1=0.01;
m2=0.01;
m3=0.01;
k1=1e4;
k2=1e5;
k3=1e6;
b1=0.1;
b2=0.4;
b3=0.4;
b4=0.4;
F1=5.6e3;

m=[m1,0,0;0,m2,0;0,0,m3];
b=[b4+b2+b1,-b2,-b4;-b2,b2+b3,-b3;-b4,-b3,b3+b4];
k=[k1+k3,0,-k3;0,k2,-k2;-k3,-k2,k3+k2];

A = [-b*m^-1,-k*m^-1;eye(3),[0,0,0;0,0,0;0,0,0]];
B = [m^-1;[0,0,0;0,0,0;0,0,0]];
B = [B(:,1)];
C = [[0,0,0;0,0,0;0,0,0],F1*eye(3)];
C = C(1,:);
D=zeros(1,1);
%[L, P] =lqr(A', C', B*(B'), eye(3));
sys1 = ss(A,B,C,D)
K = [-700,-0.8]
integ = ss(-(K(1)*K(2)),1,K(1),0)
sys2 = feedback(sys1,integ)
bode(sys2)

pop = 100
kill_ratio=0.9;
mutation_rate = 0.7;
itterations = 500;

time = 0:0.00005:3;
u = sawtooth(time*(2*pi)*5,0.5);

cost_base = costgen111(sys1)
cost_irc = costgen111(sys2)

%initialize random k matrixes
K_gen =[]
a = 10*max(abs(K));
b = -10*max(abs(K));
```

```
cost_history=[];
parfor i=1:pop*10
    K_gen = [K_gen;((b-a).*rand(length(K),1) + a)']
end
K_gen(:,1:2)=-abs(K_gen(:,1:2));
cost_arr = [];
parfor i=1:pop*10
     sys3 = feedback(sys1,ss(-(K_gen(i,1)*K_gen(i,2)),1,K_gen(i,1),0))
     cost_arr = [cost_arr;costgen111(sys3)];
end
    [cost_arr,I] = sort(cost_arr);
    cost_history = [cost_history, cost_arr(1)];
    K_gen = K_gen(1:pop,:);
K_best_gen_j=[]
for j = 1:itterations
     cost_arr = [];
    parfor i=1:pop
        sys3 = feedback(sys1,ss(-(K_gen(i,1)*K_gen(i,2)),1,K_gen(i,1),0))
        cost_arr = [cost_arr;costgen111(sys3)];
    end
    %rank by cost
    [cost_arr,I] = sort(cost_arr);
    cost_history = [cost_history, cost_arr(1)];
    K_best_gen_j = [K_best_gen_j;K_gen(I(1),:)];
    cost_arr;
    cost_history(j)
    j
    K_gen;
    K_gen_temp = [];
    for i=1:pop
        K_gen_temp(i,:) = K_gen(I(i),:);
    end
    K_gen = K_gen_temp;
%save best k
    K_gen_temp = [];
    parfor i=1:pop*kill_ratio
        K_gen_temp(i,:) = K_gen(i,:);
    end
%Crossover & Mutation
    parfor i=(pop*kill_ratio+1):pop
        r = randi(length(K)-1);
        r1 = randi(pop*kill_ratio);
        r2 = randi(pop*kill_ratio);
        K_temp = [[K_gen(r1,1:r-1)],[K_gen(r2,r:length(K))]];
        for z = 1:length(K_temp)
           K_temp(z)= K_temp(z)+mutation_rate*normrnd(0,1)*K_temp(z)*cost_history(j);
        end
        K_gen_temp = [K_gen_temp;K_temp];

    end
    K_gen = K_gen_temp;

end
    [cost_arr,I] = sort(cost_arr);
    cost_history = [cost_history, cost_arr(1)];
    K_best_gen_j = [K_best_gen_j;K_gen(I(1),:)];
        sys3 = feedback(sys1,ss(-(K_best_gen_j(j,1)*K_best_gen_j(j,2)),1,K_best_gen_j(j,1),0))
```

Appendix C –Code for comparison of the best gains found with all methods

```
close all
clear all
clc
opts = bodeoptions('cstprefs');
opts.FreqUnits = 'Hz';
opts.PhaseUnits = 'deg';
D = 0
m1=0.01;
m2=0.01;
m3=0.01;
k1=1e4;
k2=1e5;
k3=1e6;
b1=0.1;
b2=0.4;
b3=0.4;
b4=0.4;
F1=5.6e3;
% m1=0.011;
% m2=0.011;
% m3=0.011;
% k1=1.1e4;
% k2=0.9e5;
% k3=0.95e6;
% b1=0.11;
% b2=0.43;
% b3=0.43;
% b4=0.43;
% F1=5.6e3;

m=[m1,0,0;0,m2,0;0,0,m3]
b=[b4+b2+b1,-b2,-b4;-b2,b2+b3,-b3;-b4,-b3,b3+b4]
k=[k1+k3,0,-k3;0,k2,-k2;-k3,-k2,k3+k2]

A = [-b*m^-1,-k*m^-1;eye(3),[0,0,0;0,0,0;0,0,0]];
B = [m^-1;[0,0,0;0,0,0;0,0,0]]
B = [B(:,1)]
%B = [100;0;0;0;0;0]
%C = [[0,0,0;0,0,0;0,0,0],F1*eye(3)]
C = [0,0,0,F1,0,0]
%D=zeros(3,3)
%D=[1,0,0;0,1,0;0,0,1]
D=zeros(1,1)
K = [19.4320144796553,-0.596086033903499,3.84249605994027,15499.1842390034,-6322.52436727039,-
9176.62987177804]
%[L, P] =lqr(A', C', B*(B'), 0.1);
L =
[63.7683072910489,58.1262440182190,61.1884390038592,0.150911879787866,0.139041438947538,0.1463
14089327819]
K_gen_first_cf = [327.285621524668,-115.835239544228,-198.462003734287,-459.111733231092,-
3449.51076396375,135.336106394596]
%K_gen_later_cf = [74.1010025886348,-40.6593261338343,-0.879104981387587,-
327.405476318692,1494.01592098490,33.5736898756449]
K_gen_later_cf = [59.1355391948145  1.45036061177731    -37.0591992350529   2207.88575566246
728.998697171049    -7331.42908886986]
K_IRC =[-700,-0.8]
K_gen_IRC =[-1022.47650328727,-0.720756182281557]
SYS1 = ss(A,B,C,D)
compensator = ss(A-B*K-L'*C,L',K,D)
LQR = ss(A-B*K,B,C,D)
SYS_LQG = feedback(SYS1,compensator,-1)
SYS_IRC = feedback(SYS1,ss(-(K_IRC(1)*K_IRC(2)),1,K_IRC(1),0))
% bode(SYS1,opts)
% hold
% bode(SYS_LQG)
% bode(SYS_IRC)
% legend('Open loop','LQR','IRC')
% grid on
% grid minor
```