# Discrete Controller Synthesis Based On Genetic Programming

RAMI A. MAHER, MOHAMED J. MOHAMED
ELECTRIC ENGINEERING, CONTROL AND SYSTEMS ENGINEERING
Isra University, University of Technology
Amman, Baghdad
JORDAN, IRAQ
rami.maher@iu.edu.jo, moh62moh@yahoo.com

*Abstract:* - In this paper, an alternative way of a discrete controller synthesis is introduced. The synthesis is based on a proposed genetic programming algorithm, which is named Block Diagram Oriented Genetic Programming BDOGP. The standard GP structure tree is modified in such a way to obtain a complete block diagram of the discrete controller that satisfies a deadbeat response in a closed-loop system. In one framework solution, the algorithm gives both the block diagram topology and the values to the parameters within the controller structure. A new numeric constant mutation operation is added to the algorithm to strengthen the search for optimal parameters of the BDOGP solutions. Two examples are introduced to validate the use of the proposed algorithm, and for the sake of completeness, the state-space approach design of a deadbeat response is introduced briefly. For a servo system, a comparison between the results of the GP and the conventional state-space solutions shows the accuracy of the GP approach. The second example considers a temperature control in an HVAC system.

*Key-Words:* - Genetic Programming, Block Diagram Oriented Genetic Programming, Deadbeat Controller

## 1 Introduction

Genetic Programming (GP) is a stochastic search method, which is based on natural selection and natural genetic. GP can evolve models (structures as well as parameters) for different kinds of problems in different scientific fields such as data mining, financial, electronic circuit design, etc. [1, 2, 3]. However, GP in its basic standard form could not be directly used for all applications in system control and identification. It needs to be revised according to the applications and requirements. GP has been recently used to synthesize optimal controllers for linear and nonlinear plants [4, 5, 6, 7]. Another aspect of synthesizing is the use of the block diagram representation. For a particular, a block diagram oriented simulated tool is used for structural system identification [8]. For a two-lag plant and a three-lag plant, the block diagram oriented genetic programming approach is presented in [9]. In [10], an indirect block oriented representation for a genetic programming is also explored.

The proposed BDOGP tree structure is modified to obtain a certain controller. In the field of automatic control theory, the term block diagram stands for the description of controllers as well as process models. Most of the computer-aided modeling and controlling techniques usually adopt a block diagram with a fixed structure as a problem solution. Then after the parameters or coefficients included in this structure are tuned in order to optimize the accuracy of the model or the controller.

This paper describes how a proposed BDOGP can be used as a general automated method for synthesizing both the topology and parameter values of discrete controllers for linear SISO control systems. The proposed algorithm automatically makes decisions concerning the total number of processing blocks that are employed in the controller, the type of each block, the topological interconnections between blocks, and the parameter values of the blocks. In other words, this approach is a direct method to find a discrete controller that satisfying deadbeat characteristics.

The rest of the paper discusses in the section 2 the concept and details of creating a proposed BDOGP to synthesis a discrete controller. In section 3, a brief of the deadbeat controller is introduced. Two numerical examples are given in section 4; the first one is used to show the accuracy of the proposed algorithm as compared to the theoretical approach and the second example is used to control the temperature in HVAC system. In section 5, conclusions and outlines of future work are highlighted. Used references and appendix are then followed.

## 2 The Proposed BDOGP Algorithm

The BDOGP is used here for evolution of block diagrams, and is backed up with the use of numeric constant mutation operation for parameters tuning. The blocks of these block diagrams can represent continuous (or discrete) time blocks defined in the $s$ (or $z$) domain, and linear and/or nonlinear algebraic signal processing blocks. BDOGP implements an iterative search for the optimum structure and parameters. In fact, the genetic operations defined for evolution effect only the structure of the block diagram, while the numeric constant mutation operation affects the parameters of the blocks. In the following sub sections, the used genetic operations are thoroughly handed over.

### 2.1 representations

There are high similarities between a control system block diagram and a GP tree. Each block in the block diagram represents an operation or function that is done on time domain input signals, so it is called a signal-processing block. A function node in GP tree also represents a function or operation that is done on its inputs represented by its arguments. To make the mapping between the tree structures and block diagrams possible, a block diagram cannot be directly represented by a tree without using some structural modifications and syntactic rules of construction. To show the similarity, Figure 1 gives an example of a signal-processing block for a *lag* function and the standard function node in GP.

Considering the simple mapping in Fig. 1, the GP function node will represent a *lag* function with a constant value of parameter $\tau$. To increase the flexibility of the *lag* function, it is required that $\tau$ could be adapted through the process of evolution. In this case, the above representation of an equivalent GP function node for the signal-processing block of the *lag* function asks for some modification.

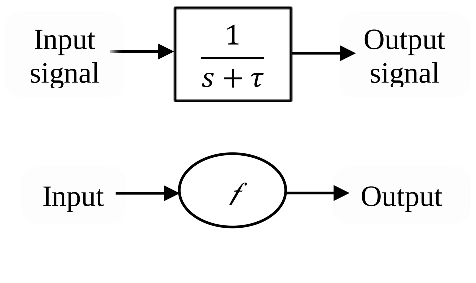


Fig. 1 Representation of signal processing block and GP function node

Two methods can be used to modify the structure of the tree. The first method is by using an implicit numeric constant terminal node in the tree structure to hold the value of $\tau$. This type of node is embedded (hidden) in each *lag* function node and does not appear explicitly in the tree structure. However, the values of this type of numeric constant nodes can be fetched and optimized using the parameter optimization methods. The second method, which is used in this paper, is by implementing an explicit numeric constant terminal node in the tree structure to hold the value of $\tau$. This type of node is visible in the tree structure and appears in the second argument of each *lag* function node. The *lag* function node, in this case, has two arguments instead of one, the first argument carries a time-domain signal node, and the second carries a numeric constant terminal node (which holds $\tau$ value). The function node that is equivalent to the *lag* function will be represented as shown in Fig. 2.

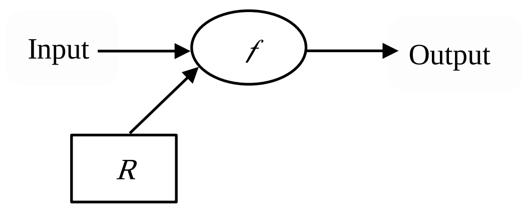


Fig. 2 Representation of the *lag* function by GP node function

To ensure a correct tree structure during the creation of initial population and genetic operations, the BDOGP applies syntactic rules to restrict the first argument of the *lag* function to be of signal node type, and the second argument to be of a numeric constant terminal node. Many signal-processing blocks may need one or more numerical parameters. These parameters are represented by numeric constant terminal nodes NCTNs [4] that are presented in some arguments of the corresponding function node. These NCTNs are not affected by the genetic operation, but their setting values can be optimized by numeric constant mutation operation used within BDOGP.

The signal-processing functions that may be used in the BDOGP are:

- *Invertor*

It is a one argument function used to negate the time-domain signal represented by its argument.

- *Differentiator*

It is a one argument function used to differentiate the time-domain signal represented by its argument.

That is, this function applies the transfer function *s*, where *s* is Laplace transform variable.

- *Integrator*

It is a one argument function used to integrate the time-domain signal represented by its argument. That is, this function applies the transfer function 1/*s*.

- *Lead term*

It is a two-argument function that applies the transfer function $(1 + \tau s)$, where $\tau$ is a real valued numerical parameter. The first argument is the time-domain input signal, while the second argument $\tau$ is a numerical parameter which represents the time constant of the lead.

- *Lag term*

It is a two-argument function that applies the transfer function, $(1 + \tau s)^{-1}$. The first argument is the time-domain input signal, while the second argument $\tau$ is the time constant.

- 2nd *order Lag term*

It is a three-argument function that applies the standard 2nd order ($\zeta$, $w_n$) transfer function, where, $\zeta$ is the damping ratio and $w_n$ is the natural frequency.

- *Add-Signal, Sub-Signal, Multi-Signal*

Each of these functions has two arguments. These functions perform addition, subtraction and multiplication respectively on the time-domain signals represented by their two arguments.

- *Add* 3 *Signals*

It is a three-argument function that adds the time-domain signals represented by its arguments.

- *Abs Signal*

It is a one-argument function that performs the absolute value function on the time-domain signal represented by its argument.

- *Gain*

It is a two-argument function that multiplies the time-domain signal represented by its first argument by a constant numerical value represented by its second argument.

The above signal-processing functions are suggested by the users of BDOGP; one may suggest different functions or special functions in the function set or may use other functions in the discrete time domain (z-domain). However, in general, the function and terminal nodes are divided into three categories. These are: the time-domain signal processing function nodes (e.g. lead, lag, etc.), the time-domain terminal nodes (e.g. input reference, output, etc.), and the numerical constant terminal nodes, which are carrying the parameters required in processing functions.

Some of these time-domain signal-processing functions are kind of dynamic functions. Therefore, in computing their outputs, the initial conditions of their outputs (or states) are required. Moreover, even if the initial conditions are assumed with certain values, the numerical methods need to save a certain number of previous output values (or previous states values) of each function node to calculate the subsequence output values through the iterations of the numerical calculations. In other words, the current and may be some previous values of the output of each dynamic function node are required to be saved in some locations of memory in order to use them in the next time step of the simulation.

To overcome this problem, Koza and others [11, 12] employed a block diagram software simulator for analysing and computing the fitness value of each candidate block diagram in the population. In case, when GP software is linked to auxiliary software simulators, like MATLAB, Spice, etc., the GP sends an individual to the simulator so that the fitness value to that individual is computed by the simulator, and then this value of fitness is sending back to the GP algorithm software.

This method has some disadvantages. It is difficult since it needs an expert in software linking. Furthermore, it needs well knowledge about the use of auxiliary software simulators, and it requires a mapping method to map the tree structure to the environment that auxiliary software simulator understands. Such auxiliary software simulators have a big disadvantage in that they will take a long time of calculation, because GP needs to communicate with the software simulator each time it needs to calculate the fitness function. So, they are time-consuming and slow the generations of the GP. Moreover, they need a very fast computer. Therefore, using these simulators for the design of a discrete controller is very unsuitable, especially for high-order plants.

In this paper, a new method is proposed for calculating the fitness value of each individual without the need to an auxiliary software simulator. In this method, each signal function node owns a certain amount of associated memory locations. Referring to the example of the lag function, the structure of the lag function node is modified as shown in Figure 3. The square *M* represents an implicit memory location associated to each function node of type lag function. This location saves the current output value of lag function. These arrangements can be done with high flexibility using object oriented languages such as C++ or alike. The resultant procedure for calculating the fitness value was found to be a very powerful and fast for small as well as large individual structure (block diagram).

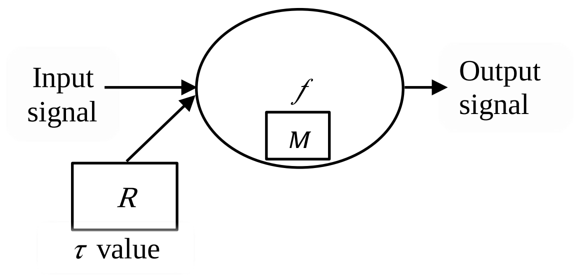


Fig. 3 The complete structure of dynamic *lag* function node

This method overcomes all difficulties mentioned before, and is so fast compared to the use of auxiliary software. In fact, all dynamic signal function nodes require some amount of memory locations, which depends on the function itself. Table 1 illustrates the type of arguments of each function node and the memory locations that are associated to each signal function node (where the symbol x in the table refers to nothing, and Y refers to yes).

## 2.2 initial population

The evolutionary process begins by creation of an initial population. There are a lot of methods for the creation of a random population [10], whatever creation method is used; the creation operation is restricted to start by choosing randomly a signal function node. The syntactic rules of construction are applied to each argument of the rooted signal function node. The attributes of the arguments for each signal function node are illustrated in Table 1.

The syntactic rules restrict the choice of arguments according to that established in Table 1. If the argument is of type numeric constant terminal node, then the rules will restrict the choice to be the numeric constant terminal node for that argument. However, if it is of a signal type, a signal function node or a signal, terminal node is chosen randomly for this argument. If a signal terminal is chosen, this branch will end at this depth, while if a signal function node is chosen, the procedure will continue in the same manner until the depth of the tree reaches the maximum allowable depth for creation. When the tree reaches the maximum depth minus one, the signal arguments are restricted to carry signal terminal nodes. This method of creation guarantees a correct structure for all individuals in the population.

Table 1 Detailed attributes of BDOGP signal function nodes

| Function node | No. of arg. | Type of arguments | | | Associated memory | |
|---|---|---|---|---|---|---|
| | | Arg. No.1 | Arg. No.2 | Arg. No.3 | 1 *M* | 2 *M* |
| Inverter | 1 | Signal | x | x | x | x |
| Differentiator | 1 | Signal | x | x | Y | x |
| Integrator | 1 | Signal | x | x | Y | x |
| Lead | 2 | Signal | Const. | x | Y | x |
| Lag | 2 | Signal | Const. | x | Y | x |
| Lag 2 | 3 | Signal | Const. | Const. | Y | Y |
| Add-signal | 2 | Signal | Signal | x | x | x |
| Sub-signal | 2 | Signal | Signal | x | x | x |
| Multi-signal | 2 | Signal | Signal | x | x | x |
| Add-3-signal | 3 | Signal | Signal | Signal | x | x |
| Abs-signal | 1 | Signal | x | x | x | x |
| Gain | 2 | Signal | Const. | x | x | x |

## 2.3 fitness computation

The signal function nodes of the BDOGP tree are divided into two types. The first type represents algebraic operations, where each function of this type performs a certain algebraic operation on its input signal represented by its arguments (e.g., Add-signal, Sub-signal, Gain, etc.). Therefore, it does not need initial conditions and memory locations to store the history of the output of the signal function node. The second type represents a transfer function in terms of Laplace's operator $s$ (or in terms of $z$ operator); in this case, the signal function node represents a dynamic relationship between its input signals and its output signal, so some memory locations are needed to save the history of the output of the signal function nodes. The number of locations depends on the dynamic element; in this work, either one or two locations are assigned.

The output of the signal function node that represents a continuous or impulse transfer function can be computed using one of the available numerical methods. The known Runge-Kutta method is used to simulate all types of transfer function. In each simulation step, each signal function node calls its external routine to perform its operation. The simulation will be stopped whenever an unstable response is detected. In general, the data that may be needed by these routines as inputs are:

a. The current value of the input signal to the signal function node.

b. The NCTN values that represent some arguments of this node (the parameters needed in the operation, like, time constant, damping ratio, etc.).

c. The current value and may be some previous values of the signal function node output (or states) that are stored in the associated memory locations.

The routine will return the new value of the signal function node output and store it in a memory location that belongs to the corresponding calling function node. Referring to the example of the lag function, Fig. 4 illustrates the operation of calculating the output of the lag function node in each time step of simulation.

The specific routine for the lag function takes the current input signal, the value of the $\tau$ parameter that is represented by the NCTN value, and the current value of the lag function node output that is stored in memory location $M$. The routine calculates the new value of the lag function node output at this simulation step and feeds it back to replace the old value that is stored in the memory location. The new output signal of the lag function node will represent an input signal to the next signal function node.

Before starting the fitness computation, the user must specify the initial conditions for each dynamic signal function node, the observation time $T_{ob}$ (eventually, the number $N$ for discrete computations) for the simulation, the time duration of the simulation step $H_s$, and the input signal. For each individual, the simulation starts by setting the initial condition values in the memory locations for each dynamic signal function node in the individual (normally zero initial conditions are used).

The computation of the block diagram output, for each simulation step, starts from terminal nodes and moves upwards to perform the operations on the subsequent signal function nodes. This process is continuing until the output value, which belongs to the root node, is found. This output represents the overall output of the individual (block diagram) at that time step.

In each iteration, each signal function node calls its routine to compute its output; therefore, the new output for each dynamic signal function is stored in the corresponding memory location and replaces the old node output. The subsequent iterations are done in the same way to the end of the observation time $T_{ob}$ (eventually, the number $N$ for discrete computations). In fact, in each time step increment during the simulation, each signal function node calculates itself separately from other nodes in the tree. Therefore, the simulation of each individual can be considered as a collection of sub-simulations of many signal-processing blocks that construct the block diagram.

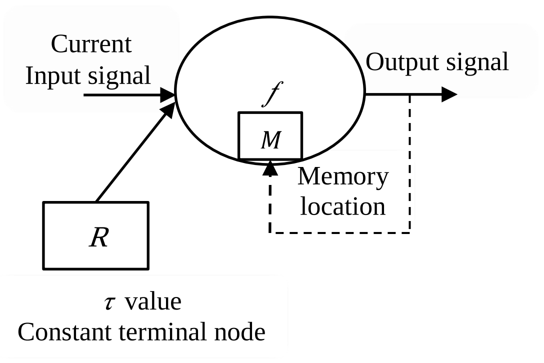


Fig. 4 The operation of calculating the output of the dynamic *lag* function node

## 2.4 genetic operations

Like any standard GP, the main genetic operations in BDOGP are crossover and mutation operations. The adopted and proposed genetic operations for BDOGP are described in what follows.

i) crossover operation: The crossover operation starts by selecting two parental individuals using any fitness based selection method. Then, using uniform probability distribution, one random point is selected in each parent to be the crossover point in that parent. All NCTNs are excluded from the selection as crossover points, using constrained syntactic rules; the two crossover fragments are exchanged to produce two offsprings. In this case, the rules guarantee that the produced offspring has the correct structure of a block diagram.

ii) swap mutation: All mutation operations are performed on a single parent. The operation of a swap mutation starts by selecting one-parent individual from the population based on the fitness. Then, using uniform probability distribution and avoiding the selection of NCTNs in the tree, a signal node is chosen randomly from the parental individual. If the chosen signal node is a terminal node, another signal terminal node is selected randomly from the terminal set to replace signal terminal node in the tree. If the chosen signal node is a signal function node, the number of signal arguments in this signal function node is checked. Then, another new signal function node, which has a similar number of signal arguments, is selected randomly from the signal function set. The signal function node in the tree and its only children of type numeric constant terminal node is deleted and replaced by the new selected signal function node from the function set. If the new signal function node has numeric constant arguments then, new NCTNs are created and inserted in these arguments. This operation is performed with the syntactic rules being respected in order to produce a correct offspring tree structure.

iii) shrink mutation: The shrink mutation operation starts by selecting randomly one signal function node from the parental tree. The selected signal function node and its arguments that represent NCTNs are deleted. Then, the argument that represents the input signal takes the place of the deleted parental signal function node. If the deleted function node has more than one argument, representing an input signal, in this case, one argument among them is randomly selected to replace its parent, and the other arguments of this type are deleted. This operation eventually produces an offspring with less depth than its parent.

iv) branch mutation: This operation starts by choosing randomly a signal node (function or terminal) from the parental tree, where all NCTNs are excluded from such selection. This chosen signal node and whatever below it are deleted and replaced by a new randomly created sub-tree. The creation operation of the sub-tree obeys the syntactic rules of construction, and it respects the overall allowable maximum depth of the offspring.

v) Inverse Shrink Mutation: The operation of inverse shrink mutation begins by excluding the NCTNs from being selected, and then a signal node (terminal or function) is chosen randomly from the parental tree. The sub-tree rooted at this node is stored in a certain place and deleted from the tree. A new signal function node is selected from the signal function set and inserted in the place of that deleted function node in the tree. The selection of a new function node is done with no regard to the number of arguments in the new selected signal function node. The stored sub-tree is inserted in the first argument that represents the input signal of the new selected function node. If the new signal function node has more than one argument, representing an input signal, the remaining arguments of this type are filled by randomly created sub-trees. Similar to branch mutation operations, the creation operation of those sub-trees obeys the syntactic rules of construction and the maximum allowable depth of the overall offspring tree. If the new function node also has arguments representing NCTNs, new NCTNs are created to fill those arguments. This operation eventually increases the depth of the overall tree.

## 2.4 numeric constant mutation operations

Generally, GP suffers a weakness in discovering useful numeric constants for terminal nodes of its program trees. This stems from the representation of the numeric constants as tree nodes, where the reproduction operations (including crossover and mutation) affect only the structure of the tree, and not, the composition of the nodes. Therefore, the individual numeric constants are not altered by the reproduction operations and thus cannot benefit from them. There are several techniques to eliminate such weakness [13, 14]. Numeric constant mutation is a technique for facilitating the creation of useful numeric constant node values during a GP run. Numeric constant mutation replaces some of the numeric constant node values with new ones for the individual to which it is applied.

In this paper, the proposed GP algorithm makes use of two modes of mutating the numeric constant values. Before the GP run, and based on the problem nature, a range and a resolution of the numeric constant node values are specified. In this case, the algorithm avoids the use of the default precision of the floating constant valid in the programming language. This gives the ability to ignore the least significant digits in the numeric constant values and gives more reliable values. The number of numeric constant nodes to be mutated in each individual is chosen randomly in the range [1, *n*], where n is some positive integer. For example, if $n = 3$, then this will mean that moving over the cost surface of the numeric constant in three dimensions will give a good probability to climb out the local minimum. After the determination of this number, these numeric constant nodes are chosen randomly in the tree, and each numeric constant node value will be mutated by either of the following two methods that are selected randomly with equal probability.

*Method* 1: The new numeric constant values are chosen randomly from a specific uniform distribution selecting range. The selecting range for each numeric constant is specified as the old value of the numeric constant plus or minus a specified percentage of the total allowed range.

*Method* 2: This method starts by selecting randomly the location of the digit to be mutated. The new numeric constant value is equal to the old value plus or minus one for selected digit. For instance, if the old value is 5.652 and the least significant digit is chosen as the digit to be mutated, then the new value of the constant will be either 5.653 or 5.651 with equal probability.

The first method lets the mutation operation explores all the search space and avoids falling in a local minimum. The second method is useful for fine tuning. These two methods are both used within the framework of the standard GP parameters optimizer to alter the numeric constant nodes. Among several other techniques like the gradient descent, the simulated annealing and genetic algorithm, the numeric constant mutation operation is properly and adequately functioning in changing the numeric constant values.

## 3 Deadbeat Controller

There are many methods to carry out the design of digital control for specified plant. However, the method of using the state space and state transition is preferable [15], because it represents a simple and systematic procedure for linear discrete control systems. The deadbeat response receives a large amount of works for both continuous and discrete control systems; the papers [16, 17, 18] are only a sample. In fact, this response is the ultimate of any design irrespective of the methodology used. Irrespective of the system order or type, ideally it is to obtain a zero overshoot and zero steady-state error for a certain reference input. However, the deadbeat response is often referred to a certain method of design in sampled-data control systems. The state transition approach insures that the deadbeat response does not have inter-sampling ripples.

It is known that the design of a controller that characterizes a deadbeat response requires a very accurate computation. For this reason only, it is selected to show how accurate design can be obtained by the GP algorithm. Therefore, a comparison between the state transition and the proposed BDOGP approaches to design a deadbeat response controller will be considered. In general, in order to design a digital controller, it is necessary to construct the sampled-data system for the continuous control system. A zero sample and hold device is assumed to be sufficient to obtain a sample data. Therefore, for the sake of completeness, next, the deadbeat response for a sampled-data theory will be briefly explained.

Consider the sampled-data *n*-order control system that is shown in Fig. 5.

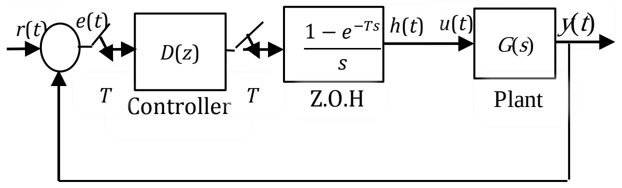


Fig. 5 A sampled-data control system

The signal $h(t)$ is the output of the zero-order-hold (Z.O.H), $e(t)$ is the error signal, and $r(t)$ is a reference input. Since the output of the digital controller is a train of impulses, its values at the sampling instants are equal to the outputs at zero order hold. Therefore, the pulse transfer function of the discrete controller is

$$D(z) = \frac{H(z)}{E(z)} = \frac{h(0) + h(T)z^{-1} + \cdots + h(mT)z^{-m}}{e(0) + e(T)z^{-1} + \cdots + e(mT)z^{-m}} \quad (1)$$

where, $H(z)$ and $E(z)$ are the z-transform of the Z.O.H output and actuating signal respectively.

The digital controller can be replaced by a variable gain $K(mT)$, where $T$ is the sampling time. It has different values during different sampling periods, thus

$$h(mT) = K(mT)e(mT), \quad m = 0,1,2 \ldots. \quad (2)$$

Next, for simplicity, $K(mT)$ is written as $K_m$. In order to realize a deadbeat response, the system error must be zero for $t \geq nT$, where $n$ is the smallest possible positive integer (order of the plant). This condition is realized if the following two conditions are satisfied [17]

$$x_1(nT) = r(nT) \quad (3)$$

$$x_1(nT) = r'(nT), \ \ldots. \ , x_n(nT) = r^{(n-1)}(nT) \quad (4)$$

where the variables $x_1, x_2 \ldots. x_n$ are the state variables of the plant; next, the plant state vector is denoted by $x$.

For step input, these conditions can be stated by the output vector, $x(nT) = [\beta\ 0\ \ldots 0]^T$. For nonzero-type linear systems, $\beta$ is an arbitrary real number.

The design procedure for deadbeat controller using the state transition approach can be accomplished as follows:

1. Use the state equations for the continuous portion of the system ($A$, $B$, $C$) to determine the discrete state equation as

$$x[(m+1)T] = Fx(mT) + Gu(mT) \qquad (5)$$

$$y(mT) = Hx(mT) \qquad (6)$$

where the discrete system matrices $F$, $G$ and $H$ are given by [19],

$$F = e^{AT} = \mathcal{L}^{-1}(sI - A)^{-1}|_{t=T} \qquad (7)$$

$$G = \int_0^T [\mathcal{L}^{-1}(sI - A)^{-1}] B d\tau \qquad (8)$$

The integral in equation 8 can be computed (exactly or approximately) as follows

$$G = \begin{cases} A^{-1}(F - I_n)B, & \det(A) \neq 0 \\ T(I_n + \frac{AT}{2!} + \frac{(AT)^2}{3!} + \frac{(AT)^3}{4!} + \cdots)B, & \det(A) = 0 \end{cases} \qquad (9)$$

$$H = C = [\beta \quad 0 \quad 0 \quad \cdots \quad 0] \qquad (10)$$

Since,

$$u(mT) = h(mT) = K_m e(mT)$$
$$= K_m(r(mT) - Hx(mT)) \qquad (11)$$

The discrete state equation becomes

$$x((m+1)T) = Fx(mT) + GK_m(r(mT) - Hx(mT)) \qquad (12)$$

2. Applying the deadbeat conditions to solve for the $n$ gains $K_m$. Then using the state equations to have the discrete values of $x_1(mT)$ and corresponding $e(mT)$; $m$ = 0, 1,…$n$. Clearly, for $m$ = 0, and $m$ = $n$, the error signal is equal to the input $r(0)$ and zero respectively.
3. The discrete values of $h(mT)$, $m$ = 0, 1…$n$ can be computed using expressions 2; eventually, $h(nT)$ = 0.
4. Finally, using equation 1, the discrete deadbeat transfer function $D(z)$ is determined.

## 4 Numerical Examples

In this section, we will introduce first a numerical example, which will be solved first by the conventional deadbeat procedure given above and then after by the proposed BDOGP algorithm. Next, a second example of a temperature control in an HVAC system is selected. It is a system of a zero-type one, where the achievement of a unity dc-gain represents an additional constraint.

Example 1:
For a third-order servo system, the open-loop transfer function is

$$G(s) = \frac{100}{s(0.25s+1)(0.025s+1)}$$

The task is to have a deadbeat unit-step response after 3 sampling periods, where $T$ = 0.15 seconds. Since the system is of type one, setting $\beta$ equal to 1 is adequate then specifically, to have

$$x_1(t \geq 0.45) = 1$$

$$x_2(t \geq 0.45) = x_3(t \geq 0.45) = 0$$

The discrete state equation is

$$\begin{bmatrix} x_1(m+1) \\ x_2(m+1) \\ x_3(m+1) \end{bmatrix} = \begin{bmatrix} 1 & 0.1226 & 0.0024 \\ 0 & 0.6095 & 0.0151 \\ 0 & -2.4282 & -0.058 \end{bmatrix} \begin{bmatrix} x_1(m) \\ x_2(m) \\ x_3(m) \end{bmatrix} + \begin{bmatrix} 2.749 \\ 39.048 \\ 242.814 \end{bmatrix} K_m[1 - x_1(m)]$$

The following three nonlinear algebraic equations will be obtained if the deadbeat is to occur after three sampling periods.

$$\begin{aligned} 11.2326K_0 + 8.14101K_1 + 2.7944K_2 & - 22.3389K_0K_1 - 22.3459K_0K_2 \\ & - 7.557K_1K_2 + 20.77491K_0K_1K_2 \\ & = 1 \end{aligned}$$

$$\begin{aligned} 15.0896K_0 + 27.48678K_1 + 39.048K_2 & - 75.55813K_0K_1 - 317.411K_0K_2 \\ & - 107.341K_1K_2 + 259.085K_0K_1K_2 \\ & = 0 \end{aligned}$$

$$\begin{aligned} -60.3962K_0 - 108.97K_1 + 242.8144K_2 & + 299.5185K_0K_1 \\ & - 1973.777K_0K_2 - 667.496K_1K_2 \\ & + 1834.9K_0K_1K_2 = 0 \end{aligned}$$

Clearly, although the above design procedure of the deadbeat controller is straightforward, the solution of the resultant system of nonlinear algebraic equations is the main drawback. For high-order systems, this could be a crucial problem. However, a combination of known numerical methods and evolutionary optimization tools relaxes this obstacle. In [20], Newton-Raphson and genetic algorithm are used to solve the above nonlinear algebraic system. That found solution is used here to check the accuracy of the solution proposed by the BDOGP algorithm. The results were

$$K_0 = 0.148276, K_1 = -0.13797, K_2 = 0.0102$$

Consequently, using these gain values and the discrete state equation, the discrete error and Z.O.H output signals are determined and hence the discrete controller.

$$D(z) = \frac{0.148276 - 0.08173z^{-1} + 0.000196z^{-2}}{1 + 0.5923z^{-1} + 0.0193z^{-2}}$$

An alternative evolutionary method is proposed here to evolve such controller, where the BDOGP is used to evolve the deadbeat controller in form of a block diagram. The terminal set is $T$ = {$e$ (k), $\mathcal{R}$}, while the signal function set which is described by transfer functions form is

$$F = \{z^{-1}, (1 + az^{-1}), (1 + az^{-1})^{-1}, (a + bz^{-1} + cz^{-2}), (a + bz^{-1} + cz^{-2})^{-1}, gain\}$$

The fitness function is

$$Fitness = \sum_{k=n}^{k=N} [e^2(k) + x_2^2(k) + x_3^2(k)]$$

where, $n$ = 3, and the control system is simulated for $N$ = 30 samples. This large number of samples (20 times the 3$T$ value) is used to ensure a high accuracy in computing the fitness function. To avoid overflow, which results from some evolved unstable controllers, simulation will be stopped whenever each state exceeds a maximum bound of ± $10^3$. The numeric constant terminal nodes $\mathcal{R}$'s are set in range -5 to 5 with resolution of $10^{-5}$.

Table 2 lists the main control parameters of the proposed BDOGP algorithm.

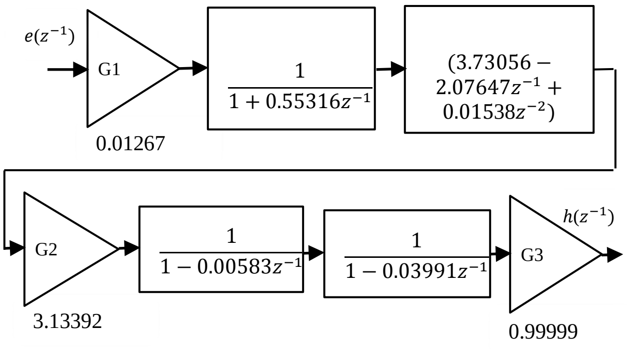
Fig. 6 shows the obtained block diagram of the evolved deadbeat controller in generation 8200, which has approximately $10^{-9}$ fitness. The transfer function of the discrete deadbeat controller evolved by the BDOGP is

$$D_{GP}(z) = \frac{0.148127 - 0.082449z^{-1} + 6.10684 \times 10^{-4}z^{-2}}{1 + 0.58769z^{-1} + 0.0188859z^{-2} - 1.18772 \times 10^{-4}z^{-3}}$$

Table 2 The main BDOGP control parameters

| | |
|---|---|
| Population size | 200 |
| Termination criterion | Stopping the generation manually |
| Creation probability | 20% |
| Creation type | Ramped-Half-and-Half |
| Crossover probability | 48% |
| Maximum depth for creation | 5 |
| Maximum depth for crossover | 8 |
| Selection type | Tournament selection |
| Tournament size | 4 |
| Swap mutation probability | 8% |
| Shrink mutation probability | 8% |
| Inverse shrink mutation probability | 8% |
| Branch mutation probability | 8% |
| Add best solutions to new generation | yes |
| Number of best solutions used in elitism | 50 |
| Type of numeric constant optimizer | Numeric constant mutation |
| Number of iterations applying in numeric constant optimizer | 10 iterations for each individual in the population, and additional 20 iterations for the elected individuals by elitism operation |

The transfer function of the BDOGP deadbeat controller has two zeros and three poles while the transfer function of state transition deadbeat controller has two zeros and two poles. However, this difference has an insignificant effect on increasing the time of computation required for each sample.



$e(z^{-1})$
G1
0.01267
$\frac{1}{1+0.55316z^{-1}}$
$(3.73056 - 2.07647z^{-1} + 0.01538z^{-2})$
G2
3.13392
$\frac{1}{1-0.00583z^{-1}}$
$\frac{1}{1-0.03991z^{-1}}$
G3
$h(z^{-1})$
0.99999

Fig. 6 Evolved discrete controller block diagram

In order to compare the behaviours of both BDOGP and state transition deadbeat controllers, table 3 illustrates the values of the control action, and the system states from $m$ = 1 to $m$ = 5 for both controllers. The results show that the response of the BDOGP deadbeat controller is very accurate, where the output reaches the value of input signal at 3 samples with zero overshoot and zero steady-state error. Moreover, $x_2(m)$ and $x_3(m)$ reach nearly zero values at sample 3, and they settle with accuracy up to $10^{-5}$. The classical deadbeat controller seems a little bit less accurate than the BDOGP controller. The reasons belong to the rounding error accumulated through the computations as well as the numerical values of $K_m$'s.

Fig. 7 shows both discrete and sampled-data responses; the responses closeness to each other indicates the accuracy of the GP algorithm. Furthermore, table 4 gives all margins of both sampled-data systems; obviously, both systems are stable.

Example 2:
Temperature control of heating, ventilation, air-conditioning system is a vital problem in most institutional buildings, hospital, warehouse, etc. The control system consists of two subsystems, one for heating and ventilation and the other for air conditioning. Both subsystems control the air temperature and the air humidity. However, for primarily design, separate design of each subsystem is carried out. In this work, only the temperature control model will be considered.

Table 3 Conventional controller and BDOGP controller

| $m$ | Contr. | $u(m)$ | $x_1(m)$ | $x_2(m)$ | $x_3(m)$ |
|---|---|---|---|---|---|
| 1 | $D$(z) | 1.48275 e-01 | 0.407608 | 5.78984 | 36.0032 |
| | $D_{GP}$ | 1.48127 e-01 | 0.407201 | 5.78406 | 35.9673 |
| 2 | $D$(z) | -8.1764 e-02 | 0.979212 | 0.88169 | -35.989 |
| | $D_{GP}$ | -8.16923 e-02 | 0.978077 | 0.87857 | -35.967 |
| 3 | $D$(z) | 4.01007 e-04 | 1.00204 | 9.61992 e-03 | 0.437914 e-02 |
| | $D_{GP}$ | 1.94625 e-04 | 1 | -1.345 e-05 | -6.01934 e-07 |
| 4 | $D$(z) | -5.4533 e-04 | 1.00182 | -1.47695 e-02 | -1.58313 e-01 |
| | $D_{GP}$ | -4.71856 e-08 | 1 | -1.00494 e-05 | 2.12368 e-05 |
| 5 | $D$(z) | 2.15594 e-4 | 1.00022 | -2.974 e-03 | 9.73946 e-02 |
| | $D_{GP}$ | 7.04361 e-08 | 1 | -3.05402 e-06 | 4.0273 e-05 |

Table 4 All margins of transfer function $D(z)\ G_p(z)$

| Closed-loop Parameters | With Theoretical Controller | With GP Controller |
|---|---|---|
| Gain Margin (dB) | 2.7917 | 2.7872 |
| GM Frequency (rad./sec.) | 12.8162 | 12.8187 |
| Phase Margin (degree) | 63.6774 | 63.8590 |
| PM Frequency (rad./sec.) | 4.1956 | 4.1852 |
| Delay Margin (dB) | 1.7660 | 1.7754 |
| DM Frequency (rad./sec.) | 4.1956 | 4.1852 |

The task is to design a discrete controller for the temperature control in an HVAC system [21, 22]; in the appendix, some detailed information of the standard model is given. For a half-range air flow rate ($f_s$ = 8.33 m$^3$/min), the transfer function is

$$G(s) = \frac{0.64}{18s+1}\, e^{-2.4\,s}$$

The delay term $e^{-2.4s}$ is approximated by second-order lag and thus the overall transfer function becomes

$$G(s) \approx \frac{0.64}{(18s+1)(1+2.4s+0.5(2.4)^2s^2)}$$

where $u(t)$ is the control signal, the controller output, and the output is the indoor temperature.

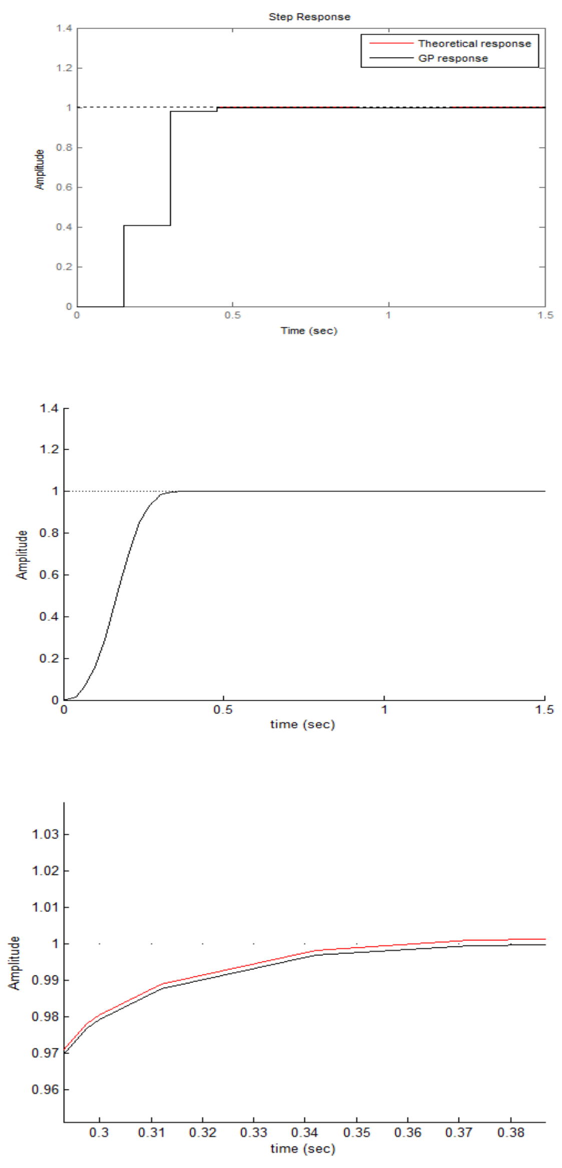


**Fig. 7** Discrete and continuous sampled-data responses (example 1); a zooming figure is also depicted

The error signal, the input of the controller, $e(t)$ is equal to the difference between the indoor temperature $y(t)$ and the set point temperature $r(t)$ (a step input). For a sampling time $T$ equal to 5 minutes, the discrete matrices are

$$F=\begin{bmatrix}0.8748 & 2.2469 & 2.3833\\ -0.0460 & -0.0627 & 0.1282\\ -0.0025 & -0.0964 & -0.1767\end{bmatrix}, G=\begin{bmatrix}0.0800\\0.0294\\0.0016\end{bmatrix}$$

For this zero-type system, in usual state-space approach, the normal conditions (equations 3 and 4) of the deadbeat step response will be satisfied, but a non-zero steady state will be resulted. However, only for a certain value of an output magnitude $\beta$, it is possible to have a deadbeat step response with zero steady-state response. This is really because the non-unity dc-gain of the closed-loop control system is changed for every desired value of the $\beta$ output, i.e. since different deadbeat controllers are obtained for different output magnitudes.

Analytically, for an output of a magnitude $\beta$, to ensure a unity closed-loop dc-gain and hence a zero steady-state error for unit-step input, it must be hold

$$\beta-\lim_{z\to 1}\frac{D(z)G_p(z)}{1+D(z)G_p(z)}=0 \qquad (13)$$

where $Gp(z)$ is the transfer function of the Z.O.H and plant, and $D(z)$, is the deadbeat controller that is designed for unit-step reference input and an output of $\beta$ magnitude.

This will complicate the state-space approach to obtain simultaneously the gains $K_m$ and the value of $\beta$. Instead, some simulation runs are necessary to find that a certain value of $\beta$. Only after that, the closed-loop controlled system is scaled by an outer feedforward gain equal to 1 / $\beta$ to obtain a zero steady-state response for all step input magnitudes. This is an essential demand because the temperature degree has to be changed during the operation hours.

For the proposed GP algorithm, a small modification is required in the individual tree to include the $\beta$ parameter that contributes in minimizing the fitness function. The same terminal node, transfer function set, fitness function with $N$ = 50, and GP parameters (table 2) are used as in example 1. Initially, a value of 1 is set for $\beta$. At the end of the computation of the fitness function, all solutions that do not satisfy the condition 13 will be punished to exclude them.

To avoid a large number of generations and hence a long time of computation, an accuracy of $10^{-6}$ is selected, and the GP generation is stopped manually. The discrete transfer function of the obtained deadbeat controller evolved by the proposed BDOGP is

$$D(z)=\frac{3.697(1+0.3646z^{-1}+0.2878z^{-2})}{(1+0.7041z^{-1}+0.3751z^{-2})}$$

Since the two poles are inside the unit circle, then the deadbeat controller is stable. The magnitude of the reference step input $\beta$ is found to be 0.6526. Therefore, to obtain zero steady-state response, an outer feedforward gain of 1.5323 is added.

Fig. 8 shows the unit-step response for an hour, where as it can be seen, after 15 minutes ($3T$) the response satisfies the deadbeat response condition. Due to the numerical accuracy, some small ripples (maximum is about 4%) appear for time greater than 3T. However, the steady-state behaviour is quite enough recovered at a time less than half an hour. Furthermore, Fig. 9 shows a demand for a temperature control profile. It is assumed that the system was in operation over the time interval (0-80) minutes, and the room temperature attains a steady value of 28 degrees centigrade. Then it is demanded for achieving 24, 20, and 16 degrees centigrade after 80, 120 and 200 minutes respectively. As shown, for all temperature demands, the controlled system is regulated within 30 minutes to more than 99 % of the demands.

It is worth mention that one may argue about decreasing the settling time by reducing the sampling time without affecting the stability. However, in this example, the task was only to demonstrate the use of the proposed BDOGP to synthesis a discrete controller, and hence further investigation could give better results.

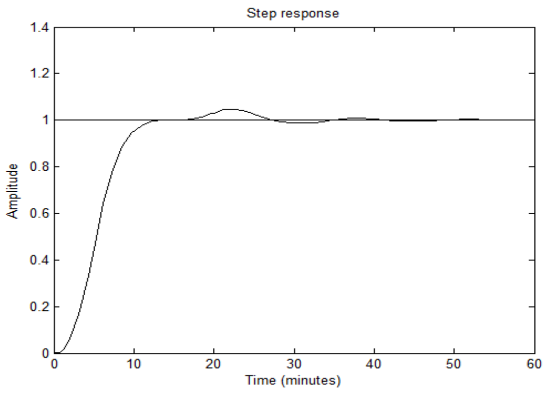


Fig. 8 Unit-step response (example 2)

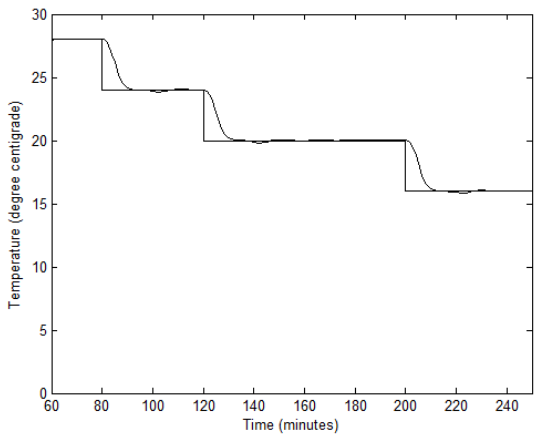


Fig. 9 Temperature control performance (example 2)

## 5 Conclusion

A synthesis of a discrete controller that is based on a proposed genetic programming algorithm has been presented. The results demonstrate that the BDOGP algorithm, which is assessed by the numeric constant mutation operation, can be used to make decisions concerning the total number of signal processing blocks to be employed in the deadbeat controller, the type of each block, and the values of all parameters for all blocks. The effectiveness of the proposed BDOGP algorithm is shown through simulation of linear SISO plants. The numerical results of the first example indicate that the response of the GP algorithm is quite similar to the theoretical deadbeat technique. The proposed BDOGP is easily adapted for the design of a discrete controller with zero steady-state error for a zero-type temperature control system. The authors are working on considering higher-order linear and nonlinear industrial SISO systems in the near future, and the digital implementation of the algorithm for specific system. Furthermore, a work will be done on reducing the long computation time that is the big disadvantage of the method.

*References:*

[1] Suhail Owais, et al ,"Data Mining by Symbolic Fuzzy Classifiers and Genetic Programming– State of the Art and Prospective Approaches", *WSEAS Transactions on computers*, Issue 3, Vol. 12, March 2013

[2] Gabriela Prelipcean, Mircea Boscouanu, Nicolae Popoviciu, " The Role of Predictability of Financial Series in Emerging Market Applications", *WSEAS Transactions on Mathematics*, Issue 1, Vol. 7, January 2008

[3] S. Asha, R. Rani Hemamalini, "Synthesis of Adder Circuit using Cartesian Genetic programming", *WSEAS Transactions on Circuits and Systems*, Vol. 14, 2015

[4] Mohamed J. Mohamed, "A proposed Genetic Programming Applied to Controller Design and System Identification", *Unpublished Ph.D. Thesis*, University of Technology, Baghdad, Iraq, February 2008

[5] Rami A. Maher, Mohamed J. Mohamed, "An Enhanced Genetic Programming Algorithm for Optimal Controller Design", *Intelligent Control and Automation*, 2013, 4, 94-101

[6] Rami A. Maher, " *Optimal Control Engineering with MATLAB*", Chapter 8, Nova Science, 2013

[7] Sekaj I., Perkacz J, "Genetic Programming-based Controller Design",. *Evolutionary Computation*, Ieee Congress on, CEC 2007

[8] Gray G. J., et al "Structural System Identification Using Genetic Programming and a Bloack Diagram Oriented Simulation Tool", *Electronics Letters*, Vol. 32 issue 15, 1996

[9] J. R. Koza , M. A. Keane, J. YU, F. H. Bennett III, and W. Mydlowec, "Automatic Creation of Human-Competitive Programs and Controllers by Means of Genetic Programming", *Genetic Programming and Evolvable Machines*, 1 , 121-164, 2000

[10] Eva Brucherscifer, et al, "An Indirect Block-Oriented Representation for Genetic Programming", *Proceeding EuroGP 01 proceeding of the 4th European Conference on Genetic Programming*, Springer-Verlag London 2001

[11] J. R. Koza, M. A. Keane, F. H. Bennett III, J. Yu, W. Mydlowec, and O. Stiffelman, "Automatic Creation of Both the Topology and Parameters for a Robust Controller by Means of Genetic Programming", *Proceeding of the 1999 IEEE. International Symposium on Intelligent Control/Intelligent Systems and Semiotics*, Cambridge, MA, September 15-17, 1999

[12] M. A. Kean, J. R. Koza, and M. J. Streeter, "Automatic Synthesis Using Genetic Programming of an Improved General-Purpose Controller for Industrially Representative Plants", *IEEE, Proceeding of the NAS/DOD Conference on Evolvable Hardware*, 2002

[13] J. R. Koza, "*Genetic Programming: On the Programming of Computers by Means of Natural Selection*", 1992, Cambridge, MA: The MIT press

[14] M. Evett, T. Fernandez, "Numeric Mutation Improves the Discovery of Numeric Constants in Genetic Programming *", Proceeding of the Third Annual Genetic Programming Conference*, Wisconsin pp. 66-71, 1998

[15] Kuo, Benjamin C. "*Analysis and Synthesis of Sampled-Data Control Systems*", Prentice Hall 1963

[16] Robert Paz, Hatem Elaydi, "Optimal ripple-free deadbeat response", *Int. J. Control*, Vol. 71, No. 6, 1998

[17] Dane Baang, Dongkyoung Chwa, "Deadbeat Control for Linear Systems with Input Constraints", *IEICE Transactions on Fundamentals of Electronics, Communications and Computer Sciences*, Vol. E 92 A (2009) No. 12, P3390-3393

[18] Constantine A. Karybakas, Constantine A. Barbargires, "Explicit Conditions for Ripple-free Dead-Beat Control", *Kybernetica* Vol. 32, No. 6, 1996

[19] Vladimir Strejc, "*State Space Theory of Discrete Linear Control*", Academia Prague, 1981

[20] Ahmed Salih, "Optimal Control Using Genetic Algorithm", *unpublished M.Sc. Thesis*, MCE, Baghdad, Iraq, 2002

[21] Yamakawa Y., et al, "Air-Conditioning PID Control System with Adjusted Reset to offset Thermal Loads Upsets", *Oyama National College of Technology, Univ. of Tokyo*, pp. 209, 2010

[22] Bothaina Ali Mahasneh, "Development of Control and Management Algorithm for Electromechanical Subsystems of Intelligent Building Management System in Institutional Buildings" *unpublished M.Sc. thesis*, Isra University, 2012

*Appendix*:

One degree of freedom controllers are denoted as a first-order plus dead time (FOPDT) model. The most commonly used approximate model for the indoor temperature control in an HVAC system is given by the FOPDT model as

$$G(s) = \frac{K_p}{T_p s + 1} e^{-L_p s}$$

where

- $K_p$ is the forward gain given in terms of the system parameters by
$$K_p = \frac{\theta_s}{w_s + \alpha} = \frac{\theta_s}{c_p \rho_a f_s + \alpha}$$
- $T_p$ is the system time constant given in terms of the system parameters by
$$T_p = \frac{C}{w_s + \alpha} = \frac{C}{c_p \rho_a f_s + \alpha}$$
- $L_p$ is the system time delay of the system given in terms of the system parameters by
$$L_p = \frac{L_{p0}}{w_s + \alpha} = \frac{L_{p0}}{c_p \rho_a f_s + \alpha}$$
- $\theta_s$ is the supply air temperature in the cooling coil; it is equal 13.1 degree centigrade.
- $C$ is the overall heat capacity of the air-conditioned space; it is equal 370.44 KJ/Kelvin.
- $c_p$ is the specific heat of air at the sea level, dry and zero 1 degree centigrade; it is equal to 1.0035 KJ/m$^3$ Kelvin.
- $f_s$ is the supply air flow rate; it is in the range (0 -16) m$^3$/min.
- $\rho_a$ is the air density at 15 degree centigrade; it is equal to 1.225 Kg/m$^2$.
- $\alpha$ is the overall transmittance-area factor; it is equal to 9.69 KJ/min Kelvin.
- $L_{p0}$ is the initial heat capacity of air-conditioned space, and it is equal to 49.4 KJ/Kelvin.