RELAY FEEDBACK IDENTIFICATION USING GUIDING EVOLUTIONARY ALGORITHM

Adrian Saldanha, Milan Hofreiter

Czech Technical University in Prague, Faculty of Mechanical Engineering

Abstract: The Guiding Evolutionary Algorithm (GEA) was proposed by Cao, Xu and Goodman in 2016. The original algorithm was designed to apply the advantages of the previously published Particle Swarm Optimization (PSO), Genetic Algorithm (GA) and the Bat Algorithm (BA). As part of this work, the effectiveness of the GEA algorithm in solving multi-modal optimization problems is analyzed and the results are compared to those obtained using PSO and BA. Once the algorithm is verified and accepted, the same is then applied to a System Identification task using a relay-based feedback method and finally, upon successful identification of a dynamic system, the system is controlled using a PID controller. The work investigates a non-linear approach to identification of dynamic systems using simple Second Order Plus Time-Delay Models (SOPTD).

Keywords: Guiding Evolutionary Algorithm, Relay-feedback identification, Particle Swarm Optimization, Bat Algorithm, Optimization, PID Control

1 Introduction

Since the late 1990’s and even more recently in the 2000’s, there has been an extensive study of different algorithms for solving optimization problems. The ‘No-Free-Lunch’ Theorems for Optimization states that there is no universal algorithm for all problems, which means that there is always scope for improvement in the field of optimization [1]. In the book ‘Nature-Inspired Optimization Algorithms’ [2] Dr. Yang explores a whole new study of metaheuristics using nature inspired techniques. The bat algorithm as well as the Cuckoo-Search algorithm were known to produce promising results compared with those of its predecessors such as Particle Swarm Optimization or Genetic Algorithms. However, it was seen that while BA is a powerful algorithm in exploitation, it may sometimes end up being trapped into some local optima which makes it harder to use for global optimization algorithms [3]. The GEA was introduced to retail some advantages of each algorithm while avoiding some of the disadvantages [4]. With the help of the software MATLAB / Simulink, we aim to visualize the optimization mechanics of the three algorithms, namely GEA, PSO and BA and test and compare the efficiency of each for six pre-defined functions of varying complexity. We shall later test the same for a control application involving system identification and PID control.

The article is arranged as follows: Section 1 provides an introduction to the article as a whole. Section (2) describes optimization approach using metaheuristic algorithms and then explains the three algorithms (GEA, PSO and BA) in brief after which, the results of the optimization are compared for a number of multi-modal functions. Section (3) introduces the concept of relay-based identification of dynamic systems and uses the GEA algorithm for parametric identification of the dynamic process following which, in section (4) a number of PID control algorithms are looked into for appropriate control of the processes using the identified dynamic model of the same. Finally, in section (6), the results of the optimization, identification and control are evaluated for a real dynamic system.

2 Optimization

The two main categories in optimization are called ***Deterministic*** and ***Stochastic*** based optimization algorithms. Deterministic algorithms follow a definite procedure for arriving at a solution. Newton's gradient-based algorithm for finding the maxima or minima of a function in finite space, also known as the ***'Hill-climbing algorithm'*** is one such approach for finding the optima. While deterministic algorithms tend to always find a solution if it does exist, conventional gradient-based approaches are not quite feasible for large-scale problems in which the search space is larger or when the dimensions are much higher and due to this, we need to add a bit of randomness while searching for a solution. With Stochastic algorithms, we use the principle of probability wherein, we use randomization techniques in a well-defined and progressive manner so as to search for the optima in various different sections of the design space. The drawback of this method is that while these algorithms converge very quickly, they may tend to get stuck in a local optimum value instead of finding the true global minimum. That is, we cannot guarantee that we shall arrive at the precise solution, but rather, we look for the best solution which meets our pre-defined criteria which comes under the branch of ***Heuristics***. The term '***Metaheuristics'*** is a sub-branch of heuristics, but Metaheuristics are inherently problem independent and can be used for a wide range of applications.

As part of this article, the role of Nature-Inspired Metaheuristic algorithms in solving multi-modal problems is looked into. The mechanics governing these algorithms is closely related to natural occurring phenomena and is loosely modelled around natural species like swarm behavior (Particle Swarm Optimization), echolocation of bats (Bat Algorithm), survival of the fittest (Genetic Algorithm). Among all these algorithms, there exist a few main patterns which are similar and are necessary for efficient convergence to the optimum value. These three operators are as follows:

1. ***Crossover:*** Provides good mixing within the solution space
2. ***Mutation:*** Provides essential exploratory framework for ***diversification*** of the solutions to avoid settling at a local optimum solution.
3. ***Selection:*** Provides the necessary exploitative framework for ***intensification*** of the search.

Each algorithm essentially uses each of the above operators, but in different fashion. The manner in which it is carried out, thus determines the efficacy of the method. While it is clear that there is no best method in general, the effectiveness is largely dependent on the problem at hand.

To apply the above methods for our problem set, we shall have to use the following terms for building our solution:

1. ***Dimensions (d):*** The dimensions is the number of variables which influence the objective functions or, in short, this can be given as the number of search parameters to be optimized.
2. ***Random variables***: These are variables whose values are randomly determined by a set of random distribution rules such as Gaussian distributions, Uniform distributions, Levy Distributions. For the purpose of the algorithms in this article, we use Gaussian distributed and uniformly distributed random variables.
3. ***Random Walk***: A random walk is essentially a step taken randomly from a fixed point. The final value depends on a number of randomly taken steps from the initial position.
4. ***Solutions*** $(\boldsymbol{X})$: The solutions is a vector containing a number of points in a d-dimensional search space. In some places, this is called ***individuals*** especially in relation to nature-inspired metaheuristics, and in some others, like in PSO, these are known as ***particles***. Solutions or individuals can be mathematically represented as:

$$\boldsymbol{x_N = \sum_{i=1}^{N} x_i} \tag{1}$$

Where,

$x_i = [x_{11}, x_{12} \dots x_{1d}]$, d = number of dimensions of the search space

5. ***Cost function*** $(\boldsymbol{f(x)})$***:*** The cost function of any optimization problem is the most important as it describes the problem or the function which needs to be optimized.

With the above definitions, we shall proceed to describe each of the methods.

2.1 Particle Swarm Optimization

Particle Swarm Optimization (PSO) is a meta-heuristic based optimization algorithm which was developed by modelling swarm behavior observed in nature such as bird flocking, fish schooling, swarming theory [5]. The idea behind this technique is based on the principle that each individual in the subsequent generations can benefit from the observations by the other individuals from the previous generations, thus improving the subsequent results.

As described by the original algorithm, with each iteration (generation) each individual particle is attracted toward the position of the current global best $\boldsymbol{g}^*$ as well as its own best location $\boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{t}}$ in history. The positions and velocities of the particles namely, $\boldsymbol{x}_{\boldsymbol{i}}$ and $\boldsymbol{v}_{\boldsymbol{i}}$ respectively are updated as follows:

$$\boldsymbol{v}_i^{t+1} = \theta \boldsymbol{v}_i^t + \alpha \boldsymbol{\epsilon}_{\boldsymbol{1}}[\boldsymbol{g}^* - \boldsymbol{x}_i^t] + \beta \boldsymbol{\epsilon}_2[x_i^* - \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{t}}] \tag{2}$$

$$\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_i^t + \boldsymbol{v}_i^{t+1} \tag{3}$$

The parameters α and β are learning parameters and the values of these can be adjusted based on the problem at hand and depending on how quickly we want the solution to converge. The parameter θ is the inertia weight and while it is not absolutely necessary, it helps to ensure that the particles do not jump around the optimum value. Finally, at the end of every generation, the global best position as well as the individual best is updated using the cost function as follows:

$$\begin{gathered} g^* = \min\{f(x_i)\} \\ x_i^* = x_i, \text{ if } f(x_i) < f(x_i^*) \end{gathered} \tag{4}$$

The Pseudo-code of the above algorithm can be given as follows [2]:

```
Particle Swarm Optimization
1. Objective function $f(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_d)^T$
2. Initialize locations $x_i$ and velocity $v_i$ of n particles
3. Find $\boldsymbol{g}^*$ from $\min\{f(\boldsymbol{x}_1), f(x_2) \ldots f(x_n)\}$ (at t = 0)
4. While (criterion)
       For (loop over n-particles and d-dimensions
           Generate new velocity $\boldsymbol{v}_i^{t+1}$
           Calculate new locations $\boldsymbol{x}_i^{t+1} = \boldsymbol{x}_{\boldsymbol{i}}^{\boldsymbol{t}} + \boldsymbol{v}_i^{t+1}$
           Evaluate objective function at locations $x_i^{t+1}$
           Find the current best for each particle $\boldsymbol{x}_i^*$
       End for
       Find the current global best $\boldsymbol{g}^*$
       Update $t = t + 1$ (iteration counter)
5. End while
6. Output final results $\boldsymbol{x}_{\boldsymbol{i}}^*$ and $\boldsymbol{g}^*$.
```

Figure 1- Pseudo Code - Particle Swarm Optimization

2.2 Bat Algorithm

The inherent principle of BA is quite similar to PSO wherein the BA also uses swarm intelligence for finding the optima, however, in case of BA, the method used is frequency tuning which, is based on the echolocation principle of bats. In essence, PSO is a specific case of BA upon appropriate parameter setting. In case of BA, the position and velocities are updated using the frequency *f* as follows:

$$f_i = f_{min} + (f_{max} - f_{min})\beta \tag{5}$$

Where,

f_i: frequency of ith particle

f_{max}, f_{min}: max and min search frequency ($f_{min} = 0, f_{max} = 2$)

$$v_i^{t+1} = v_i^t + f_i.(x_i^t - x^*) \tag{6}$$

$$x_i^{t+1} = x_i^t + v_i^{t+1} \tag{7}$$

Once again, the global best position $\boldsymbol{x}^*$ is updated at the end of each iteration similar to PSO. In addition to the above, the BA also provides a necessary mechanism for exploitation using local search as:

$$x_{new} = x_{old} + \sigma\epsilon_t A^t, \tag{8}$$

Where,

σ: Scaling factor (= 0.1 x search range for each dimension)

The intensity of the local search is supposed to increase as the solution moves closer towards the global minimum which is therefore more as the algorithm progresses as compared to the beginning stages. This can be regulated by increasing the *pulse emission rates* as the number of generations rise. Additionally, as the bat moves closer to its prey, the *loudness* decreases, which means that the algorithm moves from exploratory mode to exploitative mode. This can be represented by:

$$A_i^{t+1} = \alpha A_i^t \tag{9}$$

$$r_i^{t+1} = r_i^0[1 - e^{-\gamma t}] \tag{10}$$

The above equations and search procedure can be seen in the below pseudo-code.

```
Bat Algorithm
1. Objective function $J(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \ldots, x_d)^T$
2. Initialize locations $x_i$ and velocity $v_i$ of n particles, ($i = 1,2 \ldots n$)
3. Initialize frequencies $f_i$, pulse rates $r_i$ and loudness $A_i$
4. Find $\boldsymbol{g}^*$ from min{$J(\boldsymbol{x}_1), J(\boldsymbol{x}_2) \ldots J(\boldsymbol{x}_n)$} (at t = 0)
5. While (t<max number of iterations)

   Generate new solutions by adjusting frequency,
   Update velocities and locations as per Bat Algorithm
   If (rand > $r_i$)
         Select a solution among the best
         Generate a local solution around the best
   End if
   Generate new solution by flying randomly
   If (rand < $A_i$ & f($\mathbf{x}_i$) < f($\mathbf{x}^*$))
         Accept new solutions
         Increase $r_i$ and reduce $A_i$
   End if
   Rank the bats and find current best $\mathbf{x}^*$
6. End while
```

Figure 2 - Pseudo Code - Bat Algorithm

2.3 Guiding Evolutionary Algorithm (GEA)

The Guiding Evolutionary Algorithm, while similar to the above two described algorithms, is essentially simpler to use due to the fewer number of parameters required for tuning the optimization. Like the above, the algorithm consists of Crossover, mutation and local search which can be described by the below equations much simpler as compared to the previous ones.

1. Crossover: The crossover for GEA is given by:

$$x_i^t = x_i^{t-1} + (x_*^{t-1} - x_i^{t-1}) * \beta \qquad (11)$$

Where,

β: Step length of position increment, uniformly distributed r.v

The step length β defines the rate of convergence and is generally between 1 and 2. A higher value represents faster convergence.

2. Mutation: The mutation provides the required exploratory mechanics for optimization. It can be given by the equation:

$$x_i^t = x_i^t + \epsilon M \qquad (12)$$

Where,

ϵ: Uniform r.v [-1, 1]

M: Mutation vector,

$M_j = (\max(x_{ij}^t - a, b - x_{ij}^t)$; [a, b] = range of jth dimension

According to GEA, the probability of mutation is given by the following:

$$p = c * \ln\left(\frac{T_{max}}{T_{max}-t}\right) \quad (13)$$

Where,
T_{max}: max number of generations
t: current generation
c: 0.2 (constant)

The above equation shows that the probability of mutation increases as the generations pass, thus it helps ensuring that the solution does not settle around the local maxima / minima.

3. Local: As before, the local search provides the necessary exploitative dynamics and can be given as follows:

$$x_i^t = x_*^{t-1} + \epsilon L \quad (14)$$

Where,
L: Local search vector
$L_j = 0.1(b-a)$; [a, b] = range of jth dimension

Similar to the mutation vector, the local search functions similar to the mutation except that it serves to find a solution around the current best unlike mutation, which helps to find a new solution around the unsearched territory.

Again, the probability of local search is given by p defined by equation (13).

GEA Algorithm

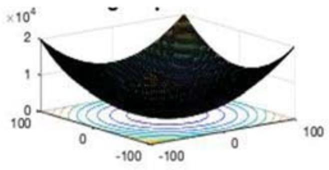
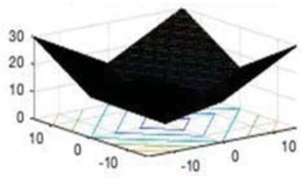
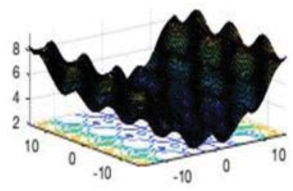
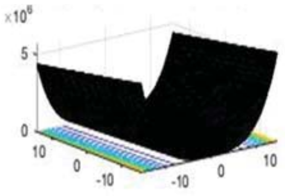
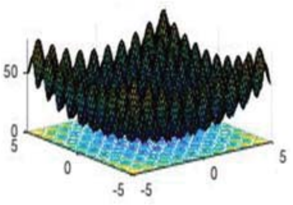
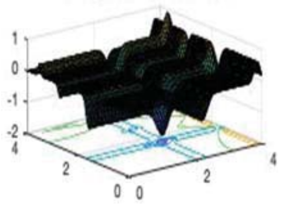
```
1. Objective function $J(\boldsymbol{x})$, $\boldsymbol{x} = (x_1, \dots, x_d)^T$
2. Initialize locations $x_i$, define parameters c, M, L
3. Evaluate the initialized positions
4. Select the best individual $x_*$
5. While (t<max number of iterations)
   Update p;
   For each individual:
   Make crossover to generate a new individual $x_i^t$
   If (rand <p)
        Make mutation for $x_i^t$
   End if
   If (rand <p)
        Make local search for $x_i^t$
   End if
   If (f($\mathbf{x}_i$) < f($\mathbf{x}^*$))
        Accept new solutions
   End if
   Find current best $\mathbf{x}^*$
6. End while
```

Figure 3 - Pseudocode - GEA Algorithm

2.4 Test functions

To evaluate the above methods, we shall test the above algorithms on six test functions as below and verify how quickly each tends to arrive at the real global optimum value.

Table 1 - Test Functions

Functions	Function Name	Expression	Domain
F1	De-Jong's Sphere function 	$f(x) = \sum_{i=1}^{D} x_i^2$	[-100, 100]
F2	Schwefel 2.2 function 	$f(x) = \sum_{i=1}^{D} \lvert x_i \rvert + \prod_{i=1}^{D} \lvert x_i \rvert$	[-15, 15]
F3	Griewangk's function 	$f(x) = -\prod_{i=1}^{D} \cos\left(\frac{x_i}{\sqrt{i}}\right) + \sum_{i=1}^{D} \frac{x_i^2}{4000} + 1$	[-15, 15]
F4	Rosenbrock's function 	$f(x) = \sum_{i=1}^{D-1} 100 * (x_{i+1} - x_i^2)^2 + (x_i - 1)^2$	[-15, 15]
F5	Rastrigin's function 	$f(x) = D * 10 + \sum_{i=1}^{D} (x_i^2 - 10 * \cos(2\pi x_i))$	[-5, 5]
F6	Michalewicz function 	$f(x) = -\left\{\sin(x)\left[\sin\left(\frac{x^2}{\pi}\right)\right]^{2m} + \sin(y)\left[\sin\left(\frac{2y^2}{\pi}\right)\right]^{2m}\right\}$	[0, 4]

For all of the above functions, our goal is to evaluate the point or points at which, the value of the above functions is at a minimum. We can visualize the above test functions as per the below fig. 4. For the purpose of visualization, we contain the number of dimensions to three at max since, for greater dimensions, we are unable to see the working mechanism of each in detail.

2.5 Parameterization

Before simulating the above functions, we first need to set appropriate parameters to ensure that our algorithm works in an efficient manner. The parameters for each algorithm are taken from literature. For PSO, we consider the parameters as follows [4]:

Parameter	Value
β	1.5
γ	0.9
θ	0.7
α_0	1.5
α	$\alpha_0 * \gamma^t$

Similarly, the parameters for BA are given as [2]:

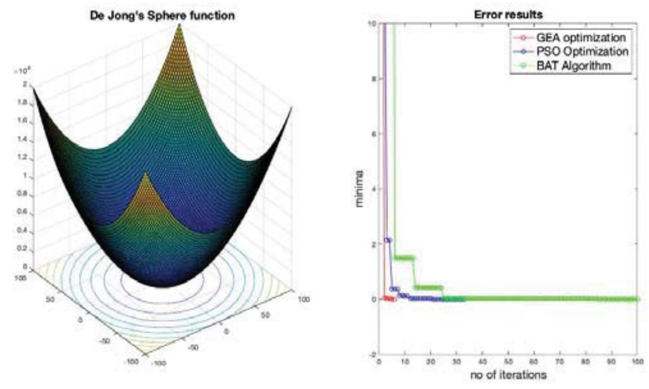
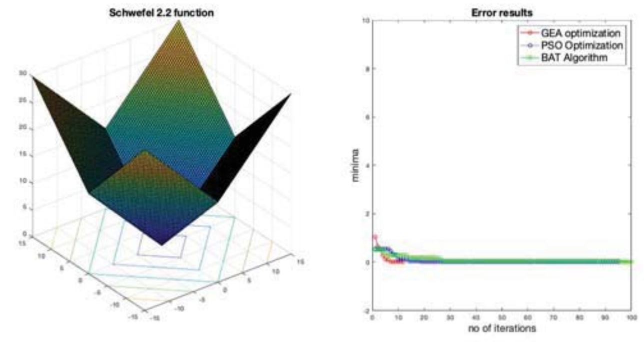
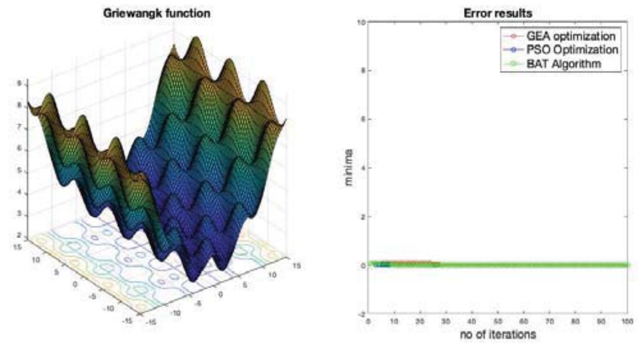
Parameter	Value
Q_{min}	0
Q_{max}	2
α	0.97
γ	0.9
θ	0.7
σ	$0.1 * (ub - lb)$
A	0.95
ρ	0.6

For GEA, we use the parameters as follows [4]:

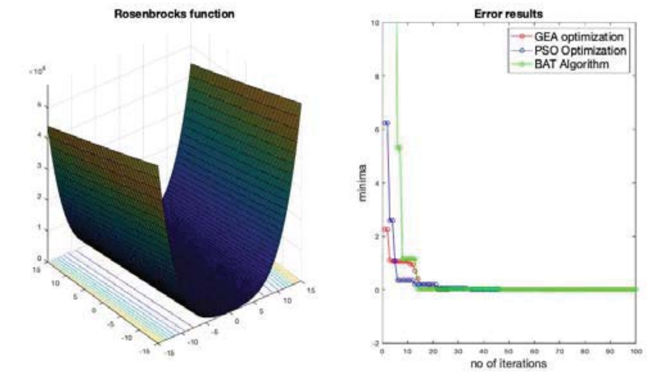
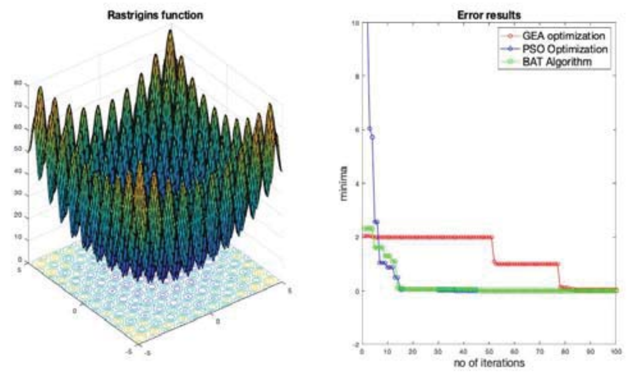
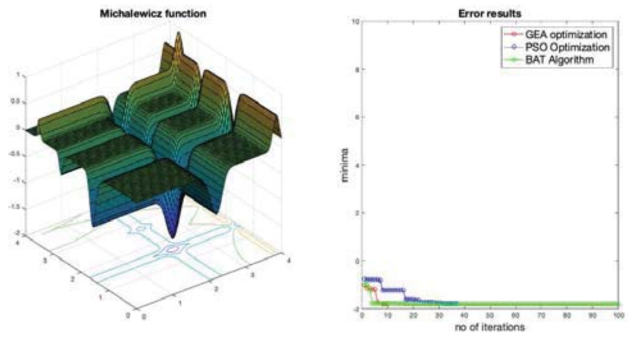
Parameter	Value
c	0.97
β	[0, 2]

2.6 Test Results

Using the above parameters for each algorithm, the functions are simulated in MATLAB from which we obtain the results as per the below table[1].

Functions	Results	Minima
F1		(0,0,0)
F2		(0,0,0)
F3		(0,0,0)

[1] It must be noted that the results obtained in the table refers to a single random observation. Nevertheless, the results displayed correctly reflect the observations obtained over a wide number of simulations.

F4		(1,1,0)
F5		(0,0,0)
F6		(2.2,1.57,-1.8013)

From the above observations, we see that the algorithms perform similarly for unimodal functions. For multimodal functions F3, F5 and F6, the algorithms GEA performs better than BAT and PSO especially due to the presence of the mutation operator which helps to prevent the algorithm getting stuck at its local optima. Furthermore, to enhance the capability of GEA to not get stuck at local optima, it is necessary to tweak the mutation and local search operator in such a way as to perform the mutation and local search on each dimension individually and not on all parameters as a whole. This can be done by modifying equations (12) and (14) as follows:

$$x_i^t = x_i^t + \epsilon M_j .* (rand(1,j) < p) \tag{15}$$

And,

$$x_i^t = x_*^{t-1} + \epsilon L .* (rand(1,j) < p) \tag{16}$$

The above modification is quite important in the case of multimodal functions wherein, the solution tends to get stuck in a local optima as the combined mutation approach renders it difficult for the solution to move all at once, but, when done individually for each dimension, there is a greater chance of finding a different solution. Without this modification, the next tasks of system identification are fruitless as the solution rarely reaches its optimum value.

In conclusion, we see that the GEA algorithm works extremely well for unimodal as well as multimodal functions and hence we can confirm its suitability for usage in the next part which involves system identification.

3 Relay-based Feedback Identification

With our optimization algorithm working, the next step is to apply this same algorithm in parametric identification of a dynamic system. In the part the goal is to perform *black box identification* of a dynamic system applicable to a wide range of processes. The main advantage of black box identification is the fact that this approach necessitates no knowledge about the physical system, but rather, uses the experimental data which includes inputs and outputs and a certain defining factor in terms of the cost function to identify the system to a reasonable accuracy. In this case, the ITAE (*Integral Time Absolute Error)* Method will be used as a criterion for the cost function. With system identification, the end goal is to find an accurate model of the process so as to control it correctly. In the later section, we shall use the model identified from this section to tune our PID controller which will be used to control our process.

3.1 System Schematic

The schematic of our system can be seen in the below figure.

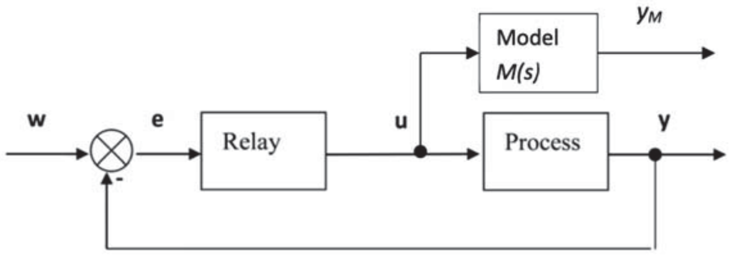


Figure 4 - Experimental Block Diagram Using Simulink

The above figure shows the process *G(s)* that we wish to identify and then control. The input *u* is given by the output of the on-off relay which is used to generate sustained oscillations within the closed-loop system as proposed by Astrom and Hagglund [6]. Once the closed-loop system is running automatically, we use the output from the controller ‘u’ and feed the same as input to our model $G_M(s)$ which, we consider to be a Second Order Plus Time Delay (SOPTD) model which is required to represent the dynamics of the actual process which we are trying to control. The reason for using an SOPTD model is that it can represent almost any linear system. As explained by Ramakrishnan and Chidambaram, the SOPTD model can incorporate various processes such as under-damped and higher order processes in which case, an FOPTD model is not sufficient [7]. Furthermore, SOPTD models can also be used for unstable processes in which case, an FOPTD model is not sufficient.

Thus, by knowing the input ‘u’, the outputs ‘y’ and ‘y_M’, we can proceed with identification of the process.

3.2 Problem Statement

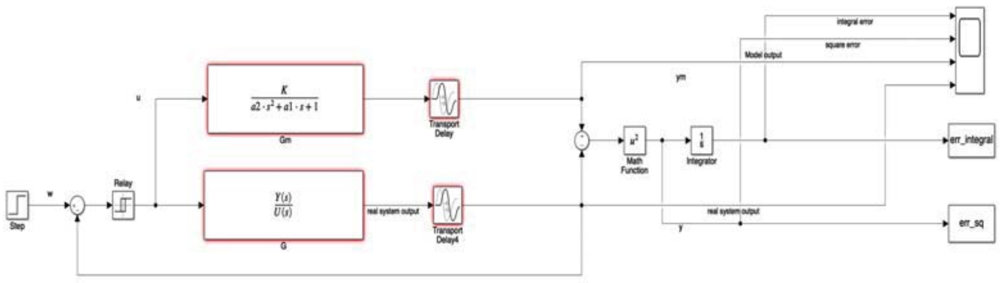


Figure 5 - Relay Identification Schematic

With reference to the schematic from Simulink, we proceed with defining our problem.

1. Model: The SOPTD model describe above can be represented as follows:

$$G_M(s) = \frac{K}{a_2 s^2 + a_1 s + 1} e^{-sT_d} \tag{17}$$

Where, 'K' = Process gain, 'θ' = Time Delay, 'a_1' and 'a_2' are dynamic constants of the transfer function.

2. Parameters: Using the SOPTD model from equation (17), our goal is to identify the parameters as follows:

$$x = [K, a_1, a_2, T_d]^T \tag{18}$$

3. Cost function: The criterion for optimization can be given by the cost function J which can be given by:

$$J = \int_0^{T_k} \left(y_m(t) - y(t)\right)^2 dt \tag{19}$$

Where,
$y_m =$ Model output
$y_t =$ Process output
$T_k =$ Simulation time
The above cost function uses the ITAE criterion as our optimization constraint for minimizing the error over a pre-set simulation time.

4. Constraints: The constraints restrict the upper and lower limits of each parameter

$$\{K, a_1, a_2, T_d\} > 0 \tag{20}$$

3.3 Pre-defined Processes

For the purpose of identification and verifying our identification procedure, we use a number of pre-defined test functions describing various types of processes. These are given as follows:

Table 2 - Predefined Process Functions

No.	Type of Process	Function
1	Non-oscillatory Process (Lag-dominated)	$P_1(s) = \frac{1}{(s+1)(0.1s+1)(0.01s+1)(0.001s+1)}$
2	Balanced Process	$P_2(s) = \frac{1}{(s+4)^4}$
3	Delay-dominated Process	$P_3(s) = \frac{1}{(0.05s+1)^2}e^{-s}$
4	Oscillatory Process	$P_4(s) = \frac{1}{(0.5s^2+s+1)^2}$
5	Non-oscillatory Process with Time Delay	$P_5(s) = \frac{1}{(s+1)(0.3s+1)^2}e^{-sT_d}$
6	Fifth Order Process with Time Delay	$P_6(s) = \frac{e^{-s}}{(5s+1)^5}$

Each of the above processes represent a different type of process and our goal is to fit the complex process into a lower order model which can closely describe the actual response of the systems.

3.4 Identification Results

The results of the above identification procedure and optimization algorithm can be summarized in the table below:

Table 3 - Identification Results for Pre-defined systems

No.	Function	K	a2	a1	Td (s)	fmin	Elapsed Time (s)
1	$\mathrm{P1} = \frac{1}{(s+1)(0.1s+1)(0.01s+1)(0.001s+1)}$	0.998 ± 0.096	0.0674 ± 0.049	1.172 ± 0.235	0.033 ± 0.039	0.0207 ± 0.023	92.37 ± 33.76
2	$\mathrm{P2} = \frac{1}{(s+1)^4}$	0.966 ± 0.083	3.32 ± 0.369	3.034 ± 0.26	0.891 ± 0.099	0.122 ± 0.049	$93{,}9 \pm 45{,}56$
3	$\mathrm{P3} = \frac{e^{-s}}{(0.05s+1)^2}$	1.002 ± 0.009	0.00038 ± 0.001	0.0823 ± 0.008	1.029 ± 0.029	0.324 ± 0.139	135.1 ± 50.1
4	$\mathrm{P4} = \frac{1}{0.5s^2+s+1}$	0.943 ± 0.065	1.017 ± 0.122	1.147 ± 0.107	0.364 ± 0.105	0.364 ± 0.105	107.4 ± 39.65
5	$\mathrm{P5} = \frac{e^{-s}}{(s+1)(0.3s+1)^2}$	0.99 ± 0.10	0.515 ± 0.153	1.437 ± 0.114	1.122 ± 0.105	0.155 ± 0.107	$107{,}83 \pm 57{,}30$
6	$\mathrm{P6} = \frac{e^{-s}}{(5s+1)^5}$	1.022 ± 0.217	109.9 ± 13.17	17.17 ± 1.33	8.20 ± 0.66	0.222 ± 0.058	83.65 ± 69.09

Each system was simulated ten times for consistency to obtain the above results. We can verify the correctness of the above identified systems by comparing the step response and the Nyquist plot of the modelled system with those of the real processes.

For example, a random simulation of the system 6 (P1) obtained the below step response and Nyquist:

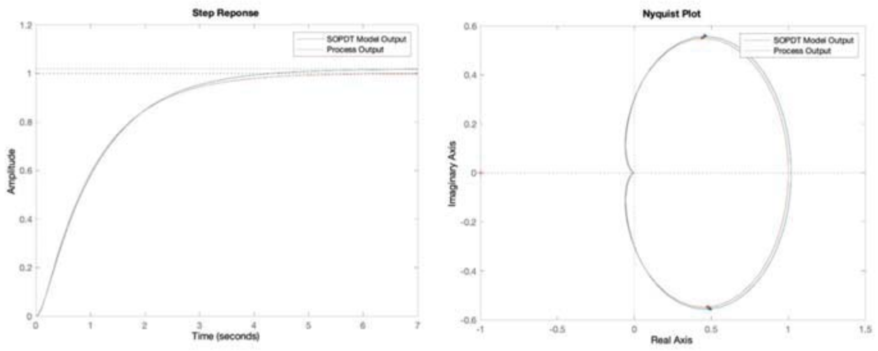


Figure 6 - Step Response, Nyquist Plot - Process P1

We see that both the step response as well as the Nyquist plot, closely resemble the actual system which we are trying to model and therefore, we can say that our identification is correct. Similar results were observed for the remaining systems as well. It must, be noted that although the algorithm manages to converge quickly, it is necessary to set the initial guess in the range selection to reasonable values. In this case the selected range for our initialization was [20, 20, 20, 5].

4 PID Control

As stated earlier, the end goal of system identification of a dynamic process using lower order models is control. Therefore, our goal is to achieve appropriate control of the processes described by the equations in table (2) using the model identified in section (3).

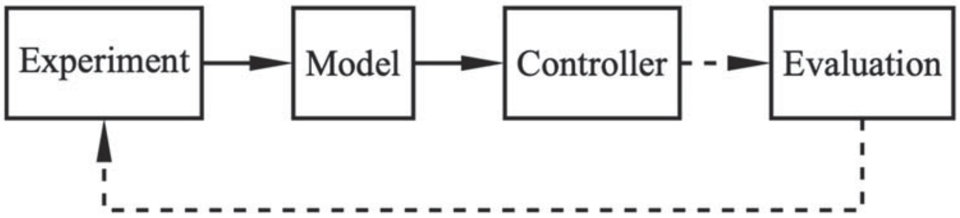


Figure 7 - Tuning Procedure [8]

It must be noted that all the processes described in table (2) are all stable processes. Hence our goal with control is to firstly minimize the settling time and secondly to reduce the overshoot to an acceptable degree. For the purpose of control, we make use of the most commonly used PID controller.

4.1 Closed-Loop System schematic

The block diagram of the controller can be shown in the below figure.

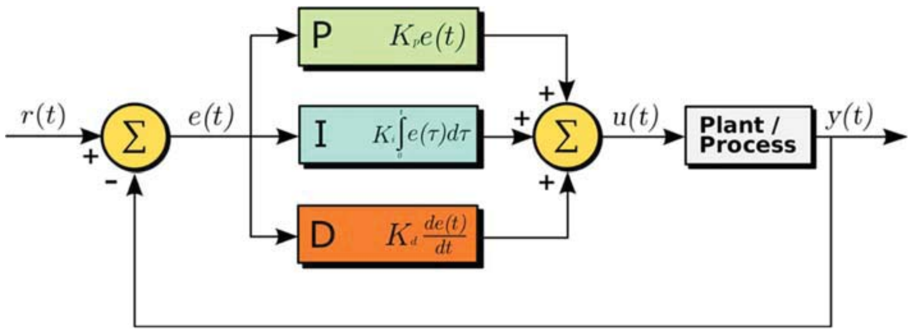


Figure 8 - PID Controller Block Diagram [9]

The equation for the controller in Laplace domain can be given by:

$$R(s) = r_P + \frac{r_I}{s} + r_D s \tag{21}$$

Where,

$r_P =$ Proportional Gain – P-term (K_P)

$r_I =$ Integral Gain – I-term (K_I)

$r_D =$ Derivative Gain – D-term (K_D)

It must be noted that in the above figure (8), we aim to try to control the real unknown process using only the identified model with no more information about the system.

4.2 Tuning Methods

To tune the controller is to find suitable values of the proportional, integral and derivative gain so as to achieve our control objectives. For this, we will have to make use of a few empirical methods for controller tuning, namely:

a) Direct Synthesis Method (DS) or Lambda-Tuning Method
b) Phase Margin Criterion (PMC) based Tuning Method [10]
c) Simple Control (SIMC Tuning Method

For applying the above methods, we shall have to reduce the SOPTD Model $G_M(s) = K.\frac{e^{-sT_d}}{a_2 s^2 + a_1 s + 1}$ obtained in section (3.4) to one of the below forms:

$$G(s) = \frac{(K\omega_0^2)e^{-sT_d}}{s^2 + 2\zeta\omega_0 s + \omega_0^2} \text{ for oscillatory processes} \tag{22}$$

Where,
$\omega_0 =$ oscillation frequency
$\zeta =$ damping ratio

or

$$G(s) = \frac{Ke^{-sT_d}}{(T_1 s + 1)(T_2 s + 1)} \text{ for non-oscillatory processes} \tag{23}$$

Where,
$T_1, T_2 =$ Time constants of modelled system

The relations for tuning the controller by the DS method [11], PMC method and the SIMC methods can be given by the below table:

a) Non-oscillatory Processes $(\zeta \geq 1)$

Table 4 - Tuning Equations (Non-Oscillatory Process)

No.	Parameter	DS	PMC	
			$\boldsymbol{T_d > 0}$	$\boldsymbol{T_d = 0}$
1	$\boldsymbol{r_P}$	$\frac{1}{K}.\frac{T_1+T_2}{\tau_C+T_d}$	$r_p = a_1.r_i$	$r_p = a_1 r_i$
2	$\boldsymbol{r_I}$	$\tau_I = T_1 + T_2$; $r_i = r_p/\tau_i$	$r_i = \frac{\pi - 2\gamma}{2.\lvert K\rvert T_d}$ or $r_I = \frac{\pi}{2T_d.m_A.K}$	$r_i = \frac{1}{K\tau_c}$
3	$\boldsymbol{r_D}$	$\tau_d = \frac{T_0}{2\zeta}$; $r_d = r_p.\tau_d$	$r_d = a_2.r_i$	$r_d = a_2 r_i$

In the above equations, the parameter τ_C is the closed loop time constant and can be estimated using a general rule:

$$\tau_{dom} > \tau_c > T_d \tag{24}$$

Where,
$\tau_{dom} =$ dominant time constant of the process

As a thumb rule, we shall estimate the value of τ_c as: $\tau_C = \frac{\tau_{dom}}{3}$.

For PMC Tuning, the parameters γ and m_A refer to the phase margin and gain margin respectively. These values are generally in the range:

$$\frac{\pi}{6} < \gamma < \frac{\pi}{3}\,;\ 2 < m_a < 5 \tag{25}$$

It must be noted that the PMC tuning method is suitable for an SOPTD Model only.

b) Oscillatory Processes ($\zeta < 1$)

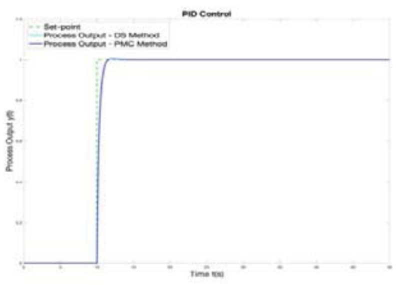
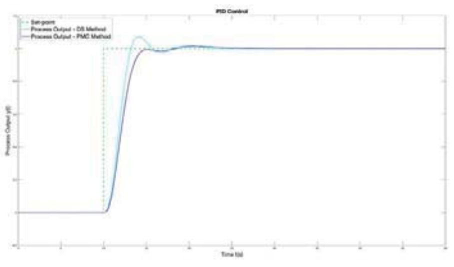
Table 5 - Tuning Equations (Oscillatory Process)

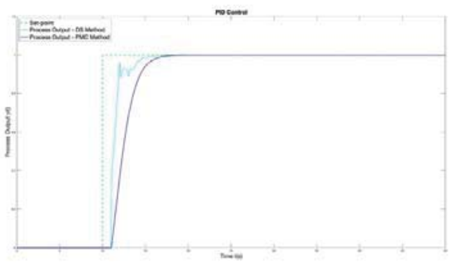
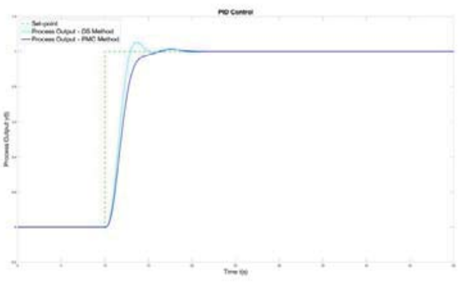
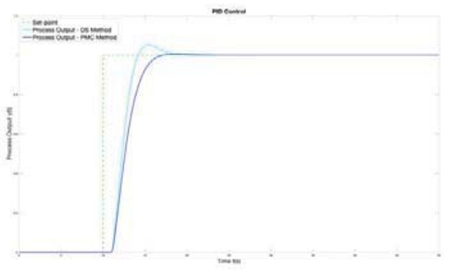
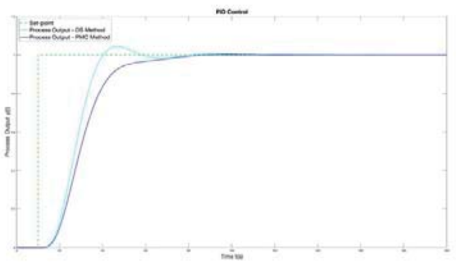
No.	Parameter	DS	PMC	
			$\boldsymbol{T_d > 0}$	$\boldsymbol{T_d = 0}$
1	$\boldsymbol{r_P}$	$r_p = \frac{2\zeta T_0}{\tau_c + T_d}$	$r_p = a_1 . r_i$	$r_p = a_1 r_i$
2	$\boldsymbol{r_I}$	$\tau_i = 2\zeta T_0$ $r_i = \frac{r_p}{\tau_i}$	$r_i = \frac{\pi - 2\gamma}{2.\lvert K\rvert T_d}$ or $r_I = \frac{\pi}{2T_d . m_A . K}$	$r_i = \frac{1}{K\tau_c}$
3	$\boldsymbol{r_D}$	$\tau_d = \frac{T_0}{2\zeta}$; $r_d = r_p . \tau_d$	$r_d = a_2 . r_i$	$r_d = a_2 r_i$

4.3 Tuning Results

The results of the tuning using the above methods are displayed in table

Table 6 - PID Tuning Results

No.	Function	Identified Model	PID Control
1	P1	$G_M(s) = \frac{0.9843}{0.1095s^2 + 1.038s + 1}$	
2	P2	$G_M(s) = \frac{0.9446e^{-0.8837s}}{3.3514s^2 + 2.9958s + 1}$	

3	P3	$G_M(s) = 1.0175.\frac{e^{-1.0285s}}{0.0786s+1}$	
4	P4	$G_M(s) = 0.9469.\frac{e^{-0.8172s}}{0.8799s^2+1.1987s+1}$	
5	P5	$G_M(s) = \frac{0.9767.e^{-1.1509s}}{0.4658s^2+1.4062s+1}$	
6	P6	$G_M(s) = \frac{0.9773.e^{-8.61s}}{107.21s^2+18.07s+1}$	

From the above output, we can conclude that the DS and PMC algorithms perform satisfactorily when it comes to tuning the processes P1-P6 using the identified SOPTD model.

5 Identification and Control on Physical System

To verify the theory and algorithms presented in the previous sections, we apply the same concepts of identification and PID control on a real system from the Automatic Control Laboratory.

5.1 Experimental Setup

The system which we will be using is a combination of two chambers arranged vertically in a tube with a system of interconnecting valves. The schematic of the setup can be seen in the below figures.

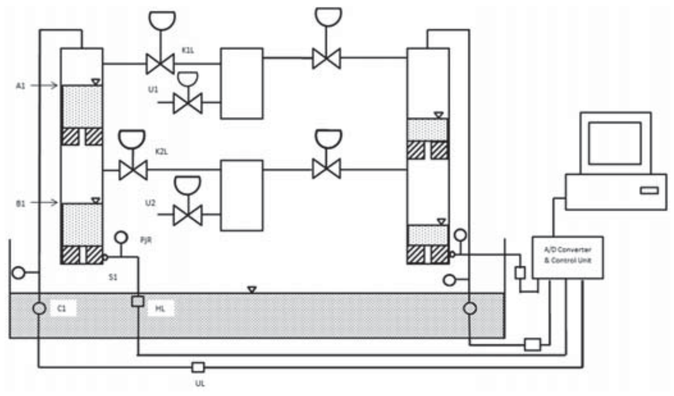


Figure 9 - Schematic Diagram - Two Tanks System

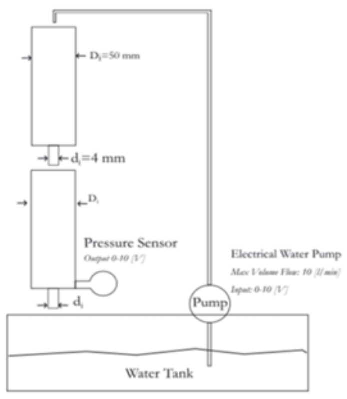


Figure 10 - Functional Diagram - Two Tanks System

The experiment is a combination of two hollow chambers arranged vertically in a tube with a system of interconnecting valves. Pump C1 supplies water to the upper chamber, thus, delivering a pressure head which causes the accumulated water in the upper chamber to trickle down into the lower chamber and eventually back into the tank. A pressure differential sensor is used to map the height of the water in the lower column. Since the upper and lower tanks are connected to each other in series, the system is, by default a second-order system.
To perform the experiment, we need to supply an input signal 0-10V to the pump via the MATLAB / SIMULINK software from the PC which is then transmitted to the pump which in turn pumps up water to the upper chamber A1. From the bottom hole of the upper chamber, water trickles down to the lower tank B1. The system output is measured in terms of the height of water column in the lower chamber B1. This output is measured by a pressure sensor which converts the pressure head to an equivalent height of water column. Before the experiment is performed, it is necessary to test the system for its static characteristics so that we supply the inputs and obtain the readings at around the operating point.

5.2 Simulink Schematic Setup

Similar to the Simulink setup in section (3.2), the experimental Simulink Schema of the physical system is arranged likewise, except that in this case, the input is not fed directly to the theoretical model, but instead is done separately once the readings have been collected. Hence, we will have two separate schematics, one for the physical system and the other, our *Identification Schematic* which uses the values from the physical system to optimize our process model for the purpose of simulation only.

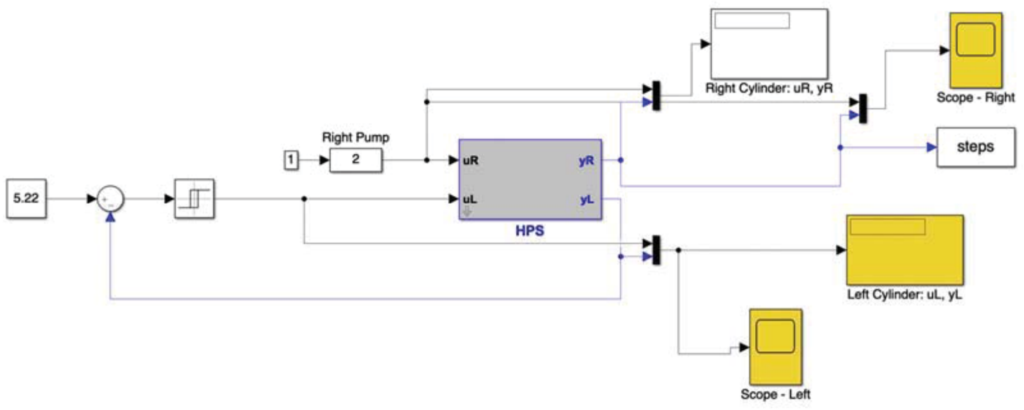


Figure 11 - Simulink Schematic - Physical system

As can be seen from the schematic in Figure (11), we supply the model with an input from our relay controller which in turn causes the system to auto-oscillate. The input set-point 5.22(cm) is supplied to ensure that during experimentation, the process operates around the linear area only which is around the height 5.22 cm and the outputs of the relay are chosen so as be within the linear range. From this setup, the input to the system uL and the output yL

are recorded from the practical setup.

Once the values are collected, these are now fed to the simulation setup (*Identification Schematic)* which operates within the MATLAB / Simulink environment only. The advantage of this is that using just one reading from the physical setup, we can simulate the theoretical model a number of times until we obtain the model which nearly describes the unknown process. The schematic of the optimization part can be seen in Figure (12).

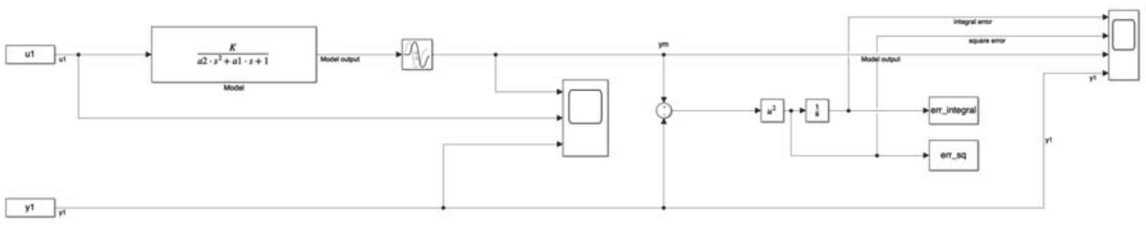


Figure 12 - Identification Schematic

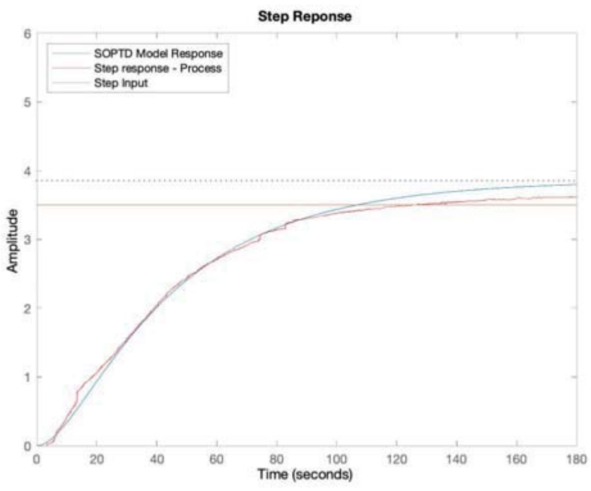
As can be seen from above, the same input $u1$ which was supplied to the real process are fed to the theoretical SOPTD model to obtain its output y_m. By comparing the model output y_m with the real output y_1, we can use our previously described method using the ITAE criterion and the GEA algorithm to identify our system.

5.3 Identification Results

By performing the above experiment and on simulation, we obtain the identified parameters and the model of the system as follows:

$$G_M(s) = \frac{1.10}{376.28s^2+49.62s+1} \tag{26}$$

We shall verify the accuracy of our identified models by plotting the step response and Nyquist plot of the SOPTD model which we have obtained and compare it with the real step response and frequency characteristics.



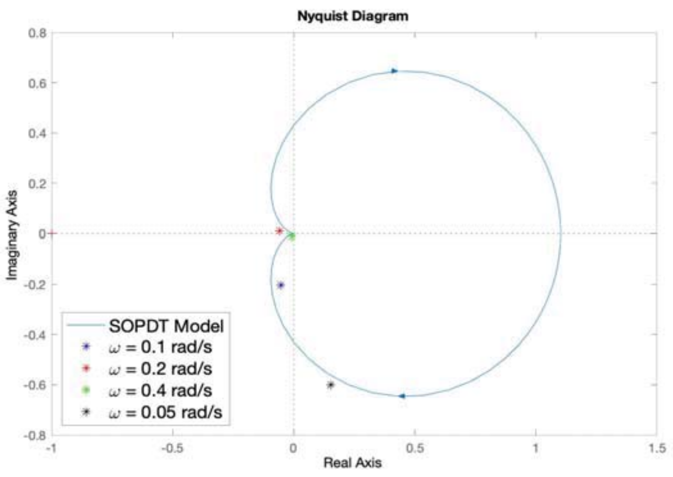


Figure 14 - Step Response - Model vs Real

Figure 13 - Nyquist Plot - Model vs Real

We can see from the above step response and the Nyquist plot that the modelled output closely matches the real results process and therefore we can conclude that our identification method works correctly.

5.4 PID Control

Using the above-identified models, we proceed to control the processes using the PID control methods detailed in section (4.2). The tuning parameters for control of the identified process from equation (26) are as follows:

Table 7 - PID Tuning Parameters - Physical System

No.	Parameter	DS	PMC
1	Proportional Gain, rp	3.36	3.36
2	Integral Gain, ri	0.07	0.07
3	Derivative Gain, rd	25.48	25.48

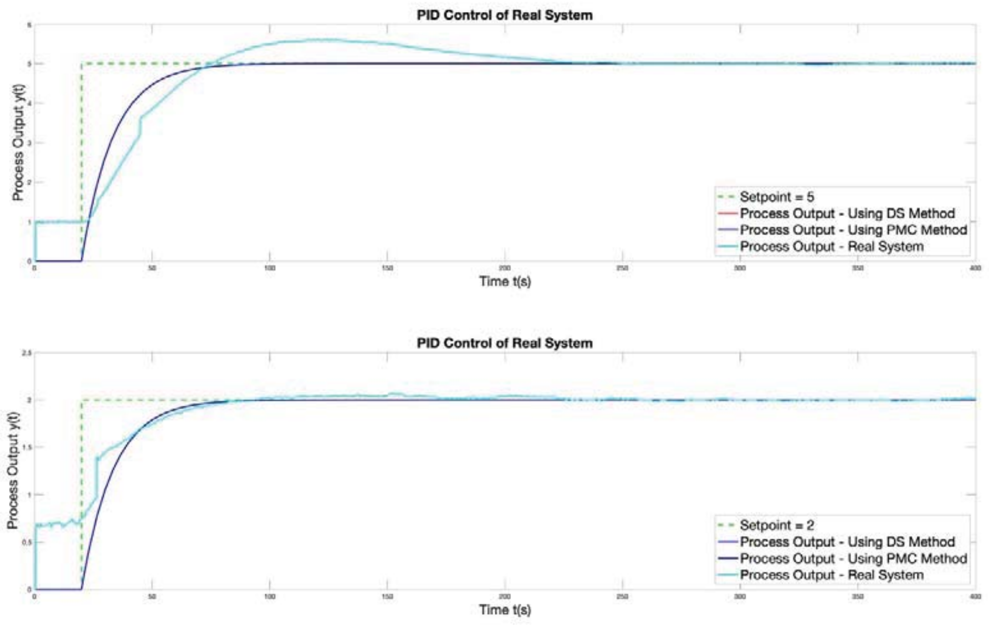


Figure 15 - PID Tuning of Two-tanks system – Simulated vs Real

The figure (15) shows the PID tuning results of the physical setup. As we can see, the control results of the physical system are closer to the simulated results in the second case, when the set-point is 2 (cm) as compared to when the set-point is 5 (cm). This discrepancy can be due to the fact that the theoretical simulation assumes that the system is perfectly linear and doesn’t take into account the effects of saturation on the pump output. This results in a large overshoot when the set-point is increased which is a result of integral wind-up effect in the pump. The overshoot can be reduced suppling a setpoint of lower magnitude around a region nearer to the operating point, or by re-simulating the process by taking into account the pump saturation.

6 Conclusion

To sum up the above concepts, we have briefly discussed three important topics namely Optimization, Identification and Control. From section (2), we have evaluated the possibility of using GEA with the afore-mentioned modifications as an effective strategy for non-linear optimization problems. The advantage of this algorithm lies mainly in its inherent simplicity in execution and its versatility in solving multi-modal optimization problems as compared to the other available algorithms which often remain stuck at the local optima rather than finding the global optimum value in a given search-space. The method was later used in section (3) and applied to a task involving relay-based identification of a dynamic system and from the results in section (3.4), it was confirmed to correctly model the unknown system. Post identification, the process was correctly controlled via simulation using a PID control and with the help of the DS and PMC tuning methods both of which used the identified SOPTD models from identification to effectively control the process. Finally, the results of the complete identification and control using our optimization approach were correctly verified using a real system.

The above methods thus serve as an effective and efficient means of identification and control of unknown processes. Further research into this topic can include application of the GEA algorithm for Model Predictive Control as well as looking into different approaches for optimization like TABU Search [12] or SOMA optimization method or Hybrid Bat Algorithm with Harmony Search [13].

Acknowledgement

This work was supported by the Grant Agency of the Czech Technical University in Prague, grant No. SGS19/158/OHK2/3T/12

REFERENCES

[1] D. Wolpert and W. Macready, “No free lunch theorems for optimization,” *IEEE Transactions on Evolutionary Computation, vol. 1, no. 1,* pp. 67-82, 1997.

[2] X.-S. Yang, Nature-Inspired Optimization Algorithms, Elsevier, 2014.

[3] G. Wang and L. Guo, “A Novel Hybrid Bat Algorithm with Harmony Search for Global Numerical Optimization,” *Journal of Applied Mathematics,* 2013.

[4] L. Cao, L. Xu and E. Goodman, “A Guiding Evolutionary Algorithm with Greedy Strategy for Global Optimization Problems,” *Computational Intelligence and Neuroscience,* pp. 1-10, 2016.

[5] J. Kennedy and R. Eberhart, “Particle Swarm Optimization,” in *Proceedings of ICNN'95 - International Conference on Neural Networks*, Perth, 1995.

[6] K. Astrom and T. Hagglund, “Automatic Tuning f Simple Regulators with Specifications on Phase and Amplitude Margins,” *Automatica, Volume 20, Issue 5,* pp. 645 - 651, 1984.

[7] V. Ramakrishnan and M. Chidambaram, “Estimation of a SOPTD transfer function model using a single asymmetrical relay feedback test,” *Computers & Chemical Engineering, Volume 27, Issue 12,* pp. 1779-1784, 2003.

[8] J. Berner, T. Hägglund and K. Åström, “Asymmetric Relay Autotuning - Practical features for industrial use,” *Control Engineering Practice, Volume 54,* pp. 231-245, 2016.

[9] “PID Controller,” 09 08 2020. [Online]. Available: https://en.wikipedia.org/wiki/PID_controller.

[10] M. Hofreiter, Zaklady Automatickeho Rizeni, Prague: CVUT, 2016.

[11] D. Chen and D. Seborg, “PI/PID Controller Design Based on Direct Synthesis and Disturbance Rejection,” *Industrial & Engineering Chemistry Research*, pp. 4807-4822, 2002.

[12] F. Glover, “Future Paths for Integer Programming and Links to Artificial Intelligence,” *Computers and Operations Research*, pp. 533-549, 1986.

[13] G. Wang and L. Guo, “A Novel Hybrid Bat Algorithm with Harmony Search for Global Numerical Optimization,” *Journal of Applied Mathematics*, 2013.

[14] L. Ljung, System Identfication: Theory for the User, Linkoping: Prentice Hall, 1987.

[15] M. Hofreiter, “Alternative Identification Method using Biased Relay Feedback,” in *IFAC Papers Online*, Prague, 2018.

[16] A. Gupta, *Bat Optimization Algorithm,* MATLAB Central File Exchange, 2020.

ISBN 978-80-01-06617-1