Adaptative, auto-tuning PID controller using genetic algorithms

Final report and major project

Théo Champion

School of Science (CSSE), RMIT University, COSC1207/2033 Evolutionary Computing

Submitted 5 June 2019

Abstract

Control loop feedback mechanism such as proportional–integral–derivative (PID) controllers are widely used for industrial control systems. They, however, have limitations when it comes to time-varying parameters and often still require manual tuning. The aim of this project is to investigate the use of genetic algorithms for designing a self-tuning, adaptative PID controller. The algorithm will have to determine optimal values for the proportional, integral and derivative gains in real-time to satisfy the control application's need even in an environment subjected to dynamic changes.

A simulator was developed to test and benchmark the algorithm. It was then open sourced and made accessible on the internet for anyone to use, providing decentralized research data gathering.

Genetic algorithms were examined in details and shown to be outperforming standard tuning methods such as the Ziegler Nichols method in parameter optimization. This was however discovered to be at the cost of stability as it was discovered that the genetic algorithm would sometime yield unstable controllers.

Adaptability through online tuning using genetic algorithms was achieved in the simulation but was shown to be inapplicable for real world control system due to an unpredictable convergence speed during the recovery process.

Keywords: Genetic algorithms, PID, Evolutionary computing

Introduction

PID controller is one of the earliest and most used control method in the industry due to the simplicity and robustness of the algorithm (Astrom, 1995). However, PID controllers rely on a set of parameters that has to be derived for each control application, as they depend on the response characteristics of the complete loop external to the controller. For this reason, implementation of such system still requires good knowledge about the system and, if many methods for PID tuning have been proposed, most of them are still done via the laborious process of manual tuning.

This paper will investigate the use of genetic algorithms for designing self-tuning, adaptative PID controllers. The algorithm will have to determine optimal values for the proportional, integral and derivative gains in real-time to satisfy the control application's need even in an environment subjected to dynamic changes.

Literature Review

The idea of using different variations of genetic algorithms for PID tuning has been around for a long time and has been the subject of multiple related research projects. This section will review some of the most influential papers in that area and indicates where the current model fits in.

Ziegler, J. and Nichols, N. (1993). Optimum Settings for Automatic Controllers. Journal of Dynamic Systems, Measurement, and Control, 115(2B), p.220.

This paper is the groundwork for the field of dynamic system control. It describes a heuristic method of tuning a PID controller that works the following way; First, the proportional (P), integral (I) and derivative (D) gains are set to zero. The proportional gain is then increased until the output of the control loop has a consistent oscillation. The value of P along with the oscillation period T is then used the determine the value of the integral and derivative gains from predefined values depending on the type of controller. This method, known as the Ziegler method, is of interest for this paper as it is widely used in the industry and therefore will be used as a comparison against our proposed method.

Harris, T., Astrom, J. and Wittenmark, B. (1991). Adaptive Control. Technometrics, 33(4), p.487.

A constant gain-feedback system such as PID controllers are not adaptative systems as they do not modify their behavior in response to dynamic changes in the dynamics of the process. This paper provides a method for converting a non-adaptive system to adaptative one using a parameter adjustment loop in addition to the normal feedback loop of the controller. It is of interest for this paper as the goal is to design an adaptive controller resilient dynamic changes.

Mirzal A., Yoshii S. and Furukawa M. (n.d.). PID Parameters Optimization by Using Genetic Algorithm

This paper explores the process of static PID tuning using genetics algorithms for minimizing the First Order Lag plus Time Delay of a system. It is of interest for this paper as it gives an extensive look on how to design efficient objective functions for evaluating PID controllers performance, good insights of selection operators as well as techniques for efficient mutation and crossover operation on floating point numbers.

From this review, we conclude the following: Classical methods for PID tuning such as the Iterative method or the Ziegler-Nichols method have been around for a long time and are still used today. In recent years, genetic algorithms have been used to automate the process of tuning and in some cases achieved up to 70% increase in performance of the previously mentioned methods (Mirzal, Yoshii & Furukawa 2003). However, as with the classical methods, methods utilizing genetic algorithms require the system to be taken offline for new parameters to be applied making it not suitable for systems with time-varying parameters. Work has been made in the field of adaptive PID controllers and many methods proposed, however, none of these methods have yet been integrated with genetic algorithms.

Description of the model

1. Simulated dynamic test environment

In order to develop and test the solution, a simulated dynamic test environment was created.

It is based on a classical problem widely used for testing PID controllers that is creating a self-balancing vehicle. It consists of the following components:

A scene which is comprised of 4 delimitation walls and a randomly shaped rotating wheel (or challenge wheel) (Fig. 1). And vehicles, which are comprised of a body and a rotating wheel which angular speed is controlled by a standard PID controller (Fig. 1).

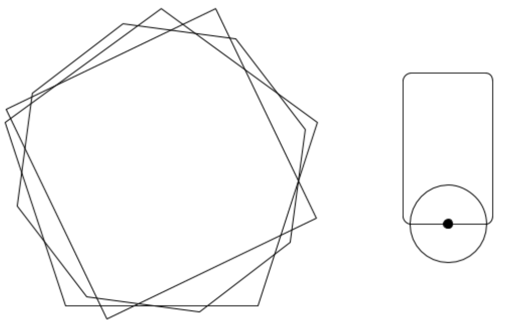
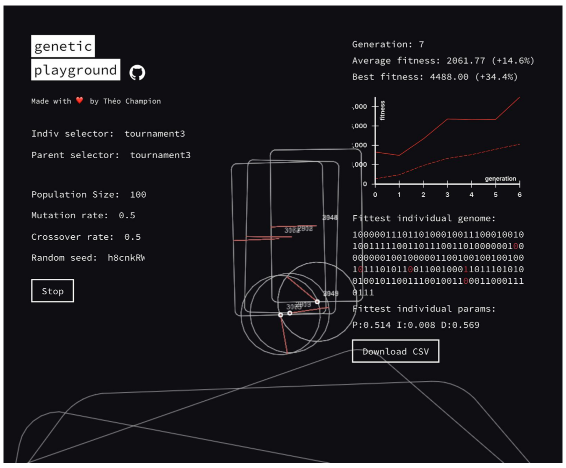


Figure 1: An example challenge wheel next to a vehicle

When a simulation is run, vehicles are added to the scene along with parameters for their PID controller and will have to stay balanced as long as possible on the increasingly faster rotating wheel.

If a vehicle's body touches the wheel or any of the 4 delimiter wall, this vehicle is considered dead and removed from the simulation. Performance stats for that vehicle are then recorded and returned to the caller. The dynamic environment is simulated by randomly varying the rotation speed of the challenge wheel during the simulation.

The simulator was developed entirely in the Javascript programming language allowing it to run anywhere without compilation (Fig. 2).



```
Indiv selector:  tournament3
Parent selector:  tournament3

Population Size:  100
Mutation rate:  0.5
Crossover rate:  0.5
Random seed:  h8cnkRW

Generation: 7
Average fitness: 2061.77 (+14.6%)
Best fitness: 4488.00 (+34.4%)

Fittest individual genome:
10000011101101000100111000100101
10011111001101110011010000000100
00000010010000011001001001001001
01111010110011001000110111010100
10010110011100100110001100011101
11
Fittest individual params:
P:0.514 I:0.008 D:0.569
```

Figure 2: Simulator running in a browser environment

The simulator was then served through a web server (https://theochampion.github.io/genetic-playground/) and shared on the internet allowing anyone to run and tinker with the settings. This allowed the gathering of useful research data and insights in a decentralized way.

Source code for the simulator was also made available under the MIT license at https://github.com/theochampion/genetic-playground allowing anyone to suggest and contribute to its development.

2. Genetic Algorithm

2.1 Evaluation

When applying a genetic algorithm to a problem, one of the most crucial parts is to properly evaluate the performance of each chromosomes. This is done in the objective function (or fitness function). In the proposed solution we use a performance index as the objective function.
The first indices we evaluated is the time (or the number of ticks in the simulation engine) a vehicle has survived in the simulation. This gives a good insight into the overall performance of an individual in the simulation. However, this doesn't account for the variable change in performance during the simulation needed for real-time adaptative tuning. To palliate this problem we added another performance index evaluating the difference in angular position from the desired (upright) position (Da).

The final performance index is computed using the following formula:

$$fitness = \sum_{i=1}^{n} 1 - |Da|$$

With n being the number of engine ticks.

2.2 Reproduction

Reproduction is the process of selecting the individuals that will populate the next generation.
This selection can be achieved using different techniques but is usually bias toward fitter individuals. Different selection operators were evaluated like the `*Roulette selection*`, `*Stochastic Universal sampling*` or `*Normalised geometric selection*`. Best results were however achieved using `*Tournament selection*`. In tournament selection, a few individuals (or chromosomes) are chosen at random. The individual with the best fitness of that random selection pool is then selected. The tournament selection technique used can be described by the following pseudo-code:

```
func tournament(pop, k):
fittest = null
for i=1 to k
    chr = pop[random(1, popSize)]
    if (fittest == null) or fitness(chr) > fitness(chr)
        fittest = chr
return fittest
```

Probability of an individual to be part of the initial selection pool (or "selection pressure") define the behavior of that selection technique and can be adjusted by changing the tournament size. A higher tournament size will yield fitter individuals whereas a smaller tournament size will yield weaker but more diverse individuals. The adaptative nature of our solution creates constants need for new solutions to accommodate dynamic changes in the environment hence the need for a diverse population to be evaluated and each generation. We, therefore, settled on a relatively low tournament size (k) of 3.

2.3 Crossover

After the selection process is complete, parents will undergo the crossover operation. Crossover is an operation that aims to combine the genetic information of two individuals to create a new offspring for the next generation. As with selection, it exists multiple methods for implementing crossover operations. One common method is "*single-point-crossover"* where one point is chosen at random in the binary string and bits to the right of that point are swapped between the two parents

(Goldberg, 1989). We used "*multi-point-crossover"* which is a variant that works the same way, only allowing multiple crossover points (Fig. 3) which accommodate for our three encoded floating point parameters.

Parent1: 00110 1010010 01110010 1010

Parent2: 01000 1000011 00000110 1110

⇩

Child 1: 00110 1000011 01110010 1110

Child 2: 01000 1010010 00000110 1010

Figure 3: Example of multi-point-crossover

2.4 Mutation

Selection and crossover on their own will generate a large number of new chromosomes. However, over generations, this alone might lead to a lack of diversity which can cause the algorithms to converge to suboptimal solutions. To overcome this problem, random changes in individuals chromosomes are introduced through a process called mutation. Again, there are many different techniques for implementing mutation the most common one being the "*bit-flip-mutation"* in which a bit is selected at random in the binary string and flipped to introduce change. However, this type of mutation can cause problems when used with floating point encoded binary string. In effect, in our implementation, we used the double precision format defined by the IEEE 754-1985 standard in which the number is defined using 64 bits, 1 bit for the sign, 11 bits for the exponent and the remaining 52 bits for the fractions (Fig. 4).

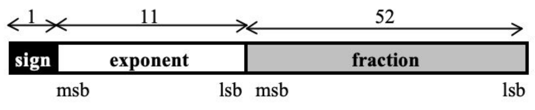


Figure 4: IEEE 754-1985 floating-point format

Using standard bit-flip mutation, in that case, can cause impossible solutions to be generated (e.g. if the sign bit is flipped). We, therefore, used a modified version of the random bit-flip operation, operating only on the safe parts of the binary string by ignoring the sign bit and the most significant bit of the exponent (Budin, Leo & Golub, 2003).

Comparison with existing models

Compared to the works identified in the literature review, the proposed model differs in several ways.
Firstly, compared to the the Ziegler & Nichols method, the proposed method offer on-line auto-tuning meaning the system will no longer need to be taken offline to be re-tuned while offering better performance on average.
Compared to the proposed method for creating an adaptive PID controller by Harris, Astrom, and Wittenmark, the proposed solution offers the same results by modifying its behavior via continuous tuning in response to dynamic changes in the dynamic of the system.
Compared the the method for PID parameter optimization by Mirzal, Yoshii and Furukawa, the proposed solution is very similar as they both make use of genetic algorithms for parameter optimization. The proposed solution, however, focuses more on its application for creating a adaptive system than the raw tuning performance.

Results

All the following tests were run over 200 generations (as no visible improvements were observed after the 200th generation) with a population size of 100, a mutation rate of 0.1 and a crossover rate of 0.6.

Parameter optimization

To analyze the performance of the parameter optimization, tuning performance tests were conducted against the Ziegler-Nichols method. The simulator was set to run without random variation to benchmark raw tuning performance.

	Proportional	Integral	Derivative
Ziegler	0.91091	0.00821	0.67242
GA	0.90731	0.00791	0.71237

Table 1: Parameter tuning comparison

Results in table 1 show very similar tuning settings after 200 generations suggesting an efficient tuning achieved by the GA. This however wasn't always the case with the GA sometime yielding an underperforming, unstable controller.

Adaptability

A series of adaptability tests were conducted against the Ziegler-Nichols standard tuning method (Fig. 5).

To introduce dynamic change, the simulator was set up to produce random variation in the rotation speed of the challenge wheel at each generation.

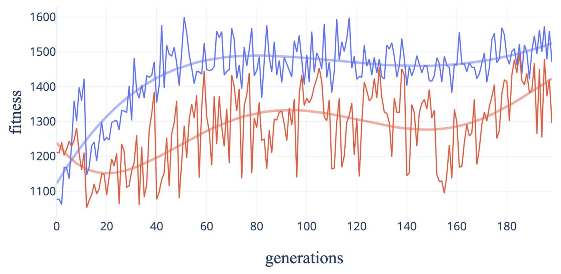


Figure 5: Comparaison over Ziegler method

The results in Fig. 5 show the proposed solution (shown in blue) outperforming the Ziegler Nichols method (shown in red) by an average 30% after the initial tuning phase (generation 0 to 10). The difference in variation over time of the fitted curves (shown in bold colors), also suggest the proposed solution being more resilient to dynamic changes (here, the varying rotation speed).

Discussion

The results previously mentioned established GAs as a valid solution for PID parameter optimization yielding controllers that outperform traditional methods in a relatively low number of generations. There is however still convergence problems in applying the GA leading to suboptimal solutions in some cases.

Real-time adaptability of the system was also investigated and if results show that the system is able to successfully recover and adapt to dynamic changes, the speed at which the system reaches convergence again is too slow and unpredictable for being used in any critical applications outside of the simulator. The previously mentioned unpredictability is due to the relatively high mutation rate needed for fast adaptation of the system.

Conclusion

It was established, by comparison, that GA's can be used for parameter optimizations as it outperformed standard tuning methods (e.g. Ziegler Nichols) without requiring the labor and expertise associated with manual tuning.

Using genetic algorithm to create an adaptive PID controller through online tuning however was unsuccessful. If the proposed system was able to respond and adapt to dynamic changes, the speed at which the system reaches convergence again is too slow and unpredictable for being used in any critical applications outside of the simulator. Recommendation for the continuation of this project would include designing a system for improving stability in the dynamic change recovery process by replacing any unstable controllers generated by the last stable controller evaluated by the genetic algorithm.

References

- Astrom K, Hagglund T. 1995. PID controllers: theory, design and tuning. New York: ISA
- Ziegler, J. and Nichols, N. (1993). Optimum Settings for Automatic Controllers. Journal of Dynamic Systems, Measurement, and Control, 115(2B), p.220.
- Harris, T., Astrom, J. and Wittenmark, B. (1991). Adaptive Control. Technometrics, 33(4), p.487.
- Mirzal A., Yoshii S. and Furukawa M. (n.d.). PID Parameters Optimization by Using Genetic Algorithm
- Budin, Leo & Golub, M & Budin, Andrea. (2019). Traditional Techniques of Genetic Algorithms Applied to Floating-Point Chromosome Representations. 61-29.
- An American National Standard: IEEE Standard for Binary Floating-Point Arithmetic, ANSI/IEEE Standard 754-1985.