Testing Proportional-Integral-Derivative (PID) Controller with Metamorphic Testing

Kun Qiu, Zheng Zheng*
Beihang University, China
Email: qiukun@buaa.edu.cn; zhengz@buaa.edu.cn

Tsong Yueh Chen
Swinburne University of Technology, Australia
Email: tychen@swin.edu.au

I. Introduction

Proportional-Integral-Derivative (PID) controller is the most commonly used controller in the industry [1]. Since controller program is the core of control systems that are commonly used in safety-critical applications, it is essential to conduct a thorough testing prior to the lease of the software. However, the testing of the controller program has long been a challenging problem because of the infeasibility to know whether its output is correctly computed or not, which is known as the oracle problem.

Metamorphic Testing (MT) [2] was developed to alleviate the oracle problem, that is, to make testing possible even when the correctness of the computed outputs could not be verified. In this study, we propose to use MT to test PID controller programs. We investigated four popular PID algorithms and used the technique of mutation analysis [3] to measure the failure detection capability of our proposed method. The experimental results demonstrated the usefulness of MT in revealing failures for these four popular controller programs.

II. Background

A. PID Controller

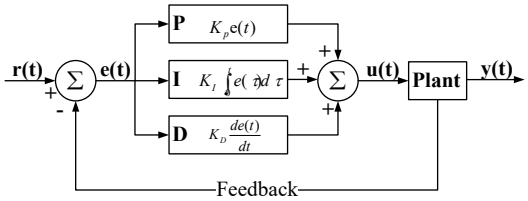

Fig. 1: A typical control system block diagram with a PID controller in a closed feedback loop.

Fig. 1 shows a typical control system block diagram with a PID controller in a closed feedback loop. The control system has two main components, the PID controller and the plant. The PID controller consists of three blocks, namely P, I, and D block. The plant appears as the rightmost block in Fig. 1.

The following equation describes the mathematical model for the PID controller, which consists of the proportional term, the integral term, and the derivative term.

$$u(t) = K_P e(t) + K_I \int_0^t e(\tau)d\tau + K_D \frac{de(t)}{dt} \quad (1)$$

where $u(t)$ denotes the control signal output, $e(t)$ denotes the error input, K_P denotes the proportional gain, K_I denotes the integral gain, and K_D denotes the derivative gain.

There are four popular algorithms to implement equation (1).

1) Algorithm 1:

$$u[k] = K_P e[k] + K_I \sum_{i=0}^{k} e[i] + K_D (e[k] - e[k-1]) \quad (2)$$

2) Algorithm 2:

$$u[k] = K_P e[k] + K_I \sum_{i=1}^{k} \frac{e[i] + e[i-1]}{2} + K_D (e[k] - e[k-1]) \quad (3)$$

3) Algorithm 3:

$$u[k] = K_P e[k] + \beta K_I \sum_{i=0}^{k} e[i] + K_D (e[k] - e[k-1])$$
$$\beta = \begin{cases} 1, Abs(e[k]) \le \varepsilon \\ 0, Abs(e[k]) > \varepsilon \end{cases} \quad (4)$$

4) Algorithm 4:

$$u[k] = K_P e[k] + I[k] + K_D(e[k] - e[k-1])$$
$$I[k] = \begin{cases} I_{\max}, & I[k-1] + e[k] \ge I_{\max} \\ I[k-1] + e[k], & -I_{max} < I[k-1] + e[k] < I_{max} \\ -I_{max}, & I[k-1] + e[k] \le -I_{max} \end{cases} \quad (5)$$

Where $e[k], I[k], u[k]$ denote the k_{th} element of the sequence respectively, $K_P, K_I, K_D, \varepsilon, I_{max}$ denote parameters, $Abs(\cdot)$ denotes the absolute value function.

B. Metamorphic Testing (MT)

MT was developed to test programs whose computed outputs could not be verified. MT has four steps: 1) The first and most important step is to identify the metamorphic relations (MRs) for the algorithm under test; 2) The second step is to generate source test cases and execute them; 3) The third step is to construct follow-up test cases based on the MRs and the source test cases (and their outputs if necessary) and execute them; 4) The last step is to compare the results of source test cases and their follow-up test cases against MRs. If not all the MRs are satisfied, the program is incorrect [2].

978-1-5386-2387-9/17 $31.00 © 2017 IEEE
DOI 10.1109/ISSREW.2017.47

III. Testing the PID Controller Program

In this section, we firstly identify the MRs for PID controller programs. After that, we can follow the MT steps in section II-B to make test. Let $e_i(u_i)$ denote the i_{th} error input sequence (control output sequence), $e_i[-1](u_i[-1])$ denote sequence's last element, $K_{Pi}, K_{Ii}, K_{Di}, \varepsilon_i, I_{max_i}$ denote the i_{th} program configurable parameters, $\mathbb{R}$ denote the set of real numbers, $max\{\cdot\}$ denote the maximum element function for a sequence, $Perm(\cdot)$ denote the permutation function for the sequence, $Sum(\cdot)$ denote the summation function for the sequence, and a $Const$ denote a constant real number.

MR1: Homogeneity
IF $e_i = \alpha e_j, \alpha \in \mathbb{R}$ THEN $u_i = \alpha u_j$.

MR2: Homogeneity
IF $e_i = \alpha e_j$ AND $\varepsilon_i = \alpha \varepsilon_j = max\{e_i\}, \alpha \in \mathbb{R}$ THEN $u_i = \alpha u_j$.

MR3: Homogeneity
IF $e_i = \alpha e_j, \alpha \in \mathbb{R}$ AND $I_{max_i} = I_{max_j} = \infty$ THEN $u_i = \alpha u_j$.

MR4: Additivity
IF $e_k = e_i + e_j$ THEN $u_k = u_i + u_j$

MR5: Additivity
IF $e_k = e_i + e_j$ AND $I_{max_i} = I_{max_j} = \infty$ THEN $u_k = u_i + u_j$

MR6: Permutation
IF $K_{Ii} = K_{Ij} = K_{Di} = K_{Dj} = 0$ AND $0 \neq K_{Pi} \neq K_{Pj}$ AND $e_j = Perm(e_i)$ THEN $K_{Pj} \cdot Sum(u_i) = K_{Pi} \cdot Sum(u_j)$

MR7: Permutation
IF $K_{Pi} = K_{Pj} = K_{Di} = K_{Dj} = 0$ AND $0 \neq K_{Ii} \neq K_{Ij}$ AND $e_j = Perm(e_j)$ THEN $K_{Ij} \cdot u_i[-1] = K_{Ii} \cdot u_j[-1]$

MR8: Permutation
IF $K_{Pi} = K_{Pj} = K_{Di} = K_{Dj} = 0$ AND $0 \neq K_{Ii} \neq K_{Ij}$ AND $e_j = Perm(e_j)$ AND $I_{max_i} = I_{max_j} = \infty$ THEN $K_{Ij} \cdot u_i[-1] = K_{Ii} \cdot u_j[-1]$

MR9: Constant Addition
IF $K_{Pi} = K_{Pj} = K_{Ii} = K_{Ij} = 0$ AND $0 \neq K_{Di} \neq K_{Dj}$ AND $e_j = e_i + Const(Const \in \mathbb{R})$ THEN $K_{Di} \cdot u_j = K_{Dj} \cdot u_i$

IV. Experimental Results and Discussions

We adopt the mutation analysis approach [3] to evaluate our approach's effectiveness. We used a tool, known as *mutate.py* [4], to randomly generate mutants based on traditional mutation operators [3]. We totally generated 56 source test cases. Then we generated the follow-up test cases based on the source test cases and MRs. In our context, a mutant is said to be killed if not all MRs are satisfied for the results of all generated source test cases and their follow-up test cases.

Table I summarizes the mutation analysis results on the implementations of the four algorithms given in Section II-A. The second column shows the MRs used for the relevant algorithms. The third column shows the number of total mutants seeded for the relevant algorithm implementations. The last column presents the mutation scores (the ratio between the number of killed mutants and total mutants [3]). Larger mutation score indicates better failure detection capability. All mutation scores are equal or greater than 80.0%. Considering that the seeded mutants will not cause program crash and then will not be easily explored by traditional testing methods, it can be concluded that our approach is very effective.

TABLE I: MRs used for relevant algorithms and mutation analysis results

	MRs Used	# Mutants	Score
Algorithm 1	MR1, MR4, MR6, MR7, MR9	25	88.0%
Algorithm 2	MR1, MR4, MR6, MR9	30	80.0%
Algorithm 3	MR2, MR6, MR7, MR9	28	85.7%
Algorithm 4	MR3, MR5, MR6, MR8, MR9	28	89.3%

Fig. 2 (a)-(d) show the mutation score of individual MR for four algorithms respectively. As shown, all used MRs' mutation scores are greater than 28.6%, indicating that all the proposed MRs are useful at failure detection. The experimental data also showed that even when only one of the MRs is proposed, our approach can detect faults.

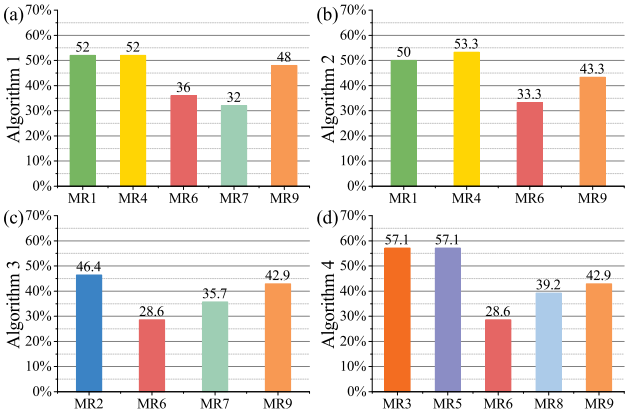

Fig. 2: Mutation score of individual MR for each algorithm

V. Conclusion

In this paper, we proposed to use MT to test the PID controller programs. The mutation analysis results show that MT has a satisfactory failure detection capability, which should be further studied on how to test more complex controller programs.

VI. Acknowledgement

This work is supported by the National Natural Science Foundation of China (Grant NO. 61772055).

References

[1] A. O'Dwyer, *Handbook of PI and PID controller tuning rules*. World Scientific, 2009.

[2] F. Chan, T. Chen, S. C. Cheung, M. Lau, and S. Yiu, "Application of metamorphic testing in numerical analysis," in *Proceedings of the IASTED International Conference on Software Engineering (SE98)*, pp. 191–197, 1998.

[3] B. H. Smith and L. Williams, "On guiding the augmentation of an automated test suite via mutation analysis," *Empirical Software Engineering*, vol. 14, no. 3, pp. 341–369, 2009.

[4] A. Babu. (2016) Github - arun-babu/mutate.py. [Online]. Available: https://github.com/arun-babu/mutate.py