Accuracy of Timers made in various programming languages

Mark Kocherovsky

Lawrence Technological University, Department of Mathematics and Computer Science, Southfield, MI 48075

(Dated: October 13, 2019)

Abstract

The problem is that a timer made in Flash is not accurate to real time. Substitutes made in Python, C#, and C++ are also inaccurate. An attempt is made to correct the error using the C++ by giving the program high CPU priority. This is unsuccessful. Another attempt to correct the issue is made by calculating the factor of on-screen time to real time. This is a marked improvement, but not totally successful. It reveals that the deviation is not always the same. A third attempt to correct the error was made by using online timers. This was successful. Investigations into the pattern of average deviation with different seconds as input reveal no discernible pattern. However, it reveals that the cause of the issue is the Input/Output processes.

I. INTRODUCTION

For many years, Robofest has used a proprietary timer to manage competitions. This timer is in the form of an executable made in Flash. However, in 2019, Robofest received complaints that the timer is inaccurate. This paper explores this claim and the time deviation of other programming languages, as well as potential solutions and explanations.

II. EXPERIMENTAL PROCEDURE

Each programming language was tested by creating a timer program that would display a number on the screen (taken as input from the user), wait a second, and then decrease the number, and repeat until the time has run out. In theory, input of 60 would have taken 60 seconds to run out. For the Robofest Timer, the executable was opened and used as normal. The languages tested were the Robofest Timer (Flash), C++, C#, and Python. C++ and C# were tested through Visual Studio 2017, and Python through the Python IDLE. Each platform was tested for 60 seconds, 180 seconds, and 600 seconds three times each, for a total of nine tests per platform, leading to 36 tests in total. They were tested against an android phone's stopwatch, which introduces a measure of uncertainty.

III. EXPERIMENTAL DATA AND INTERPRETATION

For each test, the null hypothesis is that each timer has no discrepancy between the real time and the intended time, and the alternative hypothesis, which is the claim, is that there is a discrepancy.

A. Robofest Timer

Figure 1 shows the data collected for the Robofest Timer. The timer, on average, takes 5.74% longer than it is meant to, with a standard deviation of .324%. A t-test reveals a p-value on the order of 10^{-11}. Therefore, at 95% confidence, there is evidence to support the claim. The Robofest Timer is not accurate.

Robofest Timer				
Trial	Time (s)	Actual Time (s)	Difference (s)	% change
1	60	63	3.00	5.00%
2	60	63.63	3.63	6.05%
3	60	63.36	3.36	5.60%
4	180	190.78	10.78	5.99%
5	180	190.79	10.79	5.99%
6	180	190.72	10.72	5.96%
7	600	634.21	34.21	5.70%
8	600	634.21	34.21	5.70%
9	600	634.16	34.16	5.69%
Average				5.74%
Standard Deviation				0.00324
t-test			p-value	1.74E-11

FIG. 1. Data collected for the Robofest Timer.

B. C++ Timer

Figure 2 shows the data collected for a timer made in C++. The timer, on average, takes 11.84% longer than it is meant to, with a standard deviation of .285%. A t-test reveals a p-value on the order of 10^{-20}. Therefore, at 95% confidence, there is evidence to support the claim. The C++ Timer is not accurate.

C. C# Timer

Figure 3 shows the data collected for a timer made in C#. The timer, on average, takes 9.07% longer than it is meant to, with a standard deviation of .356%. A t-test reveals a p-value on the order of 10^{-13}. Therefore, at 95% confidence, there is evidence to support the claim. The C# Timer is not accurate.

D. Python Timer

Figure 4 shows the data collected for a timer made in Python. The timer, on average, takes 23.83% longer than it is meant to, with a standard deviation of .018%. A t-test reveals a p-value on the order of 10^{-11}. Therefore, at 95% confidence, there is evidence to support

C++ Timer				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	67.25	7.25	12.08%
2	60	67.4	7.40	12.33%
3	60	67.31	7.31	12.18%
4	180	200.76	20.76	11.53%
5	180	201.25	21.25	11.81%
6	180	200.87	20.87	11.59%
7	600	670.17	70.17	11.70%
8	600	670.13	70.13	11.69%
9	600	670	70.00	11.67%
Average				**11.84%**
Standard Devation				0.00285
t-test			**p-value**	1.92E-20

FIG. 2. Data collected for the C++ Timer.

C# Timer				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	65.71	5.71	9.52%
2	60	65.74	5.74	9.57%
3	60	65.58	5.58	9.30%
4	180	195.13	15.13	8.41%
5	180	196.12	16.12	8.96%
6	180	196.31	16.31	9.06%
7	600	654.11	54.11	9.02%
8	600	653.3	53.30	8.88%
9	600	653.49	53.49	8.92%
Average				**9.07%**
Standard Deviation				0.00356
t-test			**p-value**	9.57E-13

FIG. 3. Data collected for the C# Timer.

the claim. The Python Timer is not accurate.

Python Timer				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	75.53	15.53	25.88%
2	60	75.47	15.47	25.78%
3	60	75.27	15.27	25.45%
4	180	226.36	46.36	25.76%
5	180	221.25	41.25	22.92%
6	180	219.83	39.83	22.13%
7	600	732.79	132.79	22.13%
8	600	733.61	133.61	22.27%
9	600	733	133.00	22.17%
Average				**23.83%**
Standard Deviation:				0.01809
t-test			**p-value**	9.42E-11

FIG. 4. Data collected for the Python Timer.

E. Analysis

Although the Robofest Flash timer is inaccurate, it is still more accurate than timers made in other languages. However, this analysis must take into account the discrepancies caused by battery level and possible error introduced by running the Robofest timer in a .exe executable file, and running other language programs in their respective Development Environments.

Regardless of error, the central problem still remains: The timers are inaccurate. To correct for this issue, this study will test three methods. First, a series of trials will be made after all CPU power is diverted into the timer program. Second, a new program will be developed to force the timer to account for error. Both tests will use C++, as the deviation of C++ is a decent median to test with. Thirdly, two Online Timers will be tested.

IV. FINDING A SOLUTION

Subsequent testing was performed on a second computer system, as the first was malfunctioning. While this may have altered actual data, I do not believe this affects our conclusion significantly.

A. CPU Diversion

In this series of tests, before the program was given the direction to start counting down, the program was given High CPU priority, and access to all cores. This was to prevent the program - which was already very small - from lagging due to insufficient resources. This seems to have had no effect on deviation, as is shown in Figure 5. Again, while this test was run on a different computer, the fact that the real time is far different from the expected time means that this solution does not work to solve this issue. Furthermore, there are less tests, but it seems that they are sufficient to reach a conclusion.

C++ Timer (Diverted CPU)				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	71.06	11.06	18.43%
2	60	70.5	10.50	17.50%
3	60	70.63	10.63	17.72%
4	180	211.64	31.64	17.58%
5	180	211.95	31.95	17.75%
6	180	211.11	31.11	17.28%
Average				**17.71%**
Standard Devation				0.00392
t-test			**p-value**	5.71E-10

FIG. 5. Data collected for the C++ Timer with a diverted CPU.

B. Compensation

For this series of tests, the c++ program was altered to include a "setup" phase, where upon starting, the program would automatically count down from 10 to figure out the factor of deviation, and change the waiting time to account for this. Theoretically, it would mean that each on-screen second would actually take a little bit less than a second, and the delay would make it count as one full second. I only used three tests, as shown in Figure 6. However, I do not think that more tests are necessary to reach a conclusion that this compensation did not entirely correct for error. The reason that this did not entirely correct for error is because it seems to be different depending on how much time the program is given.

C++ Timer (Compensated Time)				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	58.69	-1.31	-2.18%
2	60	58.41	-1.59	-2.65%
3	60	59.44	-0.56	-0.93%
Average				**-1.92%**
Standard Devation				0.00888
t-test			**p-value**	3.57E-06

FIG. 6. Data collected for the C++ Timer with Compensation.

C. Online Timers

If visual timers running directly on the computer are not satisfactory, online timers, as they run using internet time, may be better.

1. vClock

The first Online Timer I tried was vClock, found at "vclock.com", using the same procedures as before. Figure 7 shows the data collected for vClock. The timer, on average, takes 0.68% longer than real time, with a standard deviation of 0.549%. A t-test reveals a very small p-value, which supports the claim that it is not accurate at a 95% confidence level. However, the magnitude of difference is generally even and does not show a progression. This leads me to believe that this was caused by human reaction time, rather than timer error. Therefore, the vClock is a usable alternative to the Robofest timer.

2. Google Timer

The first Online Timer I tried was the Google Search Timer, which can be found by searching for "timer" into Google, using the same procedures as before. Figure 8 shows the data collected for the Google Search Timer. The timer, on average, takes 0.52% longer than real time, with a standard deviation of 0.474%. A t-test reveals a p-value on the order of 10^{-3}, which supports the claim that it is not accurate at a 95% confidence level. However, the magnitude of difference is generally even and even decreases as the tested time increases.

vclock				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	60.72	0.72	1.20%
2	60	60.85	0.85	1.42%
3	60	60.87	0.87	1.45%
4	180	181.35	1.35	0.75%
5	180	180.87	0.87	0.48%
6	180	180.83	0.83	0.46%
7	600	601.07	1.07	0.18%
8	600	600.36	0.36	0.06%
9	600	600.94	0.94	0.16%
Average				**0.68%**
Standard Deviation:				0.00549
t-test			**p-value**	Very Small

FIG. 7. Data collected for the vClock timer.

This leads me to believe that this was caused by human reaction time, rather than timer error. Therefore, the Google Search Timer is a usable alternative to the Robofest timer. The Google Search Timer is also the optimal timer to use.

Google Timer				
Trial	**Time (s)**	**Actual Time (s)**	**Difference (s)**	**% change**
1	60	60.8	0.80	1.33%
2	60	60.56	0.56	0.93%
3	60	60.63	0.63	1.05%
4	180	180.63	0.63	0.35%
5	180	180.46	0.46	0.26%
6	180	180.95	0.95	0.53%
7	600	600.6	0.60	0.10%
8	600	600.47	0.47	0.08%
9	600	600.37	0.37	0.06%
Average				**0.52%**
Standard Deviation:				0.00474
t-test			**p-value**	5.50E-03

FIG. 8. Data collected for the Google timer.

V. EXPLANATION

This raises an interesting question: What causes this deviation? I started out by trying another strategy to correct for it. I tried to see if I could get a good enough trend line to give the program to calculate a compensatory factor to correct for the error. This is based on the second test, which indicates to me that there should be some kind of growth trend the time being timed increases.

First, I tested to see whether there was a good pattern by making a new program that would test the timer for all times between 1 second and 30 seconds. It would test each time three times, and create an average factor. This is the blue plot in Figure 9. The orange plot, on the other hand, is the same test but without outputting time remaining on the screen. The timer is thus free to act without I/O. Although the former test gives an average factor of 1.094, the latter gives a factor of 1.00056, meaning that the reason behind the discrepancy is the Input and Output time, which are not included in the "waiting" processes. It is akin to writing out the time by hand, and then waiting a full second, not taking into account the time taken to write out the number.

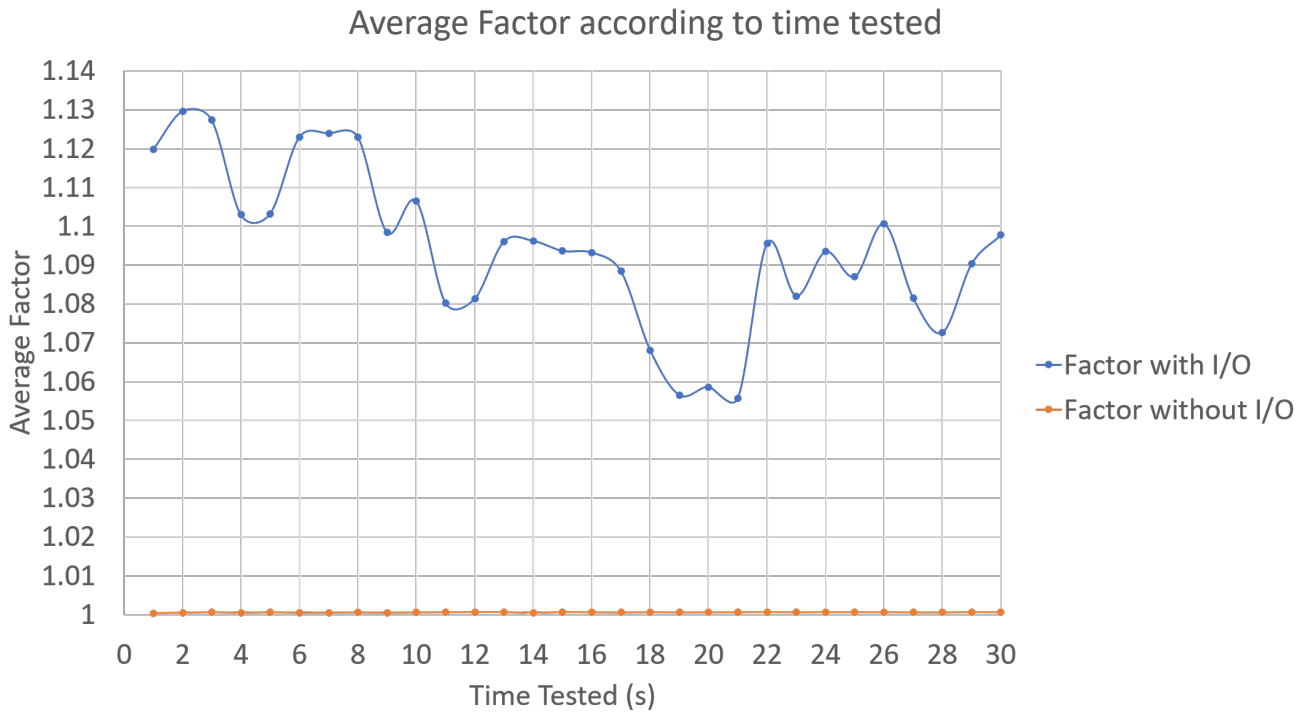

FIG. 9. Compared factors with and without Input/Output.