Genetic Algorithm based PI controller tuning with stability analysis for Linux ptp4l optimization

Milena Olech
Intel Technology Poland
Gdańsk, Poland
milena.olech@intel.com

Marta A Plantykow
Synopsys
Gdańsk, Poland
pmarta@synopsys.com

Maciej Machnikowski
Nvidia
Gdańsk, Poland
maciej@machnikowski.net

Abstract

This paper presents the research results of ptp4l Proportional-Integral (PI) controller tuning using a Genetic Algorithm. Furthermore, it describes recent advancements in the PTP-optimization framework.

PI controller optimization is required to meet the strict phase synchronization industry standards introduced by the LTE E-ULTRA TDD, 5G, and O-RAN technologies.

The ptp4l is a part of the Linux PTP project and implements Precision Time Protocol (PTP) according to IEEE 1588 standard for Linux. The tool implements a PI controller and an API to tune its Proportional (K_p) and Integral K_i) terms.

The developed framework implements the Genetic Algorithm (GA), a stochastic algorithm used in Artificial Intelligence (AI), to optimize the parameters of the PI controller. Stability condition checks are incorporated into the framework to accelerate the discovery of optimal and stable outcomes.

The enhancements resulting from utilizing the tool are presented in the article.

This paper continues research for tuning the Proportional-Integral (PI) controller configurable in the phc2sys application using the Genetic Algorithm presented on Netdev 0x15.

Introduction

Precision Time Protocol is a high-accuracy method for clock synchronization over a network, allowing for precise timekeeping across distributed systems. It is defined by the IEEE 1588 standard and is implemented in Linux OS by the linuxptp toolset, which consists of four main tools: ptp4l, ts2phc, phc2sys, and pmc.

The linuxptp toolset implements the PI controller that continuously measures system output and produces a control signal that aims to minimize the difference between the actual output of the system at a given point in time and the system's desired set-point.

The characteristic of the system under control determines the effect of each of the terms on the response. K_p, K_i, and K_d gains shall be adjusted to fine-tune the overall system's performance.

PTP-optimization framework implements a Genetic Algorithm (GA), a stochastic algorithm used in Artificial Intelligence (AI), to optimize the parameters of the PI controller.

Improvements in the PTP-Optimization framework are presented in this work. This paper also covers the evaluation of the influence of the PTP-Optimization framework on ptp4l.

The paper is divided into four subsections.

The Related work section describes the basics of Precision Time Protocol and PTP on Linux implementations. It also introduces Proportional-Integral-Derivative controllers, PID Tuning, control system, control system stability, and the Linux PTP Control system stability. At the end of this section, possible Data evaluation methods are described.

The second section describes the improvements in the developed framework, such as new methods for usage, stability verification, new applications for tests (ptp4l), changes in the workflow, and logging.

The third section describes the research Results.

Finally, the last section describes the Future Work.

Related work

Precision Time Protocol

The IEEE 1588 standard defines a method for precise computer synchronization over a Local Area Network (LAN), also known as Precision Time Protocol. This solution can synchronize clocks to less than 100 nanoseconds in a network designed according to the standard. (PTP)

PTP Synchronization messages, such as Sync, FollowUp, DelayReq, and DelayResp, are exchanged between the Grandmaster or a timeTransmitter and instance synchronizing to the provider (timeReceiver). Exactly one Grandmaster and at least one timeReceiver are required for LAN clock synchronization.

The timeTransmitter broadcasts the Sync message, which, upon arrival, is timestamped on the timeReceiver's end. Next, the timeTransmitter optionally sends a message that contains the timestamp representing the time when the previous message left the timeTransmitter (FollowUp message). Based on that, the timeReceiver calculates the offset between its timer and the timeTransmitter's.

In the next step, the timeReceiver sends a DelayReq message to the timeTransmitter to calculate the path delay, saving the transmission time. Upon arrival, the timeTransmitter timestamps the message and sends it back to the timeReceiver as a DelayResp message, where the path delay is calculated.

Based on all four timestamps, the timeReceiver calculates path delay compensation, which is crucial for synchronization. (Pla21)

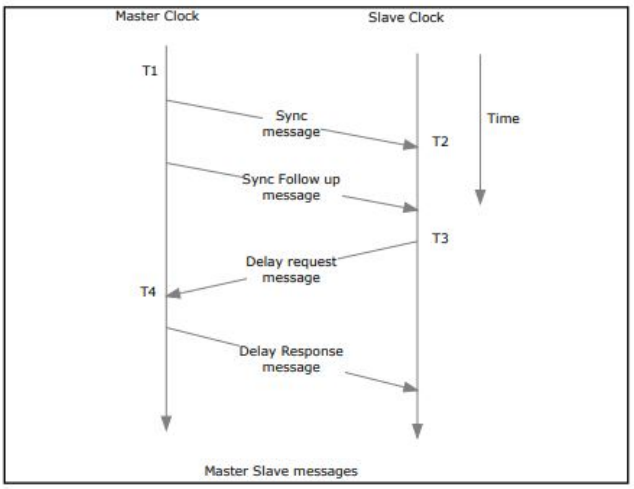

Figure 1: PTP Timestamps (PTP)

Linux PTP (Pla21)

The Linux PTP project provides an implementation of 1588 on Linux. It implements five main tools:

- ptp4l that can synchronize the PTP Hardware Clock (PHC) in the NIC Network Interface Card to the timeTransmitter time in a hardware-accelerated case. For SW usage, the application synchronizes the system clock to the time obtained from the Grandmaster.
- ts2phc that is used to synchronize PHC clocks to the external events coming from external sources, such as a 1PPS signal from the GNSS (Global Navigation Satelite System) receiver.
- phc2sys that synchronizes two (or more) POSIX clocks of choice. For example, it can synchronize the system clock to the PHC.
- pmc - PTP Management Client - that implements a PTP management client according to IEEE 1588 protocol and allows configuring ptp4l in a run-time.
- phc_ctl - is a program that can directly control a PHC clock device (man).

ptp4l

ptp4l is a command-line tool that implements PTP protocol according to the IEEE 1588. The tool generates PTP traffic and controls the timeReceiver's clock, such as a PTP Hardware Clock (PHC), setting its time and frequency to follow the timeTransmitter.

It can be used to synchronize multiple clocks and can be considered both a source (timeTransmitter or Grandmaster clock) and the consumer (timeReceiver clock) of the time packets.

If the underlying hardware supports it, hardware timestamping is used by default.

When the connection between the timeTransmitter and the timeReceiver is established, the application provides the output with thorough precision tracking.

To connect the timeTransmitter and the timeReceiver, the ptp4l application must be initiated on each system, with the timeReceiver optionally activating the clientOnly mode by appending the -s option. The tool can operate under various network transport modes, namely L2, IPv4, and IPv6. The L2 mode elects the IEEE 802.3 network transport. The -4 option enables UDP IPv4 network transport, while the -6 option is used for UDP IPv6.

The s0, s1, s2, and s3 values represent the different states of the clock servo: s0 is unlocked, s1 is the clock step, s2 is locked, and s3 is locked within the predefined range. If the servo is locked, the clock will not be stepped but slowly adjusted by changing its frequency.

The path delay value shows the estimated delay of the synchronization messages sent from the timeTransmitter (expressed in nanoseconds).

The freq value indicates the frequency adjustment of the clock (the currency is part per billion, ppb).

Proportional-Integral-Derivative controllers (Ins)

Proportional-Integral (PI) and Proportional-Integral-Derivative (PID) controllers are the most adopted controllers in the whole industry. As of 2012, in process control applications, more than 95% of the controllers were PI or PID type. (Vil12)

This type of controller continuously measures and adjusts the system's output to match the desired set point. Without prior knowledge of the system, the PI(D) controller produces a control signal to minimize the difference between the system's output at a given time and the system's desired set point.

Figure 2 presents a schematic representation of a general PID control loop in its most general form.

In a continuous feedback loop, the comparator block subtracts system output $y(t)$ from the system set-point $r(t)$ and provides error signal value $e(t)$ (difference between desired and actual value) to the Loop Filter block (Proportional, Integral, Derivative sub-blocks), where the error signal is used to calculate the control signal $u(t)$. The control signal $u(t)$ is then applied to the system (controlled object), causing a change in the system's output.

The control system shall consider not only the current error value $e(t)$ but also its accumulation over time (the integral sub-block represents that) and its future tendency (represented by the derivative at time t), as shown in 3.

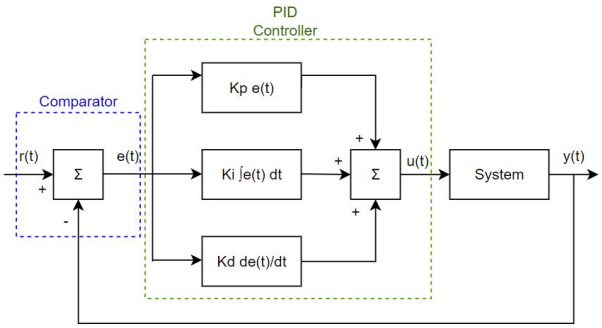

Figure 2: Schematic representation of a general PID control loop

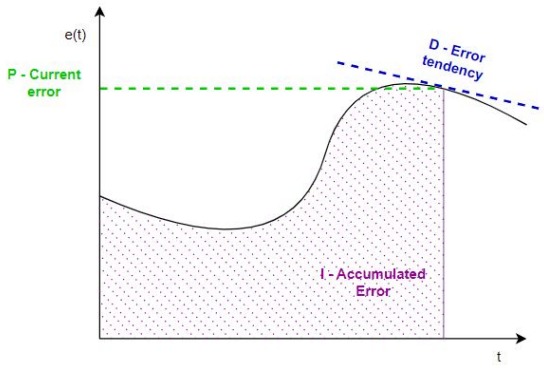

Figure 3: Example error function in time with highlighted contributions of each of the sub-blocks

The following equation represents the complete control function in its most general form:

$$u(t) = u_p(t)+u_i(t)+u_d(t) = k_p e(t)+K_i \int_0^t e(\tau)+K_d \frac{d}{dt} e(t) \quad (1)$$

Each of the PID controllers sub-block has a different role:

- Proportional term - Brings the output to the set point by applying correction proportional to the error amplitude. It can not nullify the error as it requires a non-zero error to generate its output.
- Integral term - Applies correction proportional to the error's time integral (history). It can not nullify the error even under a zero-error condition at present. This term enables the controller to bring the system precisely to the required set point.
- Differential term - By applying a correction proportional to the time derivative of the error, this term provides control over the error tendency. The derivative term can be omitted in real-world applications of PIDs, as if the reference value changes rapidly, the derivative of the error becomes very large and causes the PID controller to undergo an abrupt change that can result in instabilities or oscillations in the control loop.

Figures 4, 5 and 6 present the influence of each PID controller term on the system under control behavior.

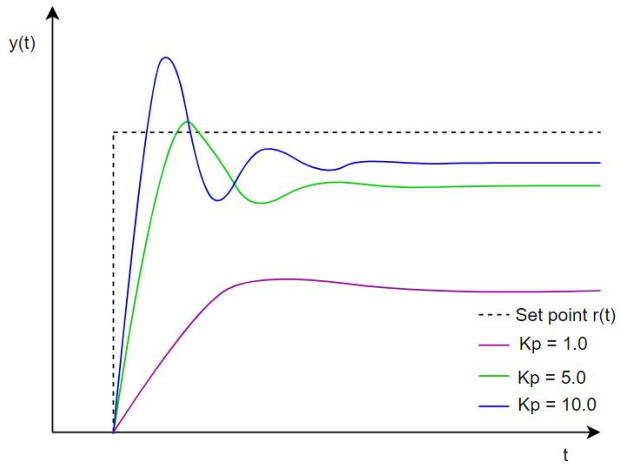

Figure 4: P term influence on the system's behavior

As Figure 4 shows, the proportional term part of the control signal $u(t)$ value is directly proportional to the error signal value $e(t)$. Increasing the K_p gain reduces the rise time, but the error never approaches zero. The system's output may oscillate if the K_p gain is too high.

As Figure 5 shows, the integral term part of the control signal $u(t)$ value is proportional to the time integral of the error. Increasing the K_i parameter increases the

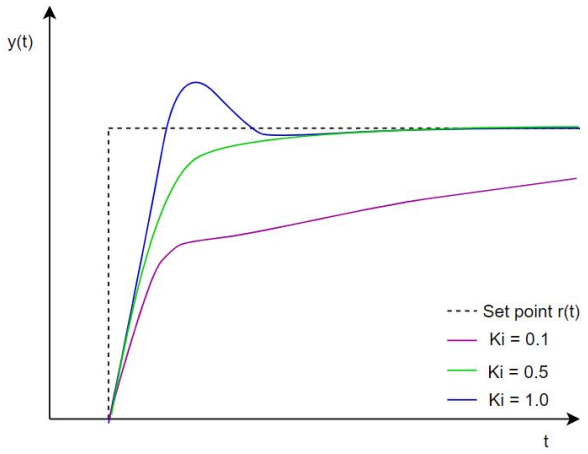

Figure 5: I term influence on the system's behavior

contribution of the accumulated error over time to the control signal and, as a result, reduces the error elimination time. The system's output may oscillate around the set point or lead to an overshoot if the K_i gain is too high.

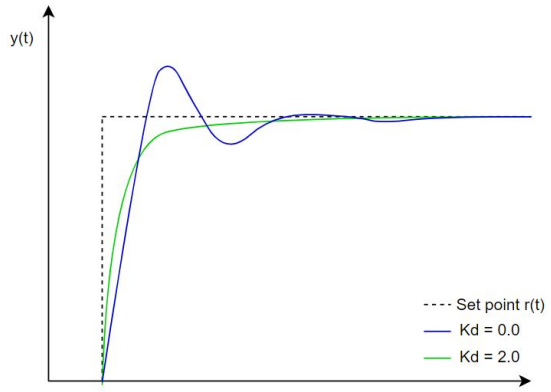

Figure 6: D term influence on the system's behavior

As Figure 6 shows, the derivative term part controls the error tendency. This term aims to anticipate the changes in the error signal: if the error shows an upward trend, the derivative action tries to compensate without waiting for the error to become significant (proportional action) or for it to persist for some time (integral action).

The characteristic of the system under control determines the effect of each of the terms on the response. K_p, K_i, and K_d gains shall be adjusted to fine-tune the overall system's performance.

Some systems may perform better with only one of two of the three sub-blocks of the PID controller. PI controller, for example, is common for applications prioritizing steady-state error elimination and stability over fast response times.

PID tuning

As described in the previous subsection, each PID controller term significantly influences the system's response. PID tuning manipulates the Kp, Ki, and Kd gains to achieve satisfactory response [1]from the control process (Jos18).

There are several well-established methods used for PID controller tuning, for example, Ziegler-Nicholsmethod, (NNZ42), Cohen-Coon method (Jos18), Asrom and Hagglung method, (Ast93), or trial and error method (too). Besides that, Genetic Algorithms (GA) have proven to be capable of locating high-performance areas in complex domains, including finding optimal settings for PID controllers. (Pla21).

Each method carries its advantages and trade-offs, requiring a judicious selection based on the specific characteristics and requirements of the system at hand.

Control system (Ele)

A control system is a system that regulates the behavior of other devices through control loops to achieve the desired result. Control loops are processes designed to maintain a process variable, or in other words, to control a system's output. Two types of control loops define two main types of control systems:

- **Open control loops** - The system's output is not directly measured or compared to the desired set-point. The control action is determined exclusively by the input and the system's characteristics. Open control loops are used in **open-loop control systems**.
- **Closed control loops** - involve measuring the system's output and comparing it to the set point. The system's input, output, and characteristics determine the control action. Closed loops are used in **closed-loop control systems**.

PID controller-based control systems implement closed control loops.

Control system stability (Nis11)

Stability is the most essential specification of a control system. Each linear [2], time-invariant [3] system's response can be described as a sum of the forced and natural responses:

$$y(t) = y_{forced}(t) + y_{natural}(t) \tag{2}$$

The natural response of the system is the behavior of the system that arises solely from its initial conditions and inherent dynamics. It is the way the system behaves without any external input or disturbance. On the

[1]Satisfactory response - a response that fulfills requirements related to the speed of response, accuracy, and stability.

[2]A **linear control system** is a type of control system in which the relationship between the input and the output of the system can be described by linear mathematical equations.

[3]A **time-invariant control system** is a type of dynamic system in which the system's parameters, characteristics, and behavior do not change over time.

other hand, the forced response is the system's behavior resulting from an external input or disturbance.

Based on the response characteristic, each control system can be defined as stable, unstable, or marginally stable, where each of the following terms is defined as:

- A linear, time-invariant system is **stable** if the system's natural response eventually reaches zero as the time approaches infinity.
- A linear, time-invariant system is **unstable** if the system's natural response grows without bound as the time approaches infinity.
- A linear, time-invariant system is **marginally-stable** if the system's natural response remains constant or oscillates as time approaches infinity.

When analyzing the total system's response exclusively, separating the natural response from the forced response is difficult. However, if the system's input is bounded and the total system's response is not approaching infinity (as time approaches infinity), the natural response is also not approaching infinity. On the other hand, if the system's input is unbounded, the total response is also unbounded, making it impossible to arrive at any conclusion about the system's stability. Based on that, **a stable system** can also be defined as every system that yields a bounded output for each bounded input. This statement is called the bounded-input, bounded-output (BIBO) definition of stability.

At the same time, if the system's input is bounded, but the total system's response is unbounded, the system is unstable since we can conclude that the natural response approaches infinity when the time approaches infinity. Based on that, **an unstable system** can also be defined as every system that, for any bounded input, yields a unbounded output.

Figure 7 presents typical responses of a system.

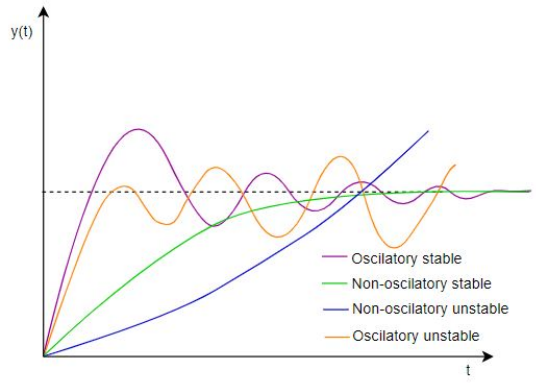

Figure 7: Typical responses of a control system (EEE)

Linux PTP Control system stability (Edi06)

Linux PTP suite implements IEEE 1588 using the Proportional-Integral (PI) controller in the clock's control system. Figure 8 presents a servo model in a timeReceiver clock.

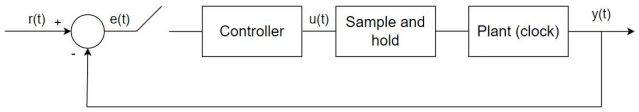

Figure 8: Model of the servo in a timeReceiver clock

The system consists of a controller, sample and hold, and a plant representing the clock. As for the general PID control loop described in section , in this control system error signal $e(t)$, that is a difference between the set-point (time of the remote master clock) $r(t)$ and the current system output value (time of the local slave clock) $y(t)$ is forwarded to the controller to produce control signal $u(t)$. The presented control loop is fundamentally an example of a discrete control system, as the error signal $e(t)$ is sampled with a period T corresponding to the PTP synchronization interval.

The control system performance depends on PI controller gains - K_p and K_i. John C. Edison (Edi06) performed an exhaustive analysis of the PTP PI controller's gains influence on control system stability. A two-dimensional plane representing the relationship between parameters K_p and K_i can be divided into three sections, each representing a different type of stability based on the roots of characteristic equation [4]:

1. **Roots of the characteristic equation are complex**
   Conditions for stability are described by equations 3-5:
$$(P+I)^2 < 4I \tag{3}$$
$$0 \leq I \leq 4 \tag{4}$$
$$0 \leq P \leq 1 \tag{5}$$
2. **Roots of the characteristic equation are real and equal**
   Conditions for stability are described by equations 6-8:
$$(P+I)^2 = 4I \tag{6}$$
$$0 \leq I \leq 4 \tag{7}$$
$$0 \leq P \leq 1 \tag{8}$$
3. **Roots of the characteristic equation are real and unequal**
   Conditions for stability are described by equations 9-10:
$$2P = 4 - I \tag{9}$$
$$0 \leq P \leq 2 \tag{10}$$

Picture 9 represents final stability regions.

[4] Characteristic equation - polynomial equation that arises from the denominator of the system's transfer function when it's expressed in terms of the Laplace variable 's'.

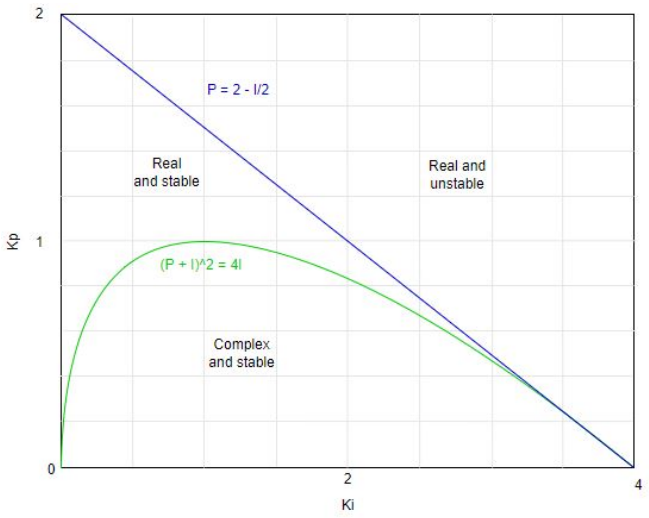

Figure 9: Stability regions for Kp and Ki parameters

Linuxptp PI controller limitations

Software implementation of PI controller in the linuxptp further limits the two-dimensional plane representing the relationship between parameters K_p and K_i.

Regardless of the input parameter value passed as an argument, its maximum value will be trimmed to hard-coded values. The maximum allowed value for the proportional gain K_p is 1.0, whereas the maximum permissible value for the integral gain K_i is 2.0.

Picture 10 presents the linuxptp controller limitations of K_p and K_i gains (dashed) along with previously introduced system stability regions.

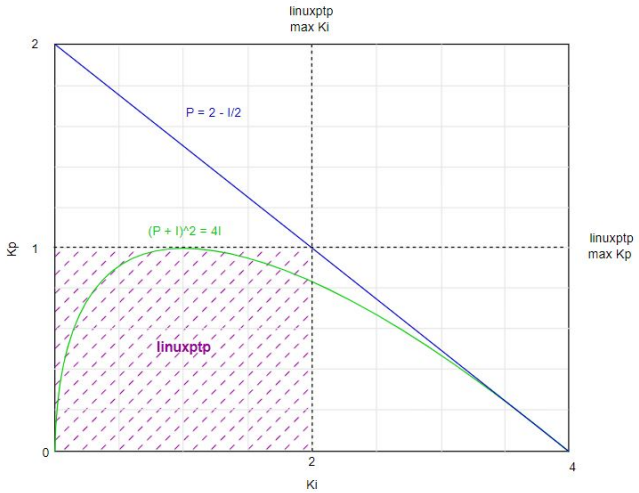

Figure 10: Linuxptp stability regions for Kp and Ki parameters

Data evaluation methods (Pla21)

Measures of the average error that depicts model performance are based on summaries of error values in time e_i $(i = 1, 2, ..., n)$. Various metrics take into account different aspects of the measured values, such as:

- $RMSE$ - Root Mean Squared Error - described by the formula 11. The sensitivity of $RMSE$ is caused by the fact that each error e_i is squared, which strives for more significant influence on the final result.

$$RMSE = [n^{-1} \sum_{i=1}^{n} |e_i|^2]^{\frac{1}{2}} \tag{11}$$

- Mean Absolute Error - MAE, described by the formula 12. This measure signs the magnitude of the errors by applying absolute value.

$$MAE = n^{-1} \sum_{i=1}^{n} |e_i| \tag{12}$$

.

- MBE is the average error defined as Mean Bias Error - MBE and is described by the formula 13. MBE value mainly indicates average model bias rather than provide error magnitude.

$$MBE = n^{-1} \sum_{i=1}^{n} e_i \tag{13}$$

The calculation of MAE consists of two parts. The first part is summing the absolute values of the errors, then dividing the total error by n. The calculation of RMSE involves three steps. First, total square error is obtained as a sum of individual squared errors. This means that significant errors have a greater influence on the total than minor errors. Subsequently, the total square error is divided by n, equal to the Mean Squared Error - MSE. The last step is to take RMSE as the square root of the MSE.

PTP-optimization framework

This section describes the improvements in the PTP-optimization [5] framework introduced during the Netdev 0x16 conference [6] (Pla21).

The framework's primary goal is to optimize the time synchronization using the linuxptp project by optimizing the Proportional-Integral (PI) servo parameters using the Genetic Algorithm (GA) described in the previous publication.

Usage

The developed framework is highly flexible and configurable through a configuration file. Table PTP-Optimization framework configureme file configurables

[5] https://github.com/mplantykow/PTP-Optimization/
[6] https://github.com/intel/PTP-optimization

(appendix) presents all configurable parameters of the framework along with their function description.

The main.py file must be executed to start the automated PTP PI controller gains optimization procedure. The parameter that determines the interface to work on $--i$ must be given as an argument to the main.py script along with the parameter that determines the test time in seconds $--t$.

```
./python3 main.py --i enp0s0 --t time
```

Within the framework, the system explores various potential values of Kp and Ki for the controller by testing different combinations over a defined number of epochs. As a result, sets of Kps and Kis, along with the scores (calculated based on the chosen metric), are generated.

Stability verification

PTP-Optimization framework was extended with the feature that reduces the possible search space for optimal values to the area defined by an object's stability analysis described in section .

In the previous implementation, the possible search area was limited only by the input parameters, namely $--max\_kp$ and $--max\_ki$, which were set to 15 by default.

The current implementation provides three possible parameters for the stability verification:

- **Complex** - reduces the possible search space for optimal values to the "Complex and Stable" area on figure 9. This area describes the case when the roots of the characteristic equation are complex and is represented by equations 3-5.
- **Real** - reduces the possible search space for optimal values to the "Real and Stable" area on figure 9. This area describes the case when the roots of the characteristic equation are real and unequal and is represented by equations 9-10.
- **False** - reduces the possible search space for optimal values to the area limited by $--gen\_max\_kp$ and $--gen\_max\_ki$ (fallback to the old implementation).

The choice of which stability type to analyze is determined through configurations in the source file.

If stability verification is enabled and, as a result of crossover or mutation procedures, the resulting creature goes beyond the stability boundaries of the object, the values of Kp and Ki of the creature are estimated to the nearest point within the selected stability region. The estimation operation is performed in steps whose size is determined by a parameter named *reduction_determinant* (see table 5.

By default, due to the linuxptp PI controller limitations described in section Linuxptp PI controller limitations), regardless of the actual parameters passed as arguments for K_p and K_i, their maximum value will be limited for the proportional gain Kp to 1.0 and 2.0 for the integral gain Ki, unless the linuxptp is modified and recompiled accordingly.

The goal of enabling this feature was to allow the user to exclude unstable combinations upfront to reduce the optimal solution search time.

New application - ptp4l

The new framework allows optimizing the second application from the linuxptp project - ptp4l. The preferred application shall be specified in the config file.

The ptp4l tool offers a lot of configurable variables that affect the time to achieve final synchronization. The implemented framework uses the following command:

```
./ptp4l -i {interface} -m -2 -s
        --tx_timestamp_timeout 100
        --pi_proportional_const {P}
        --pi_integral_const {I}
```

Where tx_timestamp_ timeout indicates the number of milliseconds to poll waiting for the transmit timestamp from the kernel.

One of the primary assumptions of the framework is that the data provided to the genetic algorithm are collected using a solid clock with stable frequency. An additional step has been applied to eliminate the noise that may negatively influence the data. The previously described ptp4l command with default *Kp=0.3* and *Ki=0.7* values has been performed for 60 seconds.

Console output and logging

PTP-Optimization framework was improved with the new debug features - more meaningful framework output, extended logging, and graphs.

With the *graph_per_epoch* flag enabled, the framework can create graphs that illustrate generated Kp and Ki values (Figure 11) and the change of the score in the following epochs (Figure 12). The first chart combines the elite creature and the epoch number, allowing us to interpret the second diagram correctly.

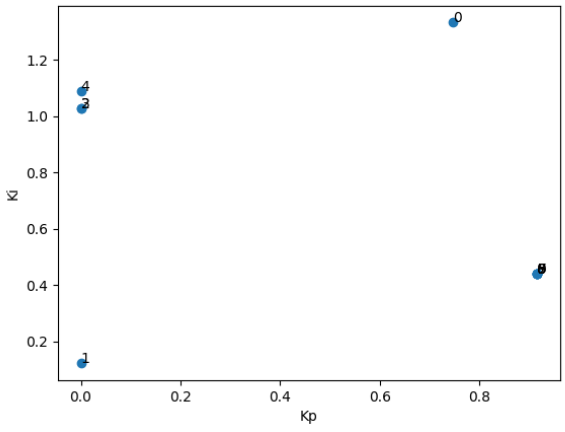

Figure 11: Kp and Ki pais with assigned epoch number

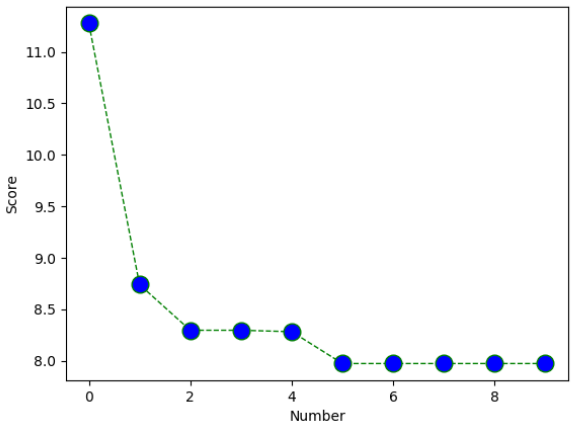

Figure 12: Scores per epoch

Besides that, at the end of each run, the framework generates the plot that represents colored Kp and Ki values, where colors indicate the assigned metric score (Figure 13).

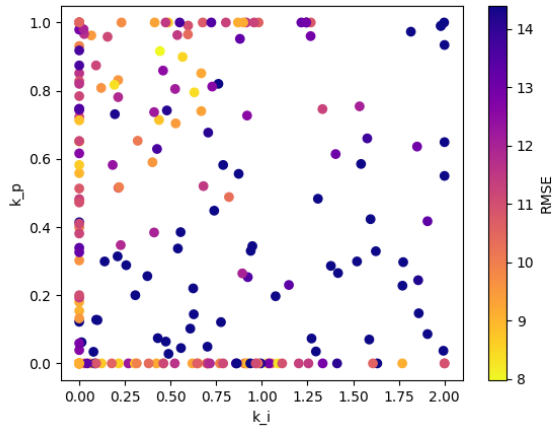

Figure 13: Combined creature and score graph

Furthermore, new csv files are generated at the end of the calculations. One collects data about all investigated Kp and Ki pairs and metric scores, and the second provides information about best-scored creatures with their rates.

These improvements alleviate data interpretation and adjust the framework according to the specific use case.

Logging during the operation was extended to provide a better understanding of the framework's learning process. It is also possible to follow the progress by tracking printed % of improvement.

Data evaluation

This work aims to synchronize our timeTransmitter and timeReceiver PHC timers as closely as possible. Modifying Kp and Ki allows us to achieve that goal more quickly. The ideal scenario is to have an offset between the leader and the timeReceiver equal to 0. Evaluation methods have been applied to calculate the deviation from the perfect scenario.

The developed framework gives a possibility to choose three different evaluation metrics: MSE, RMSE, and MAE. In this work, two are being considered - RMSE and MAE. The reason behind that is that the metric's influence on the framework determines which pairs of Kp and Ki generate the best output, where the best means output with the smallest deviation from ideal point 0. Having that, RMSE and MSE are described by the following formulas:

$$RMSE = [n^{-1} \sum_{i=1}^{n} |e_i|^2]^{\frac{1}{2}} \tag{14}$$

$$MSE = [n^{-1} \sum_{i=1}^{n} |e_i|^2] \tag{15}$$

In all cases, we will evaluate each pair of Kp and Ki precisely in the same manner, maintaining the proportions resulting from the square root. That is why the difference in the use of metrics is invisible from the perspective of the framework operation.

Test setup

The test setup consisted of two Raspberry Pi Compute Modules 4 (CM4): Grandmaster and the DUT. Both modules ran the Raspberry Pi OS "bullseye" with kernel 6.1.54-v8+ and linuxptp 3.1 installed. Modules were connected back to back with an Ethernet cable.

The Grandmaster was synchronized to the uBlox NEO-M9N GNSS receiver using NMEA data and the 1PPS signal using the ts2phc tool.

The DUT was running the framework described in this document.

Both sides had NTP synchronization disabled using timedatectl set-ntp off and systemctl disable systemd-timesyncd commands.

Results

The team conducted experiments to optimize the PI controller parameters implemented in Linux PTP. This section presents the results of individual studies conducted to assess the changes introduced in the framework. It consists of three subsections, each representing a different test and its goal.

Study of the time required to ensure test repeatability

In this section, we present the results of the time required to ensure test repeatability study. The ability to replicate

test conditions reliably is paramount in any experimental design.

We conducted sixty-minute tests of the 'ptp4l' application to define the time required to achieve repeatable conditions. We utilized the PI controller, adjusting the frequency (with default Kp 0.7 and Ki 0.3 parameters), and measured the offset value in time. Each test was respectively repeated ten times. Running metrics were calculated based on the value of the offset.

This study aimed to find the time when metrics would reach similar values. Picture 14 presents the convergence of running metrics in time. Additionally, Table 1 presents Root Mean Square Error Values (RMSE) at points in time, along with its average value and the standard deviation.

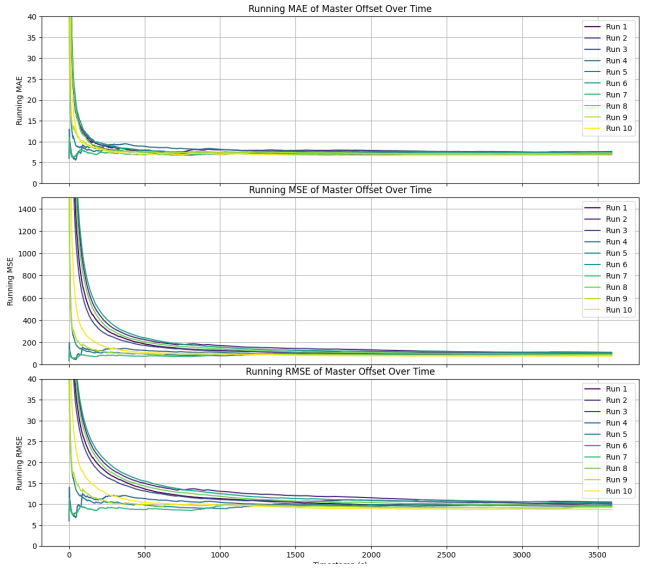

Figure 14: Convergence of running metrics in time

Table 1: Root Mean Square Error (RMSE) at different time points

	RMSE						
Run	30s	90s	150s	300s	600s	1200s	3600s
1	44.85	26.76	21.47	16.53	12.82	10.90	9.65
2	51.88	30.58	24.05	17.96	14.13	12.51	10.49
3	41.13	24.97	20.03	15.65	12.56	10.85	10.16
4	7.22	12.55	11.52	12.05	10.94	10.07	9.51
5	16.07	11.62	10.60	10.45	9.45	9.48	9.88
6	53.51	31.74	25.13	18.73	14.56	11.85	10.26
7	7.53	9.37	8.78	9.06	8.84	9.65	9.41
8	49.24	28.98	22.97	17.26	13.53	11.25	9.58
9	17.69	13.00	11.73	10.15	9.56	9.69	9.25
10	28.64	18.03	14.96	11.85	10.24	9.56	8.74
Ave	31.78	21.06	17.12	13.95	11.66	10.58	9.69
Std	17.56	8.03	5.92	3.44	1.99	1.00	0.49

The metric values over time were collected for the entire test duration. The table 1 presents only selected values, while the value at other time points can be read from the 14.

The RMSE value achieves the highest values varying from 7.22 to 53.51 for the shortest presented period (30s), resulting in 31.78 on average. For this test, the standard deviation value is 17.56. The RMSE values for tests that lasted less than 30 seconds are even higher.

The average value of the metric indicating the error decreases with the increasing duration of the test. The same dependency applies to the standard deviation.

The smallest RMSE values were achieved for the longest test that lasted 3600s (1 hour). The lowest metric score equals 8.74, while the highest equals 10.49. For this test, the average metric value equals 9,69, and the standard deviation achieved 0.49.

As described in section Usage the framework explores various potential values of Kp and Ki for the controller by testing different combinations over a defined number of epochs, conducting tens of experiments in a single run.

Due to that, the 600-second (10-minute) test time was selected as a compromise between test time and test repeatability for further tests. For this test time, the RMSE value is 8.84 - 14.13, achieving 11.66 on average. The standard deviation equals 1.99, over seven times less than the 30s-long tests.

Stability verification checks influence the framework's performance

This test investigates the impact of stability verification on the results. The system's stability is the most important control system's specification. If the system is unstable, its response grows without bounds over time.

This study examines whether stability verification affects the framework performance in terms of results or the time required for their achievement.

In this test, two scenarios were run.

The first tested area is limited to the "Complex and stable" (additionally limited by the linuxptp stability hard limits of $K_p <= 1$ and $K_i <= 2$). As a result, the test limited the search of K_p and K_i to the area described by equations 16 and 17:

$$(P + I)^2 < 4I \tag{16}$$

$$0 < I < 2 \tag{17}$$

Figure 15 presents resulting scatter plot. Each point on the scatter plot presents a single creature described by K_p and K_i values. This test was configured for X initial creatures and five epochs. As a result, Z creatures were tested.

On the figure 15, a clear accumulation of points at the boundary of the examined area is noticeable. This is a result of the algorithm described in section Stability verification, which, where a creature resulting from crossover or mutation moves beyond the defined area, shifts it to the nearest point within the defined range.

Each point is assigned a color that depends on the value of the metric calculated for that point. The points

are yellow for the smallest RMSE values, slightly less than 8. As the metric value increases, the points darken through orange and purple until they reach the blue color, which is defined as the highest metric value exceeding 16. Many points that have achieved a good result (yellow) are clustered at the upper boundary of the permissible range on its left side.

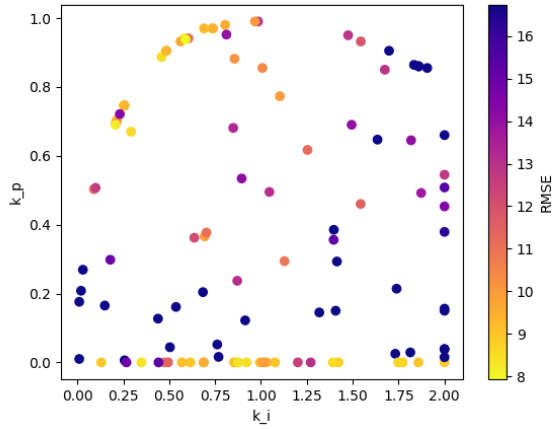

Figure 15: Results scatter plot with limited stability region

The table 2 presents the K_p, K_i, and RMSE values achieved for the best (elite) creature in each epoch. This test error metric value decreases from 9.055 to 7.926, adjusting K_p from 0.7 to 0.938 and K_i from 0.3 to 0.589.

Epoch	K_p	K_i	RMSE
ref	0.7	0.3	9.055
0	0.366	0.693	9.896
1	0.941	0.583	8.247
2	0.941	0.583	8.247
3	0.69	0.207	8.183
4	0.938	0.589	7.926

Table 2: Elite score - limited stability region

In the second run, the complete stability region limited by built-in linuxptp boundaries ($K_p <= 1$, $K_i <= 2$) was tested. Figure 16 presents a scatter plot of the second test. Points on the scatter plot present K_p and K_i pairs and are colored based on the assigned RMSE metric value. The only difference between the first and second tests is the analyzed area of K_p and K_i, which, in this case, has a larger surface area and a rectangular shape.

In the figure 16, an apparent accumulation of points at the boundary of the examined area is noticeable, as in the previous case, this is expected.

RMSE is slightly above 8 (yellow) to over 17 (dark blue) in this test. As for the previous test, many points that have achieved a good result (yellow/orange) are clustered at the upper boundary of the permissible range, on the left side.

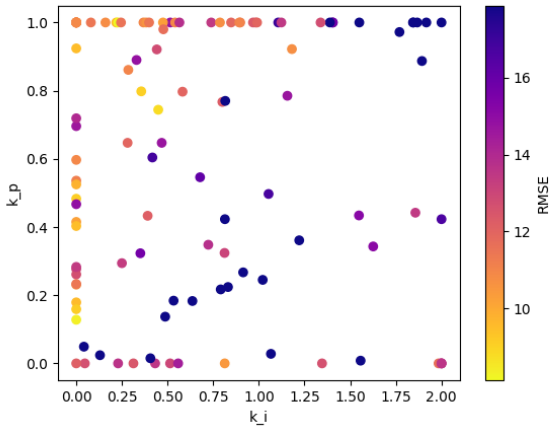

Figure 16: Results scatter plot with unlimited stability region

The table 3 presents the K_p, K_i, and RMSE values achieved for the best (elite) creature in each epoch. This test error metric value goes down from 11.8 to 8.132, increasing K_p from 0.7 to 1 and decreasing K_i from 0.3 to 0.229.

Epoch	K_p	K_i	RMSE
ref	0.7	0.3	11.8
0	0.861	0.285	11.01
1	1	0.219	9.558
2	1	0.229	8.132
3	1	0.229	8.132
4	1	0.229	8.132

Table 3: Elite score - full stability region

In both conducted tests, the lowest metric values, and thus the best results, are achieved when K_p hovers around 1.

In the first test, the values of K_i for which good results are achieved are higher than the default, most likely due solely to the shape of the analyzed area (K_p reaches a value of 1 when K_i is equal to 1). In the case of the second test, it is evident that good results are achieved with K_i slightly lower than the default.

Lower metric values are obtained for a larger K_p than default and a smaller K_i than default. This means the system should rely more on the current error value and be less responsive to accumulated errors.

The accumulation of points characterized by small metric values in the upper left part of the examined space suggests exploring a range broader than the one defined in linuxptp. At the same time, it is evident that the "Complex and stable" area excludes a portion of good

results, so the entire "Real and stable" area should be examined.

Increasing number of epochs

As a final exercise, the ptp4l test response with the PI controller tuned with values returned by GA was compared with the ptp4l test response with the PI controller tuned with default settings (Kp = 0.7 and Ki = 0.3, initial population size = 8). This test was performed to investigate the influence of increasing the number of epochs on results.

Figure 17 and table 4 presents the outcome of running 10 epochs with built-in linuxptp limitations where K_p is smaller than 1 and K_i smaller than 2.

The elite score oscillates between 8 and 11, where the usage of the default values result is approximately 11. Collected data indicates an improvement level equal to 30%.

In this case, the aggregation of points with the lowest metric values is in the upper left area and along Kp and Ki axes. This test confirmed previous observations, where larger K_p and lower K_i bring better results than default values.

Notably, increasing the number of epochs does not significantly lower the creature scores. Table 4 shows that the best-scored coefficients were found after six epochs. It proves that more epochs do not guarantee better test results.

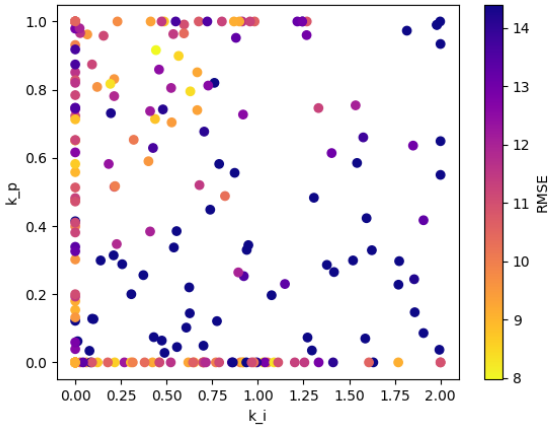

Figure 17: Results scatter plot with ten epochs

Epoch	K_p	K_i	RMSE
ref	0.7	0.3	11.277
0	0.746	1.332	11.277
1	0	0.122	8.741
2	0	1.029	8.298
3	0	1.029	8.298
4	0	1.088	8.284
5	0.916	0.442	7.975
6	0.916	0.442	7.975
7	0.916	0.442	7.975
8	0.916	0.442	7.975
9	0.916	0.442	7.975

Table 4: Elite score - longest run

Conclusions

The established framework offers a user-friendly automated approach for fine-tuning the Proportional-Integral controller integrated within two linuxptp applications: phc2sys and ptp4l. As for phc2sys, the preset parameters employed in the PI controller typically produce satisfactory outcomes. Nevertheless, the framework demonstrated its capability to achieve superior results in all examined scenarios. This study consistently indicated the effectiveness of the developed framework. In most cases, just a few iterations were needed to discover parameters that markedly outperformed the default values. Additionally, it's worth noting that this work comprehensively addresses all aspects outlined in the 'future work' section presented at the previous conference.

References

[Ast93] Tore Hagglund Karl J. Astrom. *PID Contollers, 2nd Edition.* 1993.

[Edi06] John C. Edison. *Measurment, Control and Communication using IEEE 1588.* Springer, 2006.

[EEE] EEEguide.com. Transient response of closed loop drive system. `https://www.eeeguide.com/transient-response-of-closed-loop-drive-system/`. Accessed: 2023-10-8.

[Ele] Electrical4U. Control systems: What are they? `https://www.electrical4u.com/control-system-closed-loop-open-loop-control-system/?utm_content=cmp-true`. Accessed: 2023-10-8.

[Ins] Zurich Instruments. Principles of pid controllers. `https://www.zhinst.com/sites/default/files/documents/2023-07/zi_whitepaper_principles_of_pid_controllers.pdf`. Accessed: 2023-9-13.

[Jos18] E. A Joseph. Cohen-Coon PID Tuning Method: A Better Option to Ziegler Nichols-Pid Tuning Method. 2018.

[man] Debian man pages). `https://manpages.debian.org/unstable/linuxptp/phc_ctl.8.en.html`. Accessed: 2023-10-12.

[Nis11] Norman S. Nise. *Control systems engineering, 6th edition.* John Wiley & Sons, Inc., 2011.

[NNZ42] J.G. Nichols N.B. Ziegler. Optimum settings for automatic controllers. 1942.

[Pla21] Machnikowski Plantykow, Olech. Precision Time Protocol optimization using genetic algorithm. 2021. Accessed: 2023-9-13.

[PTP] Precision time protocol (ptp/ieee-1588). `https://endruntechnologies.com/pdf/PTP-1588.pdf`. Accessed: 2023-9-13.

[too] Inca tools. Explore the 3 pid tuning methods. `https://www.incatools.com/pid-tuning/pid-tuning-methods/`. Accessed: 2023-10-8.

[Vil12] Ramon Vilanova. *PID Control in the Third Millenium.* Springer, 2012.

Table 5: PTP-Optimization framework configureme file configurables

Variable	Default value	Description
debug_level	1	This variable adjusts the level of printed debug log messages; Set to 1 for basic logging, set to 2 for full logging
app	ptp4l	This variable defines which of the PTP applications should be tested. Following arguments are accepted: ptp4l, phc2sys
metric	MAE	This variable defines which of the metrics is used for calculations. Following metrics are accepted: MSE, RMSE, MAE
initial_values	False	Set this option to True to load initial creatures (Kp, Ki) from initial_values.csv file; If set to False, initial creatures will be randomly drawn
graph_per_epoch	False	Set to true to generate a graph with results for each epoch
stability_verification	Real	This parameter determines the stability verification; There are two stability regions defined in (Edi06): Complex (Complex & stable) and Real (Real & stable). This parameter can be also set to False to disable stability verification check. In this case, gen_max_kp and gen_max_ki parameters can be used in order to limit the search area. Note, that default PI controler implementation in linuxptp limits the input parameters as described in section "Linuxptp PI controller limitations".
gen_max_kp_stable_complex	1	Based on (Edi06), it is advised to do not modify
gen_max_kp_stable_real	2	Based on (Edi06), it is advised to do not modify
gen_max_ki_stable	4	Based on (Edi06), it is advised to do not modify
reduction_determinant	0.001	If stability verification is enabled and as a result of crossover or mutation procedures the resulting creature goes beyond the stability boundaries of the object, the values of Kp and Ki of the creature are estimated to the nearest point within the selected stability region. This parameter determines estimation step size.
gen_population_size	8	Genetic algorithm - Initial population size
gen_epochs	8	Genetic algorithm - Number of epochs, minimal accepted value - 1; There is no maximal value
gen_max_kp	5	This parameter limits the algorithm search area by limiting the maximal Kp value that can be used for tests; This parameter is ignored when stability_verification is set to True
gen_max_ki	5	This parameter limits the algorithm search area by limiting the maximal Ki value that can be used for tests; This parameter is ignored when stability_verification is set to True
gen_num_random	2	Genetic algorithm - Number of random parents added to each epoch
gen_num_inherited	5	Genetic algorithm - Number of parents directly replicated to create a new generation
gen_num_replicated	4	Genetic algorithm - Number of parents replicated to create a new generation
gen_mutation_coef	1	Genetic algorithm - Mutation coefficient; determines the influence of mutations on creatures
gen_elite_size	1	Number of elite chromosomes
test_repeted_creatures	False	Set to True to retest repeated creatures, set to False in order to reduce the test time and assign previous value if the same creature shall be tested