

YAPSC Tuning Tool

PID Tuning Guide for YAPSC servo boards

Abstract:

In this document you will see how to setup your YAPSC (V1 or 10V versions) Servomotor Controller using YTT.
This documents covers rapid description of YTT and PID tuning method using the step response tool of YTT.

Table Of Content

1	YTT description.....	4
1.1	“Connection” tab.....	4
1.2	“Parameters” tab.....	4
1.3	“RT Monitor” tab.....	5
2	PID tuning: theory.....	6
2.1	Common terms.....	7
2.2	Stability.....	7
2.3	Overshoot.....	8
3	PID tuning: application.....	8
3.1	Hardware setup.....	8
3.2	PID Tuning.....	9
3.2.1	P tuning.....	10
3.2.2	I tuning.....	10
3.2.3	D tuning.....	10
3.2.4	Sum up.....	10
4	Troubleshooting.....	10
4.1	Inverted polarity.....	10
5	Extra information.....	11
5.1	From steps to rev.....	11
5.2	From steps to mm (in).....	11
5.3	Units per Step.....	11

1 YTT description

YTT is ordered in three tabs: "Connection", "Parameter" and "RT Monitor".

1.1 "Connection" tab

In this tab, you can choose the serial port you want to use to connect to the YAPSC board; along with the model of the board. Currently you can choose between V1 and 10V versions.

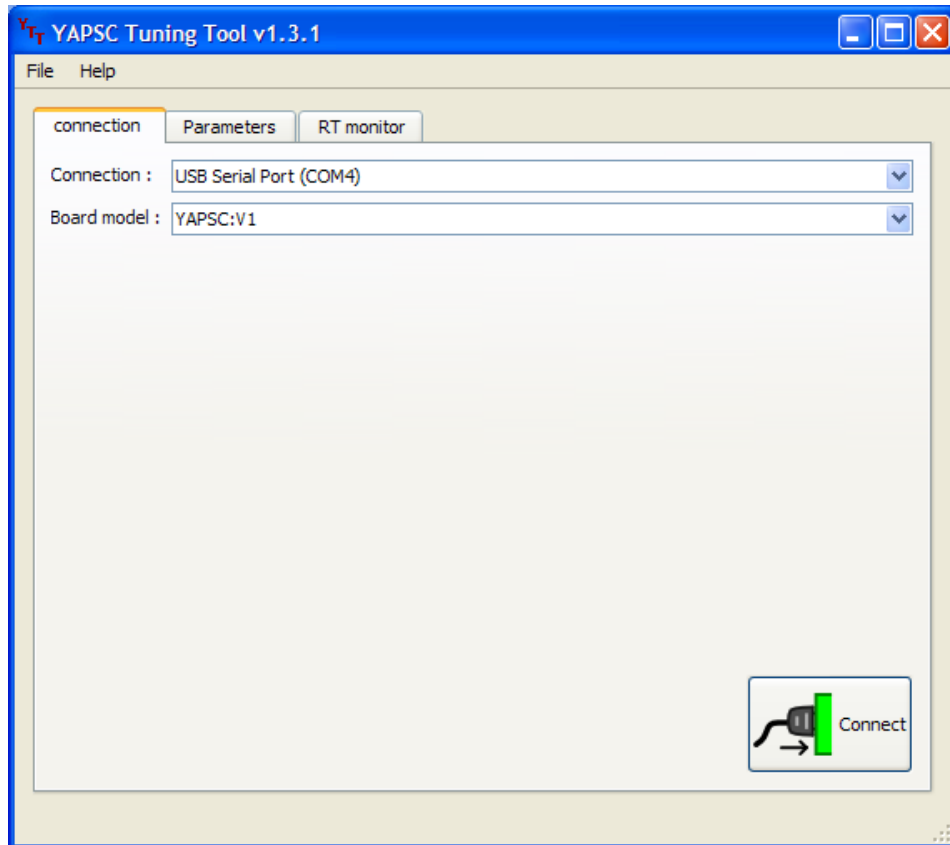


Illustration 1.1: Connection Tab

Once the connection is established, you can use the two other tabs.



Note for the USB ProgKey users:

In the latest model, the ProgKey allows you to remotely reset the board. The board will be held in reset if you have not clicked on the "Connect" button. Once connection established, the blue LED of the ProgKey will turn dark (reset mode disabled). It can be really helpful during the first setup, while it's better to click the "Connect" button before plugging or unplugging the USB ProgKey onto a running YAPSC board: if held in reset mode, YAPSC will act as if completely unpowered.

1.2 "Parameters" tab

This tab displays all the parameters of the board. You will have a short description of each parameter if you hover it with your mouse.

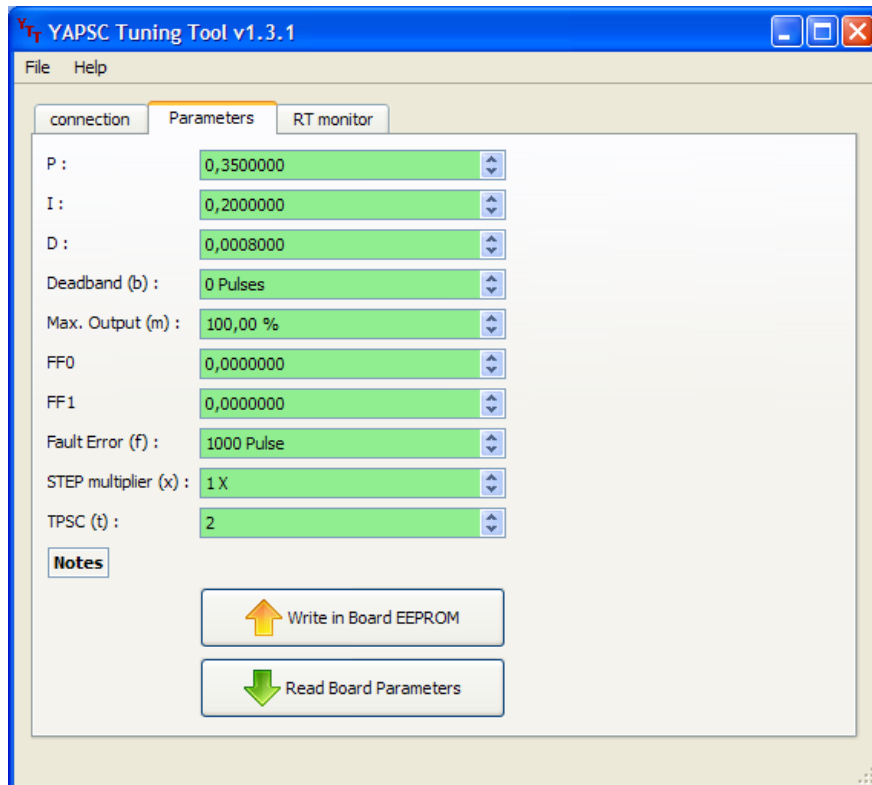


Illustration 1.2: Parameters Tab

All parameters are either

- Green: parameter displayed is saved into the board
- orange: parameter displayed is different from the one saved into the board.

Eg:

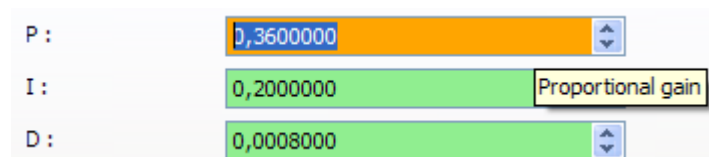


Illustration 1.3: Modified parameters turn orange

P has been modified but is not saved in the board; while the displayed I and D values are the one saved in the board.



You have to save the parameters in the board (by clicking "Write in board EEPROM") for the board to use the new value of the parameter.



You must start by reading the board parameters before writing them

1.3 "RT Monitor" tab

This tab gives you access to

- "Continuous sampling" tool: displays the boards' following error and output on the

graph. Check the “Automatic” box and choose an update period to use.

- “Status” box: actual board's parameters
 - board enabled
 - following, integral and derivate error
 - command & encoder feedback
 - output level [-100; 100]
- “Step impulse response” box: this will be our most useful tool for PID tuning.

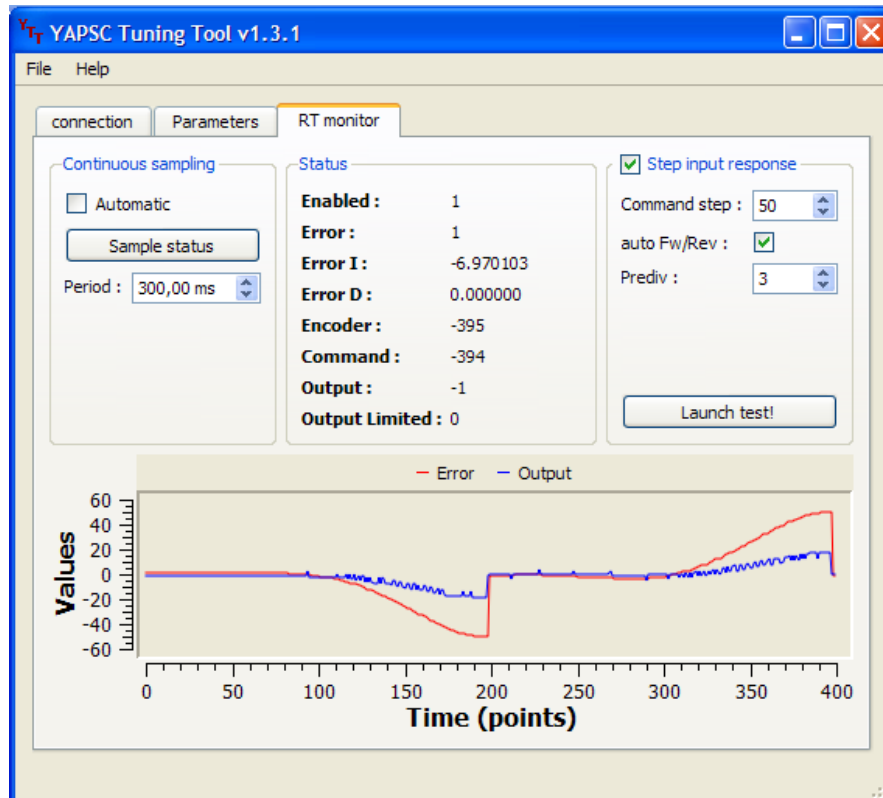


Illustration 1.4: RT Monitor Tab

2 PID tuning: theory

From Wikipedia [\[link\]](#):

If the PID controller parameters (the gains of the proportional, integral and derivative terms) are chosen incorrectly, the controlled process input can be unstable, i.e. its output diverges, with or without [oscillation](#), and is limited only by saturation or mechanical breakage. Tuning a control loop is the adjustment of its control parameters (gain/proportional band, integral gain/reset, derivative gain/rate) to the optimum values for the desired control response.

(I advise that you read the whole article which is very well made if you want to know more about PID and closed loops in general.)

To sum up, we want the system to

- be stable
- be fast
- have no overshoot

These are the key to a well-tuned servomotor.

2.1 Common terms

- **Error** or **Following Error**: the difference of position (in encoder counts) between the actual motor's shaft position and the command
- **Output** : the command (voltage) applied on the motor. It's range is from -100% to 100% of the power supply voltage.
- **Command** : desired position for the motor's shaft (command by the computer to YAPSC board via STEP/DIR)
- **Step input** : when the command suddenly rises of a given value (eg: the command suddenly changes from 0 to +100)

2.2 Stability

First of all, you want your motor to go where you want him to go.

Here are, side by side, two things you want to avoid: divergence and oscillation

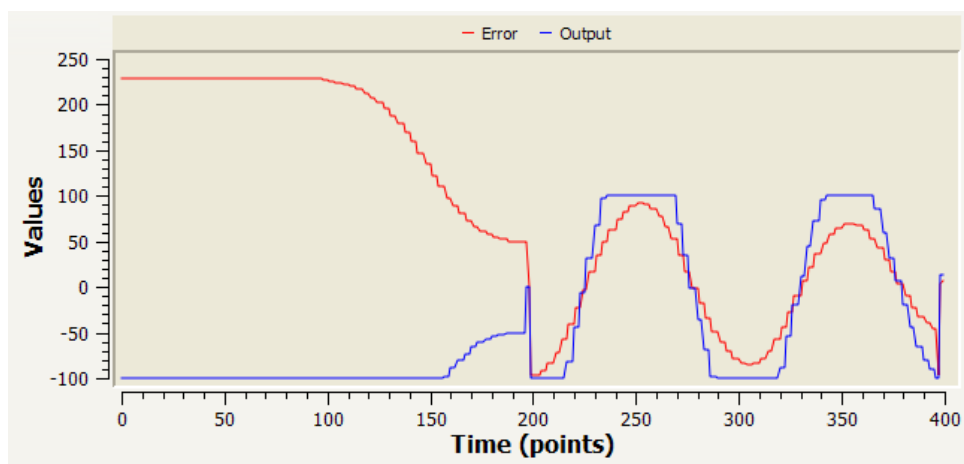


Illustration 2.1: Unstable loops

- from points 1-200, we have a nice example of divergence: the motor polarity is wrong, so instead of bringing the motor back to the desired position, the control loop does the exact opposite! To fix that, you can
 - Turn off the board, swap the motor's ends
 - **or** use negative P, I, D coefficients
- From points 201-400 we can see a nice oscillation of the motor: P and/or I and/or D are set too high. Set I and D to 0 and decrease P until the loop stabilizes:



To avoid the motor from running at full speed until it crashed your machine, don't forget to set up the "Fault Error" parameter! Typical values are in the 400-2000 counts.

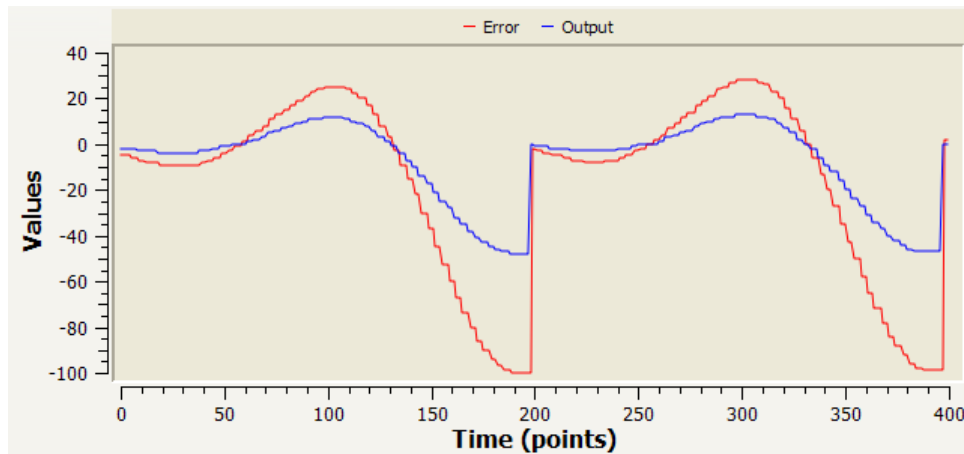


Illustration 2.2: Stable loops

Now the loop is stable: the motor will follow (approximately) the command. It does not oscillate nor it diverges.

2.3 Overshoot

Remember Ill. 7? Well we had a stable loop that follows more or less the command. It does also show something we want to avoid: **overshoot**.

Overshoot is when the motor's shaft goes *over* the command. Here is, on the right overshoot response curve and on the left a response curve with no overshoot:

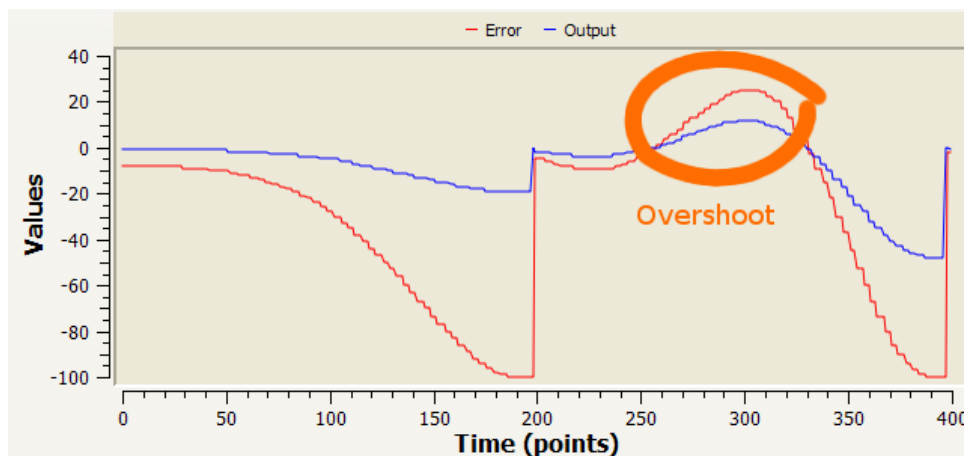


Illustration 2.3: No-overshoot and overshoot

The error is negative at the beginning as actual position was below the command, but error goes over 0 (look at point 300) which means the motor position went over the command!

The second curve (pts 1-200) shows no overshoot : error keeps the same sign (negative).



Q: Why is overshoot a bad thing?

A: Imagine you ask your CNC-milling machine to cut a 10mm long slot in some material. Because of overshoot, the slot may be 10.5mm long instead of 10mm. Ooops! The mill went too far!

3 PID tuning: application

3.1 Hardware setup

1. Let's start from the beginning:

Wire all the power supplies, STEP, DIR and ENABLE inputs.

After you have checked the wiring you can apply power. The ERR led should lit during approx ½ second at power-up.

2. If you have not installed YTT yet, it is time to do it. If you have never installed FTDI USB drivers before, check the last option to install the drivers.
3. Now, plug the Usb cable to the USB ProgKey. It should be recognized immediately.
4. Start YTT. In the "Connection" tab, check that "USB Serial Port (COMx)" is available, and choose it. If you have multiple of them, the progKey should be the latest of the list.
5. Click the "Connect" button, and pay attention to the ProgKey's blue LED: it should be dark. If not, you choose the wrong port number or the driver is not properly installed. You can try to restart your computer, it may help the first time you connect the ProgKey.
6. Now, you can plug the ProgKey on the 2*5pins header of the YAPSC board.
7. Go to "Parameter" tab and click "Read parameters" button. If all the parameters turn from gray to green, you have a working connection between your computer and YAPSC, congratulation!
8. Otherwise, try to disconnect and connect again ("Connection" tab). If it still does not work, verify your cabling with attention.
9. Now you have a working connection, you can move on the next step: testing!

3.2 PID Tuning



Do not forget that the "Step Impulse Response" test makes the motor move! In case of linear axis (example: X,Y and Z on a mill) turn the motor manually to reach the middle of the axis under testing before starting the test and setup the "Fault Error". Be sure to choose a command step small enough to avoid the carriage to crash on the ends of the slides.

First step for PID tuning is to set (roughly) the P parameter so we'll set **I** and **D** parameters to 0. The general method for loop tuning is to change one parameter at a time, run the test, analyze the response and start again until you cannot reduce the settling time anymore without losing stability.

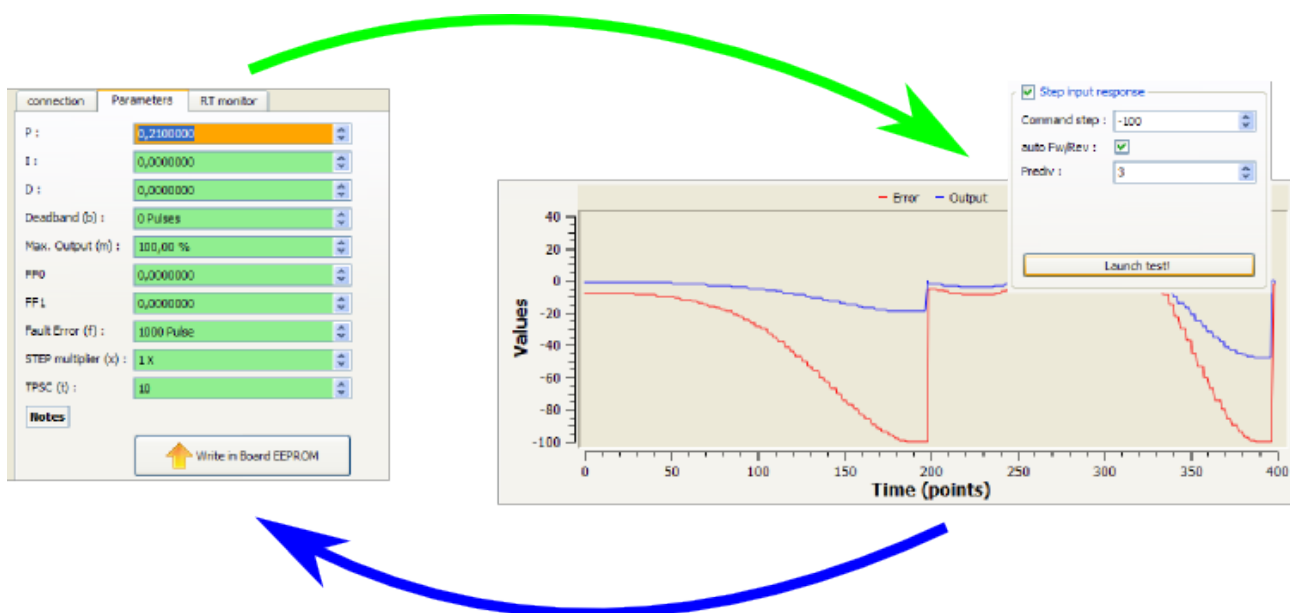


Illustration 3.1 : tuning method

First, choose a command step amplitude. Something in the 50 – 100 range can give acceptable resolution. You can go higher, but an important thing is to keep $(\text{amplitude} * P) < (\text{Max Output}/2)$. Otherwise the output is likely to saturate which will completely change the test results.



To calculate the equivalent move to the specified step, refer to section 5 Extra information.

3.2.1 P tuning

Augment progressively the P parameter until you get no or little overshoot. A nice curve would be between the "no overshoot" and "overshoot" example of Illustration 2.3: No-overshoot and overshoot.

At this point, it is not a problem if there is little overshoot. The **D** parameter reduces efficiently the overshoot.

3.2.2 I tuning

Most of time, **P** gain tuning alone cannot make the motor reach exactly the goal; there is a static error. For example, you ask for a 50 steps move and the motor stops after 46 steps: the static error is 4.

Augment **I** parameter until the static error reaches 0, but while keeping loop stability

3.2.3 D tuning

The D parameter should be used to erase eventual overshoot but don't set it too high as it will reduce the response time.

3.2.4 Sum up

To Improve...	Action on P	Action on I	Action on D
...Response time	increase	increase	decrease
...Stability	decrease	decrease	increase
...Precision	Increase (decrease if not stable)	increase	-

4 Troubleshooting

Here are the common problems you may face.

4.1 Inverted polarity

Problem description: as soon as the motor moves a bit it runs into the same direction and the boards goes in "Maxerror Fault".

Fix: The fix is simple. You can swap A and B encoder inputs **OR** use negative PID parameters (example: P=-0.23; I=-0.07; D=-0.016)

5 Extra information

5.1 From steps to rev

To calculate the equivalent in revolutions to *steps* step move use this formula:

$$rev = \frac{steps}{PPR \times 4}$$

PPR is the Pulse Per Revolution of the encoder. Can be called the number of lines of the encoder. (ex: "500 lines encoder" has $PPR=500$)

5.2 From steps to mm (in)

For a *l* lead screw driven by *m:n* reduction, steps step move is equivalent to:

$$\Delta = \frac{m \times l}{n} \times rev$$

Example for a 5mm/turn (5mm lead) ballscrew driven by our motor through a 1:3 reduction: a 50 steps move is

$$\frac{1 \times 5}{3} \times \frac{steps}{PPR \times 4} = \frac{5}{3} \times \frac{50}{500 \times 4} = 0,042 \text{ mm}$$

The encoder *PPR* is 500.

5.3 Units per Step

It represents of how much the machine moves for each step.

$$UpS = \frac{m \times l}{n \times PPR \times 4}$$



This is commonly used by your fab soft (MAC3/EMC etc).

Document Revisions

Rev	Date	Description
1	11/22/09	First version of this manual