PID TUNING MANUAL

Francisco Infantino

Class of 2027 @ Ultro Robotics

Last updated 2025-06-11.

Definitions:

A PID controller is a type of feedback control system widely used in industrial and engineering applications to regulate processes like temperature, pressure, speed, or position. The term PID stands for Proportional, Integral, and Derivative, which are the three components that work together to calculate the control output. Here's how each component functions:

1. Proportional (P):

- **Purpose:** Reacts to the current error (the difference between the desired setpoint and the actual process variable).
- **Action:** The controller output is directly proportional to the magnitude of the error.
- **Effect:** Large errors result in stronger corrective action. However, it cannot eliminate steady-state error entirely.
- **Tuning Parameter: Proportional Gain (K□)** - Adjusts the responsiveness of the proportional term.

2. Integral (I):

- **Purpose:** Eliminates steady-state error by considering the accumulation of past errors.
- **Action:** Integrates the error over time and adds it to the control output.
- **Effect:** Helps the system achieve zero steady-state error, but too much integral action can make the system unstable.
- **Tuning Parameter: Integral Gain (K_i)** - Adjusts the influence of the integral term.

3. Derivative (D):

- **Purpose:** Predicts future error trends by looking at the rate of change of the error.
- **Action:** Adds a component to the output proportional to the error's rate of change.
- **Effect:** Dampens oscillations and improves stability by counteracting rapid error changes, but excessive derivative action can amplify noise.
- **Tuning Parameter: Derivative Gain (K_d)** - Adjusts the impact of the derivative term.

4. Gravity Constant (K_g):

- **Purpose:** Accounts for gravitational effect on slides.

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

- **Action:** Adds power to motors at a certain power so that slides which are under gravitational force do not go back down once they reach their desired position
- **Effect:** Keeps slides at desired position, not necessary on horizontal slides as no gravity affects them.
- **Tuning Parameter: Additive Gain (K_g)** - Adjusts the impact of the additive term.

Setup For Tuning Linear Slides:

For the purposes of using **linear slides**, only the **P & K_g values** need to be used. To tune **linear slides**, it is useful to create a separate testing **OpMode** that allows you to quickly adjust values through trial and error using the **FTC Dashboard**. You will need to connect to the robot's Wi-Fi to use the **FTC Dashboard**. Below are examples of both sample testing **OpModes** for **linear slide PID values** and instructions on how to use the **FTC Dashboard** for tuning:

1. Testing OpMode:

```
package org.firstinspires.ftc.teamcode.common;

import com.acmerobotics.dashboard.config.Config;
import com.arcrobotics.ftclib.controller.PIDController;
import com.qualcomm.hardware.lynx.LynxModule;
import com.qualcomm.robotcore.eventloop.opmode.OpMode;
import com.qualcomm.robotcore.eventloop.opmode.TeleOp;
import com.qualcomm.robotcore.hardware.DcMotorEx;
import com.qualcomm.robotcore.hardware.DcMotorSimple;

import org.firstinspires.ftc.teamcode.Robot;

import java.util.List;

@Config
@TeleOp
```

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

```java
public class DoubleLinearSlidePIDTuner extends OpMode {

   public DcMotorEx liftMotorOne;
   public DcMotorEx liftMotorTwo;
   public DcMotorEx liftEncoder;
   public PIDController liftController;

   private Robot robot;

   public static double slideP = 0;
   public static double slideI = 0;
   public static double slideD = 0;
   public static double slideKg = 0;

   public static double SLIDE_TICKS_PER_INCH = 2 * Math.PI * 0.764445002 /
537.7;

   public static double targetPosition = 0;

   public List<LynxModule> controllers;

   @Override
   public void init(){
       liftController = new PIDController(slideP, slideI, slideD);

       //this.robot.reset();

       liftMotorOne = hardwareMap.get(DcMotorEx.class, "liftOne");
       liftMotorTwo = hardwareMap.get(DcMotorEx.class, "liftTwo");
       //liftEncoder = hardwareMap.get(DcMotorEx.class, "LB");

       liftMotorTwo.setDirection(DcMotorSimple.Direction.FORWARD);
       liftMotorOne.setDirection(DcMotorSimple.Direction.REVERSE);
       //liftEncoder.setDirection(DcMotorSimple.Direction.REVERSE);
```

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

```
        controllers = hardwareMap.getAll(LynxModule.class);

        for(LynxModule module : controllers){
            module.setBulkCachingMode(LynxModule.BulkCachingMode.MANUAL);
        }

    }

    @Override
    public void loop(){
        liftController.setPID(slideP, slideI, slideD);

        double liftEncoderPosition = liftMotorTwo.getCurrentPosition();

        double liftPosition = liftMotorOne.getCurrentPosition() *
SLIDE_TICKS_PER_INCH;
        double liftPosition2 = liftMotorTwo.getCurrentPosition() *
SLIDE_TICKS_PER_INCH;

        double pid = liftController.calculate(liftPosition2, targetPosition);

        double liftPower = pid + slideKg;

        liftMotorOne.setPower(liftPower);
        liftMotorTwo.setPower(liftPower);

        telemetry.addData("Lift Position, Motor 1", liftPosition);
        telemetry.addData("Lift Position, Motor 2", liftPosition2);
        telemetry.addLine();
        telemetry.addData("Lift Target", targetPosition);
        telemetry.addData("Lift Power", liftPower);
        telemetry.addData("lift Encoder", liftEncoderPosition);
        telemetry.update();
```

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

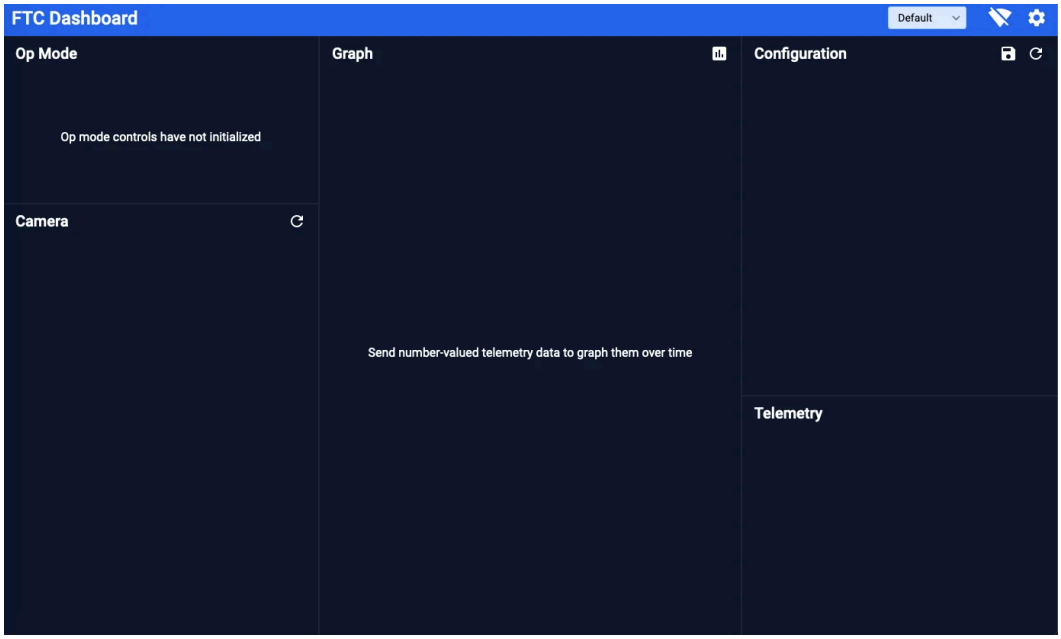
```
        for(LynxModule module : controllers){
            module.clearBulkCache();
        }
    }
}
```

This is an example of testing code for a **Double Linear Slide**, which means that it will power two motors at once. Only one motor's **encoder values** are used because utilizing encoders from both motors can cause conflicts in their values and therefore confusion in the **PID equation**. Reverse **motor direction** as well as **encoder direction** according to how they will be utilized. Be careful not to start tuning until this is done, as it can **break stringing** if not done correctly. Finally, ensure that the correct motor's **encoder values** are being used in the **PID double**, as incorrect values can cause problems during tuning. To make an OpMode for just one slide, simply remove or comment out the other motor from the code (which does not have its **encoder values** used).

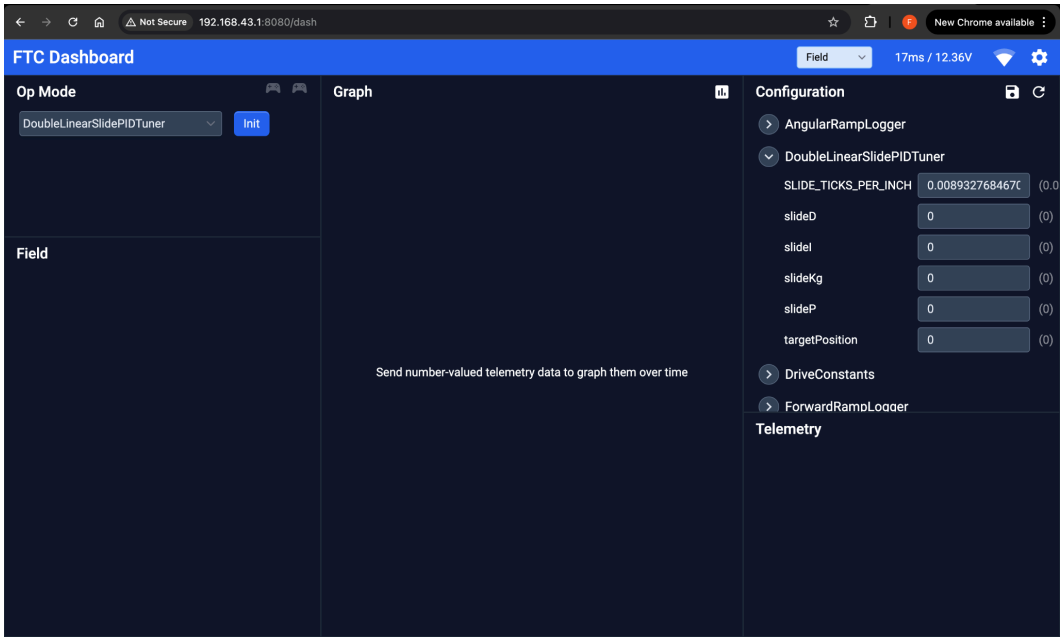
2. FTC Dashboard:

To navigate to the **FTC Dashboard**, first connect to your robot's Wi-Fi. Then, in the search bar on **Google** on your computer, navigate to **192.168.49.1:8080/dash** with a **phone RC** or **192.168.43.1:8080/dash** with a **Control Hub**. Once you have done this, you should see this window (screenshot on next page):

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

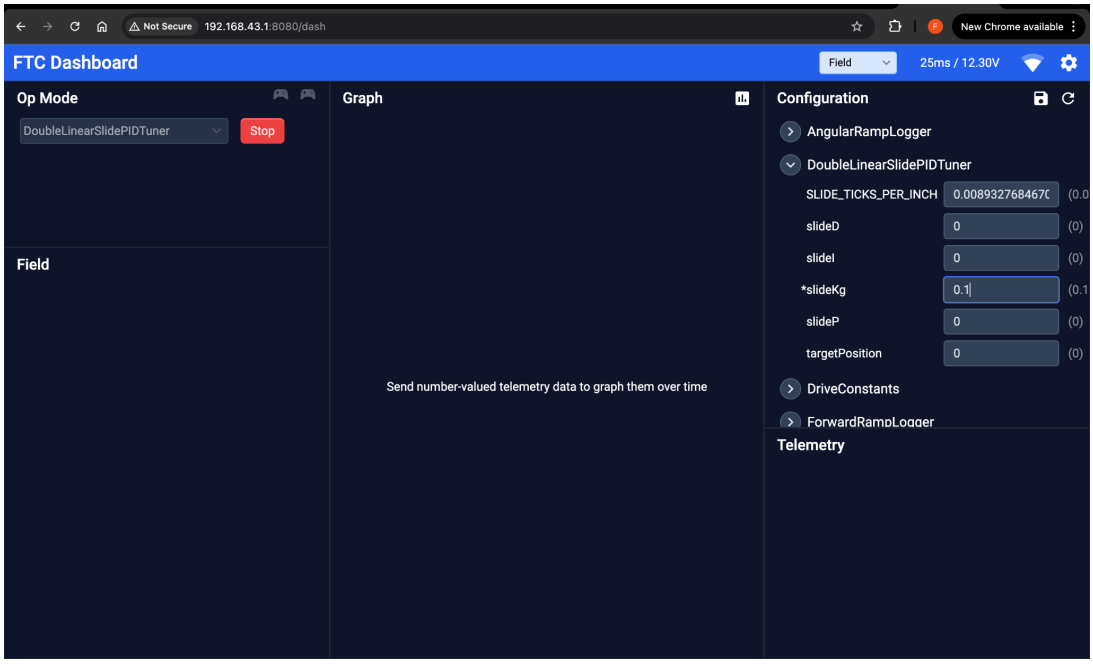


For OpModes to show up, ensure your **Control Hub** is **Powered** and you connect to the **Wi-Fi:**



Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

Select the **PID tuning OpMode** you created on the **right** side of the **Dashboard**. On the **left**, you will notice the **OpMode** you created again; open it to see all the values you can change in real time using the **FTC Dashboard**.



You will notice that when you **change values** in the FTC Dashboard, an ***** will appear next to them. This asterisk next to the value means that it is not applied to the code, in order to apply it, press **enter/return** on the value and it will apply it, make sure to be careful when doing this as it may break stringing. Now you know everything you need to know to begin tuning!

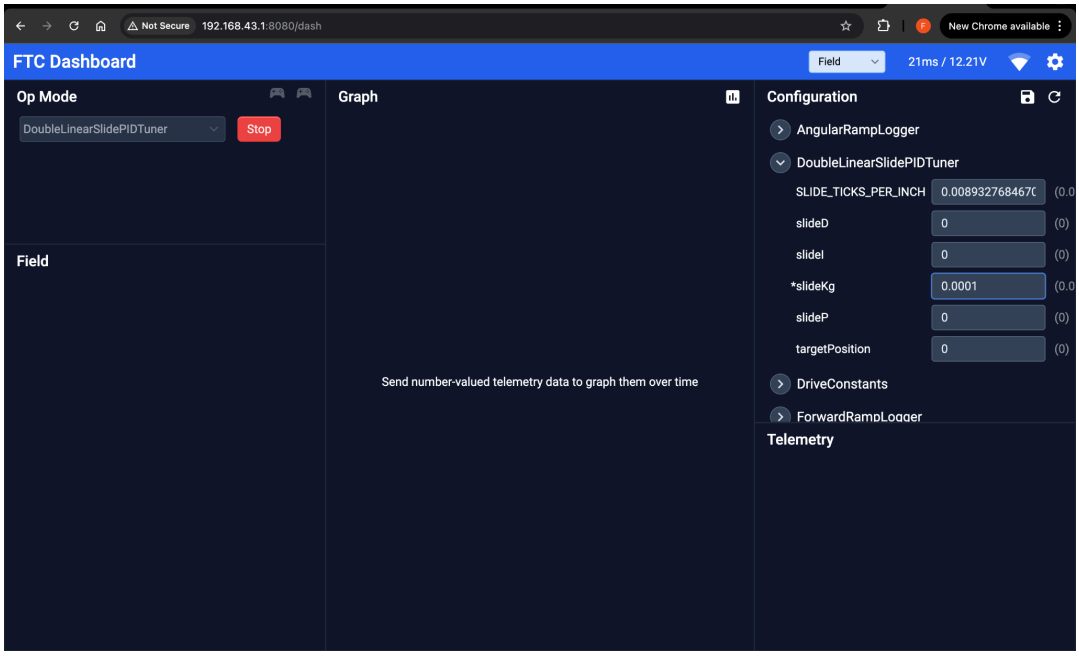
Tuning Linear Slides:

1. Tips/Background:

The first variable that should be tuned is $\mathbf{K_g}$ (of course only when the slides are vertical, or any other angle where gravity acts on the slides. Not horizontal as no gravity acts on horizontal slides), only after this, the **P** value can be tuned. As a quick check that all your encoder directions are correct, **Extend** your linear slides: if the registered ticks go **up** when you **extend** slides and go **down** when you **retract** slides the encoder directions for the motors are correct.

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

When putting in different values for the variables, make sure to pick an **extremely low value** at the beginning (1 is the maximum power for a motor, so it has to be low). For example, start at a value like: **0.0001**.



Please ensure that the **Ticks Per Inch** value of your OpMode is **correct** (with correct ticks per rotation, radius of spool etc...) as this will make tuning your **P** value **easier** later on.

Ensure that your **slides are fully retracted** when you **turn on the robot**. **Ticks are always registering** on the motors, so if the slides are up, the **targetPosition** for when your slides are fully down will be in the **negatives**. By ensuring that the **slides are down** when you start the robot, the **targetPosition** of **0** should make your slides fully retracted.

If your robot is already on, simply **turn it off and then back on** once your slides are fully retracted.

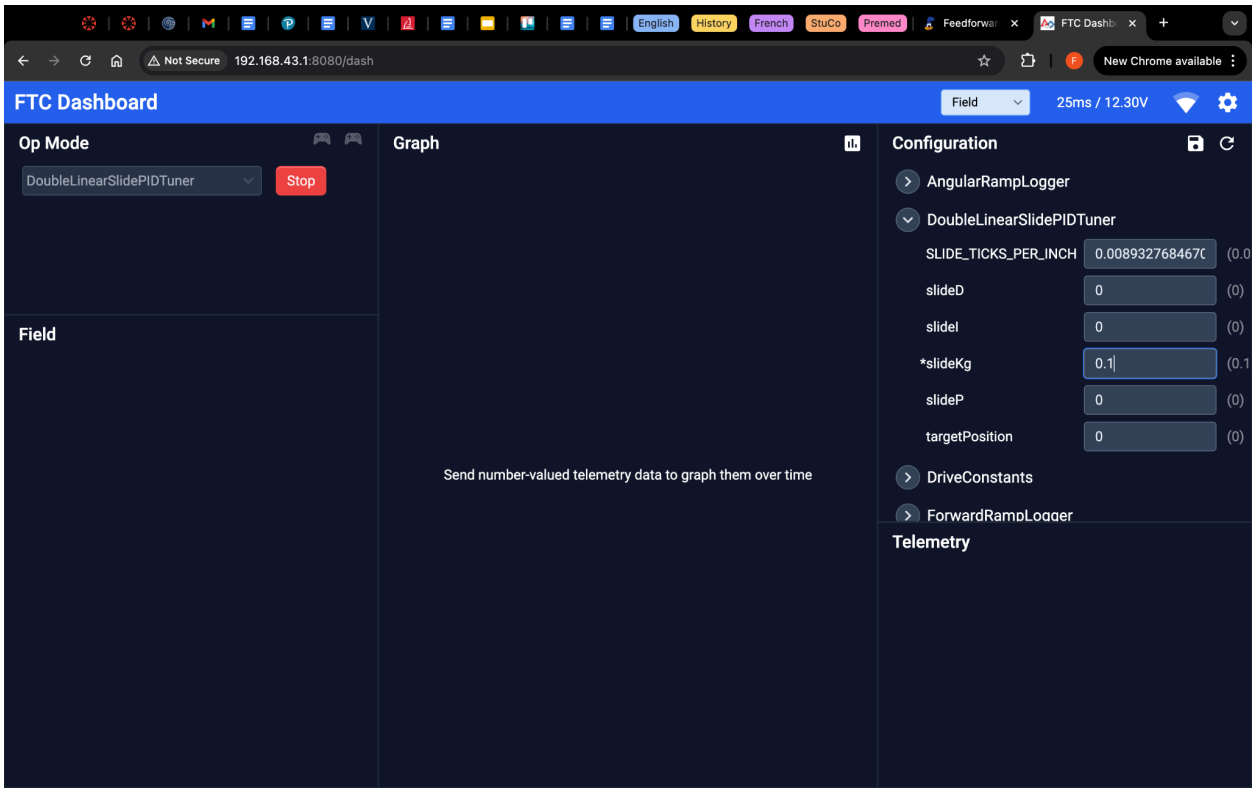
2. Tuning K_g:

In order to tune K_g, first **initialize**, and then **start** your OpMode. First we will make sure **motor directions** are correct: extend your slides **manually**(by lifting them yourself) and then **add** to the value of **K_g**, if you feel the slides pulling **down**(slides will need some form of retraction for this to happen), your motors are **reversed**; if it becomes **easier** to keep the slides **up**, your motor directions are **correct**. Please remember to keep **K_g values low**, especially now since if you

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

make it too high, not only can strings **break**, but you can also **hurt yourself**. Please check your encoder directions are correct **again** if you change your motor directions as your encoder directions may **switch** if the directions of your motors change.

Now to get the **correct** value of K_g once again **lift** the linear slides. **Add** to the value of K_g until the slides can **hold themselves up** without external forces acting on it. Try lifting the slides to **different** heights manually; if it **stays** there, the K_g value is doing what it is **intended** to do. The slides should **stay up** when slides are fully **extended** but also should not be **too difficult** to push all the way back **down**. Keep **adjusting** the values of $\mathbf{K_g}$ until the conditions from before are met. This **doesn't** have to be perfect, just **close enough**.

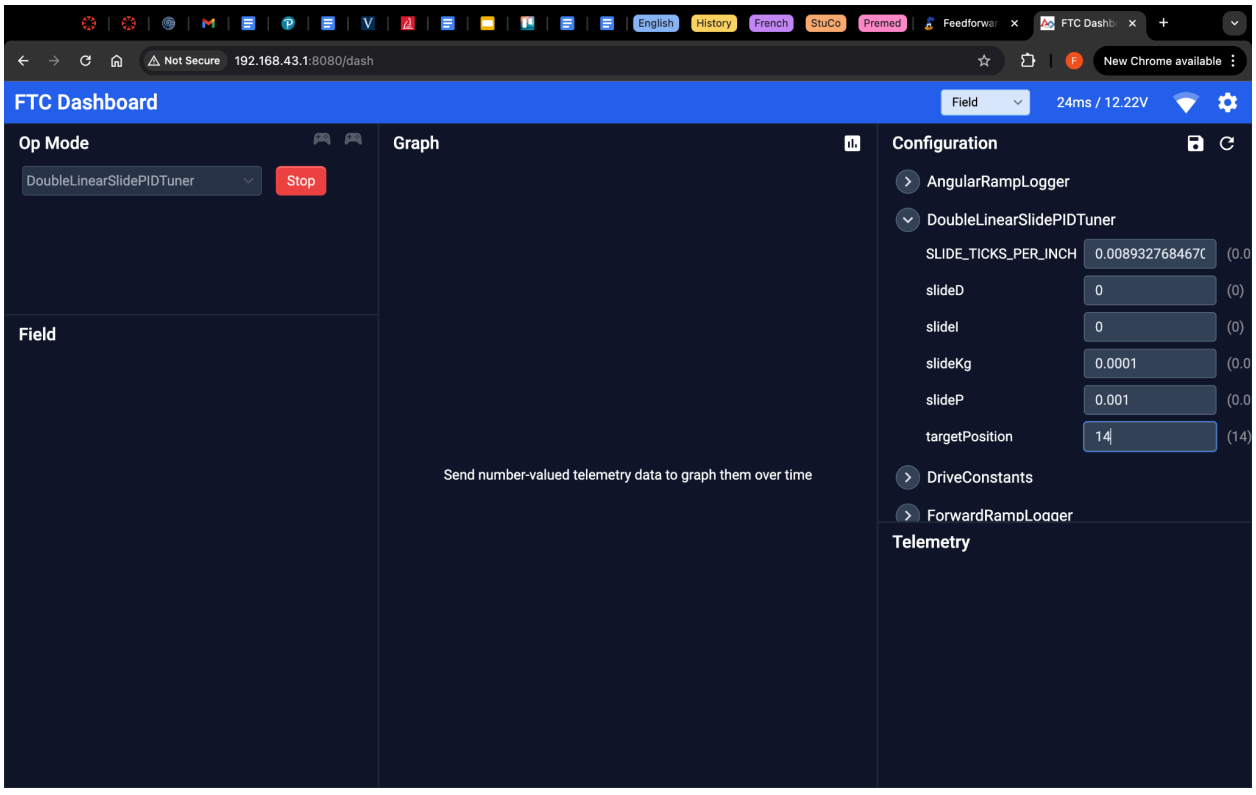


Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

3. Tuning P:

Tuning **P** is a little more dangerous than tuning $\mathbf{K_g}$, as the slides will be moving on their own now and may do so quite quickly depending on the motors you are using. Because of this, please **keep your hands away from the slides** during this portion of the tuning.

Begin by **keeping the P value at 0** and changing the **targetPosition** to any value—for example, **10** (representing **10 inches** if the ticks-to-inches value is correct).



Now, **change the value of P** (remember to start with a **very low number** as mentioned previously) until the slides reach the **desired position** (10 inches) in real life. You'll notice the slides **moving slightly more and more** as you increase the P value. Once you've reached the desired position, **change the targetPosition value again** and continue refining the P value so that it is **more accurate and fast** (a **higher P value** usually means **more speed**). Be sure to be **careful** when doing so, as you can **damage the slides** if you are not cautious.

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.

Once you are satisfied with the **speed**, **efficiency**, and **accuracy** of the P value, you are done. As a final test, I recommend **finding the maximum and minimum values** of extension or retraction for the slides. This not only tests how the P value affects the slides at **different positions** but also helps you determine the **positional values** that will be used in-game.

Copyright © Ultro Robotics
Licensed under CC-BY-SA 4.0.