

PIDControl Arduinorako

PID kontrola, prozesuen identifikazioa, sintonizazioa eta iragazketa

PIDControl v1.4.2 dokumentazioa

Esparrua. Dokumentu honek PIDControl v1.4.2 liburutegiaren APIa deskribatzen du eta, kapitulu bereizietan, hardwarean baliozkotzeko erabilitako TCLab sketch-a. Kotsola-komandoak proba-sketch-ari dagozkio; liburutegi nagusia ez dago Serial-en menpe.

19/09/2026

1. Sarrera: zer eskaintzen du PIDControl-ek	4
2. Liburutegiaren arkitektura	5
1.4.2 bertsioaren berritasunak	5
3. Terminologia, magnitudeak eta unitateak	6
4. Kontrolagailuaren eraikuntza eta oinarrizko zikloa	7
5. PID, PI-D eta I-PD egiturak	8
6. DIRECT eta REVERSE ekintzak	9
7. MAN/AUTO moduak, PV Tracking eta jauzirik gabeko transferentzia	10
MAN modua	10
AUTO modua	10
PV Tracking	10
Jauzirik gabeko transferentzia	10
8. Saturazioa, anti-windup, mugak eta laginketa-denbora	11
Irteera-mugak	11
Anti-windup	11
Laginketa-denbora	11
9. Seinale-egokitzaipena eta iragazkiak	12
Batez besteko mugikorra	12
Mediana	12
Lehen ordenako behe-paseko iragazkia	12
Iragazki osagarria	12
10. Begizta irekiko identifikazioa StepTest bidez	13
Lan-sekuentzia	13
Egoera egonkorren irizpide sendoa	13
Noranzkoko erantzuna	13
11. FOPDT eredua eta irabazi normalizatua	14
Irabazia ingeniari-tza-unitateetan	14
T0 eta Tp-ren estimazioa	14
Irabazi normalizatua	14
FOPDTModel objektua	14
12. IMC/Lambda eta Tyreus-Luyben sintonizazioa PIDTuning bidez	15
IMC / Lambda PI	15
Lambda PID	15
13. Errele-saiakuntza RelayTest bidez	16
APIko konfigurazio trinkoa	16
Funtzionamendua	16
Ku eta Tu-ren kalkulua	16

Emaitzaren iraunkortasuna memorian.....	17
TCLab-en benetako baliozkotze-adibidea.....	17
Tyreus-Luyben PIDControl 1.4.2-n integratuta	17
14. Saiakuntzen egoerak, erroreak eta portaera segurua	17
StepTestState	17
StepTestError	18
RelayTestState	18
RelayTestError.....	18
15. TCLab sketch-a: eskuratzea, iragazketa, segurtasuna eta baliozkotzea.....	20
Eskuratze-katea.....	20
Funtzionamendu-tartea eta babes termikoa	20
16. TCLab sketch-aren serieko komando-interfazea.....	21
Komando orokorrak	21
StepTest komandoak.....	21
RelayTest komandoak	22
RelayTest formatu trinkoa	22
Emaitzak IDENT bidez berreskuratzea	23
Arduino COM Plotter-erako trama.....	23
17. Gomendatutako lan-fluxuak	24
Funtzionamendu normala.....	24
FOPDT identifikazioa + IMC/Lambda.....	24
Errele-saiakuntza.....	24
18. APIaren erreferentzia osoa	25
PIDControl	25
StepTest.....	25
RelayTest.....	26
PIDTuning.....	26
PIDControlFilters	27
19. Mugak eta jardunbide egokiak	28
20. Paketearen egitura eta biltegirako prestaketa	29
Instalazioa.....	29
A eranskina. Gutxieneko adibide osoa	30
B eranskina. Laburpen azkarra	31

1. Sarrera: zer eskaintzen du PIDControl-ek

PIDControl prozesuen kontrolera bideratutako Arduino liburutegia da. Helburua ez da PID irteera bat kalkulatzera mugatzea, baizik eta kate oso bat estaltzea: seinalea egokitzea, MAN/AUTO kontrola, prozesua identifikatzea, sintonizatzea eta baliozkotzea.

Liburutegiak industria-instrumentaziotik hurbil dagoen filosofia jarraitzen du: PV, SP eta OP bereizten ditu; ekintza zuzena eta alderantzizkoa bereizten ditu; PV Tracking, jauzirik gabeko transferentzia, anti-windup eta kontrolagailuaren hainbat egitura ezartzen ditu. Identifikazio-saiakuntzak ez dira blokeatzaileak eta jada eskalatutako aldagaiekin lan egiten dute; beraz, hardwarea algoritmotik kanpo geratzen da.

- PID kontrolagailua PID, PI-D eta I-PD egiturekin.
- MAN eta AUTO moduak.
- DIRECT eta REVERSE ekintzak.
- PV Tracking eta jauzirik gabeko transferentzia.
- Integrazio baldintzatuaren bidezko anti-windup-a.
- Irteera-muga eta laginketa-denbora konfiguragarriak.
- Batez besteko mugikorreko, mediana, behe-paseko eta iragazki osagarriak.
- StepTest, begizta irekiko FOPDT identifikaziorako.
- Prozesu-irabazia ingeniartza-unitateetan eta irabazi normalizatua.
- PIDTuning, IMC/Lambda PI, Lambda PID eta Tyreus-Luyben sintonizaziorako.
- RelayTest, errele-saiakuntza bidez Ku eta Tu lortzeko.

Idea nagusia. PIDControl-en helburua da aplikazio-kodea PV irakurtzera, kontrol/identifikazio klaseei deitzera eta OP idaztera mugatzea. Kontrol-logika ez da pinekin, ADCrekin, PWMrekin edo komunikazioekin nahastu behar.

2. Liburutegiaren arkitektura

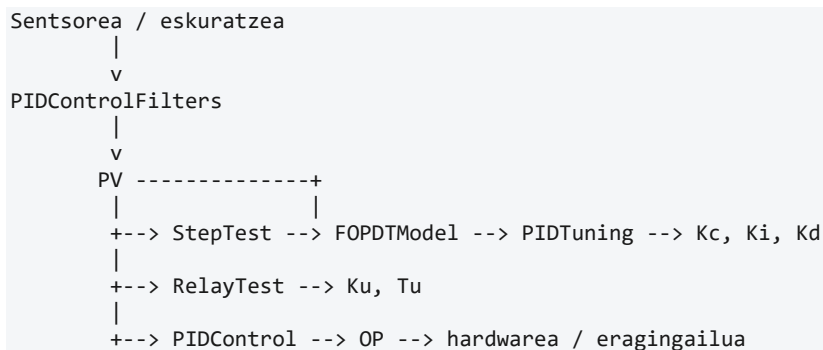
PIDControl v1.4.2 bost bloke nagusitan banatzen da. Banaketa horri esker, proiektu bakoitzean behar diren moduluak soilik erabil daitezke.

1.4.2 bertsioaren berritasunak

1.4.2 bertsioak identifikazio- eta sintonizazio-katea osatzen du. StepTest-ek IMC/Lambda PI sintonizazio bat eta Lambda PID sintonizazio bat eskaintzen ditu; RelayTest-ek, berriz, Ku/Tu balioetatik Tyreus-Luyben PI eta PID proposamenak sor ditzake. TCLab sketch-ak RELAY(...) formatu trinkoa eta memorian gordetako emaitzak berreskuratzeko IDENT komandoa gehitzen ditu.

Modulua	Eginkizuna	Goiburua
PIDControl	Begizta itxiko kontrola: PID, PI-D, I-PD, MAN/AUTO, ekintza, tracking, bumpless transfer eta anti-windup.	PIDControl.h
StepTest	Urrats-erantzuneko saiakuntza ez-blokeatzailea eta FOPDT identifikazioa.	StepTest.h
RelayTest	Ku eta Tu estimatzeko errele-saiakuntza ez-blokeatzailea.	RelayTest.h
PIDTuning	IMC/Lambda PI eta Lambda PID sintonizazioa FOPDTtik; Tyreus-Luyben PI/PID Ku/Tutik.	PIDTuning.h
PIDControlFilters	Seinale-iragazki berrerabilgarriak.	PIDControlFilters.h

Moduluen arteko ohiko erlazioa honela irudika daiteke:



StepTest-ek eta RelayTest-ek PV eta OP aldagaietarako erakusleak jasotzen dituzte. Ez dakite zein pin erabiltzen den edo irakurketa nola eskalatzen den. Horri esker, identifikazio bera erabil daiteke ADC/PWM, I2C, SPI, Modbus, simulagailu edo beste edozein hardware-geruzarekin.

3. Terminologia, magnitudeak eta unitateak

Ikurra	Esanahia	Adibidea
PV	Process Variable. Neurtutako prozesu-aldagaia.	T1 tenperatura [°C]
SP	Setpoint. Nahi den kontsigna.	50 °C
OP	Output. Kontrolagailuaren irteera.	0..100 %
Kc	Kontrolagailuaren irabazi proportzionala.	6.40
Ki	Barnean erabilitako irabazi integrala [1/s].	0.0426 1/s
Kd	Barnean erabilitako irabazi deribatiboa [s].	0
Ti	Denbora integrala.	150 s
Td	Denbora deribatiboa.	0 s
Kp	Prozesu-irabazia: $\Delta PV/\Delta OP$.	°C/%OP
T0	Prozesuaren denbora hila.	s
Tp	Prozesuaren denbora-konstantea.	s
Ku	Errele-saiakuntzaren bidez estimatutako azken irabazia.	dimentsiorik gabea, eskalatzearen arabera
Tu	Azken periodoa.	s

Sintonizazio-parametroen bihurketa

$$K_i = K_c / T_i$$

$$K_d = K_c * T_d$$

PIDControl-ek segundoak erabiltzen ditu kontrolagailuaren ekuazioetan. Beraz, T_i , T_d eta K_i/K_d -rekin lotutako denbora-oinarriak koherenteak izan behar dute segundoekin.

Kc ez da Kp. Dokumentazio honetan Kc kontrolagailuaren irabazirako gordetzen da, eta Kp identifikatutako prozesuaren irabazirako. Bereizketa horrek PID dokumentazioan ohikoa den anbiguotasuna saihesten du.

4. Kontrolagailuaren eraikuntza eta oinarrizko zikloa

Eraikitzaileak PV, OP eta SP aldagaietarako erakusleak jasotzen ditu, Kc, Ki eta Kd parametroekin batera. Adierazpen laburrenak liburutegiaren balio lehenetsiak erabiltzen ditu eta ohiko konfigurazio baterako nahikoa izaten da.

```
PIDControl pid(&PV, &OP, &SP, Kc, Ki, Kd);

// Adierazpen osoa
const bool pvTracking = true;

PIDControl pidOsoa(
    &PV, &OP, &SP,
    Kc, Ki, Kd,
    PIDType::PI_D,
    PIDAction::REVERSE,
    pvTracking
);
```

Adierazpen laburrekin kontrolagailua MAN moduan sortzen da, PI_D egiturarekin, REVERSE ekintzarekin, PV Tracking aktibatuta, 0 eta 100 % arteko irteera-mugekin eta 100 ms-ko laginketa-denborarekin. Adierazpen osoak parametro horiek esplizituki uzteko edo beste balio batzuekin ordezkatzeko aukera ematen du.

```
void loop()
{
    PV = ProzesuAldagaiaIrakurri();

    pid.Compute();

    IrteeraIdatzi(OP);
}
```

Compute() funtzioak true itzultzen du kalkulu automatiko berri bat egin denean. AUTO moduan konfiguratutako laginketa-denbora errespetatzen du. MAN moduan eskuzko irteera aplikatzen du eta, PV Tracking aktibo badago, barneko SP eguneratzen du PV-ari jarrai diezaion.

Garrantzitsua. Compute() MAN moduan ere deitzea komeni da. PIDControl-en modu manualak ez du esan nahi klaseak parte hartzeari uzten dionik: eskuzko OP mantentzen du eta, dagokionean, PV Tracking ere bai.

5. PID, PI-D eta I-PD egiturak

PIDControl-ek hiru kontrolagailu-egitura erabil ditzake PIDType bidez. Lehenetsitako egitura PI_D da.

Egitura	P-k honen gainean jarduten du	I-k honen gainean jarduten du	D-k honen gainean jarduten du	Erabilera kontzeptuala
PID	Errorea	Errorea	Errorea	PID klasikoa.
PI_D	Errorea	Errorea	PV	SP aldaketetan derivative kick saihesten du.
I_PD	PV	Errorea	PV	SP aldaketen berehalako eragina murrizten du P eta D osagaietan.

REVERSE ekintzarako, $e = SP - PV$ izanik, inplementatutako ekuazioak hauek dira:

```
PID:
P = Kc * e
I(k) = I(k-1) + Ki * e * Ts
D = Kd * (e - aurrekoErrorea) / Ts
```

```
PI-D:
P = Kc * e
I(k) = I(k-1) + Ki * e * Ts
D = -Kd * (PV - aurrekoPV) / Ts
```

```
I-PD:
P = -Kc * PV
I(k) = I(k-1) + Ki * e * Ts
D = -Kd * (PV - aurrekoPV) / Ts
```

Egitura edozein unetan hauta daiteke honela:

```
pid.SetStructure(PIDType::PID);
pid.SetStructure(PIDType::PI_D);
pid.SetStructure(PIDType::I_PD);
```

Kontrolagailua AUTO moduan dagoela egitura aldatzen bada, PIDControl-ek termino integrala berriz kalkulatzeko du InitializeBumpless() bidez, OP-ren jauzia minimizatzeko.

6. DIRECT eta REVERSE ekintzak

Kontrolagailua modu automatikoan jarri aurretik kontrol-ekintza zuzena dela egiaztatu behar dugu. Hautatzeko modurik errazena PV handitzen denean OP-k zer egin behar duen galdetzea da. PV handitzean OP txikitu behar bada, REVERSE erabiliko dugu. PV handitzean OP ere handitu behar bada, DIRECT erabiliko dugu. PIDControl-en lehenetsitako ekintza REVERSE da.

Ekintza	Ohiko portaera	Adibidea
REVERSE	PV handitzen bada, OP txikitzen da.	Berokuntza: tenperatura handitzen bada, potentzia murrizten da.
DIRECT	PV handitzen bada, OP handitzen da.	PV eta OP noranzko berean aldatu behar diren prozesuak.

```
pid.SetDirection(PIDAction::REVERSE);
pid.SetDirection(PIDAction::DIRECT);
```

TCLab. Adibideetan erabilitako prozesu termikoan, tenperatura handitzen bada berogailuaren potentzia murriztu nahi dugu. Horregatik REVERSE ekintza erabiltzen da.

7. MAN/AUTO moduak, PV Tracking eta jauzirik gabeko transferentzia

MAN modua

MAN moduan, irteera SetManualOutput() bidez ezartzen da.

```
pid.SetMode(PIDMode::MAN);
pid.SetManualOutput(40.0);
```

AUTO modua

AUTO moduan, kontrolagailuak kalkulatzen du irteera.

```
pid.SetMode(PIDMode::AUTO);
```

PV Tracking

PV Tracking lehenespenez aktibatuta dago. Kontrolagailua MAN moduan dagoen bitartean, barneko SP-ak PV-ari jarraitzen dio eta operadoreak eskatutako kontsigna operatorSetpoint aldagaian bereiz gordetzen da. Horrek MANetik AUTOrat igarotzea errazten du eta irteeran bat-bateko jauzirik gabeko transferentzia prestatzen du.

```
pid.SetPVTracking(true);
pid.SetSetpoint(50.0);    // operadorearen kontsigna gisa gordetzen da

// MAN + tracking egoeran, SP aktiboak PV-ari jarraitzen dio.
// 50.0 kontsignak GetOperatorSetpoint() bidez eskuragarri jarraitzen du.
```

Jauzirik gabeko transferentzia

AUTOrat pasatzean, Kc/Ki/Kd aldatzean, egitura aldatzean, ekintza alderantzizkatzean edo AUTO moduan irteera-mugak aldatzean, PIDControl-ek integrala hasieratzen du P + I + D batura uneko OP-tik ahalik eta hurbilen abiatzeko.

Hasieratze kontzeptuala

```
I = uneko_OP - P
```

Deribatuaren ekarpena zeroan hasieratzen da, previousPV eta previousError uneko balioekin eguneratzen direlako.

8. Saturazioa, anti-windup, mugak eta laginketa-denbora

Irteera-mugak

```
pid.SetOutputLimits(0.0, 100.0);
```

Kalkulatutako OP eta eskuzko OP konfiguratutako tartera mugatzen dira. Lehenetsitako mugak 0..100 % dira. AUTO moduan mugak aldatzen badira, jauzirik gabeko hasieratze berri bat egiten da.

Anti-windup

PIDControl-ek integrazio baldintzatua erabiltzen du. Integralaren balio berria onartu aurretik, saturatu gabeko OP kalkulatu du. Irteera maximoaren gainetik badago eta integralak are gehiago handituko balu, ez du integratzen. Logika bera erabiltzen da beheko mugan.

```
integralIncrement = sign * Ki * error * Ts;
newIntegral       = integral + integralIncrement;
unsaturatedOutput = P + newIntegral + D;

saturatu gabeko OP > max eta integralIncrement > 0 -> ez integratu
saturatu gabeko OP < min eta integralIncrement < 0 -> ez integratu
```

Laginketa-denbora

```
pid.SetSampleTime(100); // ms
```

Lehenetsitako laginketa-denbora 100 ms da. Compute() funtzioak kalkulu automatiko berri bat egiten du konfiguratutako denbora gutxienez igaro denean. Ekuazioak benetan igarotako denbora erabiltzen du, segundotan bihurtuta, eta ez du Ts konstante finko bat suposatzen.

9. Seinale-egokitzapena eta iragazkiak

PIDControlFilters kontrolagailutik independentea da eta seinalea PIDera edo identifikazio-saiakuntzetara bidali aurretik tratatzeko aukera ematen du. Helburua ez da ahalik eta kurbarik politena lortzea, baizik eta prozesuaren benetako dinamika ezkutatu gabe behar adina zarata kentzea.

Batez besteko mugikorra

```
MovingAverageFilter<10> iragazkia;
double pvIragazia = iragazkia.Update(pvGordina);
```

Ekuazioa

$$y[k] = (1/N) * \sum(x[k-i]), i=0..N-1$$

Ausazko zarata murrizten du, atzerapena gehitzearen truke. Hasieran, N lagin osatu arte benetan jasotako laginak soilik batez bestekatzen ditu.

Mediana

```
MedianFilter<5> iragazkia;
double pvIragazia = iragazkia.Update(pvGordina);
```

Oso eraginkorra da gailur isolatuen eta balio anomaloen aurka. Inplementazioak leiho bikoitiak eta bakoitiak onartzen ditu.

Lehen ordenako behe-paseko iragazkia

```
LowPassFilter iragazkia(1.0); // tau = 1 s
double pvIragazia = iragazkia.Update(pvGordina, dtSeconds);
```

Koefiziente diskretu zehatza

```
alpha = 1 - exp(-dt/tau)
y[k] = y[k-1] + alpha * (x[k] - y[k-1])
```

Parametro publikoa tau denbora-konstante fisikoa da, ez leuntze-faktore arbitrario bat.

Iragazki osagarria

```
ComplementaryFilter iragazkia(0.98);

double fusioa = iragazkia.Update(estimazioAzkarra, erreferentziaMotela);
double angelua = iragazkia.UpdateIntegrated(biraketaAbiadura, angeluAbsolutua, dt);
```

Iragazki osagarriak magnitude beraren bi estimazio behar ditu, propietate osagarriekin. Ez da seinale bakar baterako behe-paseko iragazki soil gisa interpretatu behar.

Iragazketa eta identifikazioa. Iragazki orok dinamika gehitzen du. Gehiegizko iragazketak identifikazioan prozesuak benetan baino denbora hil edo denbora-konstante handiagoa duela ematea eragin dezake. Behar dena bakarrik iragaztea komeni da.

10. Begizta irekiko identifikazioa StepTest bidez

StepTest-ek urrats-erantzuneko saiakuntza egiten du Arduino-ren loop-a blokeatu gabe. Helburua OP-ren aldaketa batetik edo gehiagotik FOPDT eredu bat estimatzea da.

```
StepTest test(pid, &PV, &OP);

test.SetStepSize(10.0);
test.SetNumberOfSteps(2);
test.SetFirstStepUp(true);
test.SetProcessGain(ProcessGain::POSITIVE);

test.SetStabilityCriteria(0.30, 60.0, 0.10);
test.SetResponseThreshold(0.40);
test.SetMaxTestTime(30.0);

test.Start();
```

loop-ean zehar:

```
test.Update();
pid.Compute();
```

Lan-sekuentzia

- Kontrolagailua MAN modura behartzen du.
- Uneko OP bias gisa gordetzen du.
- Hasierako egoera egonkorra itxaroten du.
- Konfiguraturako urratsak txandaka aplikatzen ditu.
- PV-k espero den noranzkoan erantzuten duela egiaztatzen du.
- Egoera egonkor berria itxaroten du.
- Baliozko urrats bakoitzerako eredu bat kalkulatzen du.
- Baliozko ereduaren batez bestekoa kalkulatzen du.
- Amaitzean edo bertan behera uztean, OP jatorrizko bias-era itzultzen du eta PID MAN moduan uzten du.

Egoera egonkorren irizpide sendoa

Aldi bereko baldintzak

```
PVmax - PVmin <= 2 * band
abs(erregresio_malda) <= maxSlope
```

Malda egonkortasun-leihoan karratu txikiaren erregresio linealaren bidez lortzen da, eta minutuko PV unitateetan adierazten da.

Noranzkoko erantzuna

ProcessGain::POSITIVE-k esan nahi du OP handitzeak azkenean PV handitu behar duela. NEGATIVE-k kontrakoa adierazten du. Irizpide horrek noranzko okerreko hondar-inertzia urratsaren erantzunarekin nahastea saihesten du.

11. FOPDT eredua eta irabazi normalizatua

StepTest-ek lehen ordenako eredua estimatzen du, denbora hilarekin:

Eredua

$$G(s) = K_p * \exp(-T_0 * s) / (T_p * s + 1)$$

Irabazia ingeniari-tarte-unitateetan

Prozesu-irabazia

$$K_p = \Delta PV / \Delta OP$$

T0 eta Tp-ren estimazioa

Implementazioak erantzunaren % 28,3 eta % 63,2ko gurutzaketak erabiltzen ditu:

28,3/63,2 metodoa

$$\begin{aligned} T_p &= 1.5 * (t_{63.2} - t_{28.3}) \\ T_0 &= 1.5 * t_{28.3} - 0.5 * t_{63.2} \end{aligned}$$

Irabazi normalizatua

Saiakuntzaren segurtasun-mugak eta ingeniari-tarte osoak kontzeptu desberdinak dira. Tarteak irabazia normalizatzeko bakarrik erabiltzen dira.

```
test.SetOutputLimits(0.0, 60.0); // saiakuntzaren segurtasun-muga
test.SetPVLimits(15.0, 45.0);   // saiakuntzaren segurtasun-muga

test.SetOutputRange(0.0, 100.0); // OP-ren tarte osoa
test.SetPVRRange(0.0, 80.0);    // PV-ren tarte osoa
```

Irabazi normalizatua

$$\begin{aligned} K_{p\text{Normalized}} &= (\Delta PV / PV_{\text{span}}) / (\Delta OP / OP_{\text{span}}) \\ &= K_p * OP_{\text{span}} / PV_{\text{span}} \end{aligned}$$

KpNormalized %PV/%OP gisa adierazten da eta bi ingeniari-tarteak konfiguratu direnean bakarrik ematen da.

FOPDTModel objektua

```
FOPDTModel model = test.GetModel();

model.valid
model.Kp
model.KpNormalizedValid
model.KpNormalized
model.T0
model.Tp
model.pvRangeMin
model.pvRangeMax
model.outputRangeMin
model.outputRangeMax
```

Objektu honek identifikazioa eta sintonizazioa bereizten ditu eta zuzenean PIDTuning-i pasa dakioke.

StepTest-en emaitza objektuan geratzen da saiakuntza amaitu ondoren ere. TCLab v1.4.2 sketch-ean IDENT STEP bidez berreskura daiteke, RESET exekutatzen ez den bitartean, plaka berrabiarazten ez den bitartean eta beste StepTest batek egoera gainidazten ez duen bitartean.

12. IMC/Lambda eta Tyreus-Luyben sintonizazioa PIDTuning bidez

PIDTuning-ek identifikazio-emaitzak kontrolagailu-parametro bihurtzen ditu automatikoki aplikatu gabe. FOPDT eredu batetik IMC/Lambda PI sintonizazioa eta Lambda PID sintonizazioa eskaintzen ditu. RelayTest bidez lortutako Ku eta Tu balioetatik Tyreus-Luyben PI eta PID sintonizazioak eskaintzen ditu.

```
PIDTuning tuning(model);

PIDTuningResult pi = tuning.IMC_PI();
PIDTuningResult pid = tuning.Lambda_PID();

PIDTuningResult piRobust = tuning.IMC_PI(IMCSpeed::ROBUST);
PIDTuningResult pidRobust = tuning.Lambda_PID(IMCSpeed::ROBUST);
```

IMC / Lambda PI

```
Kc = Tp / (abs(Kp) * (Tf + T0))
Ti = Tp
Td = 0
Ki = Kc / Ti
Kd = 0
```

Preset-a	Tf
IMCSpeed::AGGRESSIVE	$Tf = 1 \cdot T0$
IMCSpeed::NORMAL	$Tf = 2 \cdot T0$
IMCSpeed::ROBUST	$Tf = 3 \cdot T0$

Parametrorik gabeko overload-ek NORMAL preset-a erabiltzen dute; beraz, $Tf = 2 \cdot T0$ da bai PI bai PID kasurako. $T0$ zenbakiz zero bada, PIDControl-ek $0,1 \cdot Tp$ erabiltzen du ordeko balio gisa. AGGRESSIVE, NORMAL eta ROBUST liburutegiko preset praktikoak dira, ez metodoaren arau unibertsalak.

Lambda PID

Rivera/Lambda PID arauak FOPDT eredu bera eta begizta itxiaren nahi den erantzun-abiadura ezartzen duen Tf parametroa erabiltzen ditu:

```
Kc = (Tp + T0/2) / (abs(Kp) * (Tf + T0/2))
Ti = Tp + T0/2
Td = Tp*T0 / (2*Tp + T0)
Ki = Kc / Ti
Kd = Kc * Td
```

PIDControl-en, `Lambda_PID()` funtzioak Kc , Ti , Td , Ki eta Kd itzultzen ditu. `PIDTuningResult`-eko ``lambda`` eremuak izen hori mantentzen du API bateragarritasunagatik, baina dokumentazio honetan sintonizazioa kalkulatzeko erabilitako Tf balioa adierazten du.

```
PIDTuningResult pid = tuning.Lambda_PID();
PIDTuningResult pid60 = tuning.Lambda_PID(60.0);
PIDTuningResult pidRobust = tuning.Lambda_PID(IMCSpeed::ROBUST);
```

Kp-ren zeinua. PIDTuning-ek `abs(Kp)` erabiltzen du Kc positiboa kalkulatzeko. Kontrolagailuaren noranzkoa aparte kudeatzen da `PIDAction` bidez.

Sintonizazioa esplizituki aplikatzen da:

```
PIDTuning::Apply(pid, pi);
```

`Apply()` funtzioak Kc , Ki eta Kd bakarrik aldatzen ditu. Ez ditu egitura, modua, ekintza edo SP aldatzen.

13. Errele-saiakuntza RelayTest bidez

RelayTest-ek Åström-Hägglund-en errele bidezko autotuning printzipioan oinarritutako saiakuntza ez-blokeatzailea inplementatzen du. Bere eginkizuna Ku eta Tu identifikatzea da; balio horiek kontrolagailu-parametro bihurtzea PIDTuning-ek egiten du ondoren.

```
RelayTest relay(pid, &PV, &OP);

relay.SetRelayAmplitude(10.0);
relay.SetHysteresis(0.5);
relay.SetCycles(3);
relay.UseCurrentPVAsReference();
relay.SetMaxTestTime(15.0);
relay.SetOutputLimits(20.0, 60.0);
relay.SetPVLimits(15.0, 48.0);

relay.Start();
```

APIko konfigurazio trinkoa

v1.4.1 bertsiotik, RelayTest klaseak berak RelayTestConfig eta Start(config) eskaintzen ditu:

```
RelayTestConfig cfg;
cfg.amplitude = 10.0;
cfg.hysteresis = 0.5;
cfg.cycles = 3;
cfg.useCurrentReference = true;
cfg.maxTestTimeMinutes = 15.0;
cfg.outputMin = 20.0;
cfg.outputMax = 60.0;
cfg.pvMin = 15.0;
cfg.pvMax = 48.0;

relay.Start(cfg);
```

Desberdintasun garrantzitsua. RelayTestConfig liburutegiaren parte da. RELAY(...) testu-sintaxia sketch zehatz baten parser-aren ardura da; PIDControl-ek ez ditu Serial komandoak zuzenean interpretatzen.

Funtzionamendua

Errelearen mailak

```
OPhigh = bias + d
OPlow = bias - d
```

Bias-a saiakuntza hasten denean dagoen OP da. Lehenespenez, errelearen erreferentzia Start() unean dagoen PV da. REVERSE ekintzarako, normalean prozesu-irabazi positiboarekin lotuta, logika hau da:

```
PV >= erreferentzia + histeresia -> OP low
PV <= erreferentzia - histeresia -> OP high
```

DIRECT ekintzan konmutazio-noranzkoa alderantzikatzen da.

Ku eta Tu-ren kalkulua

Errele simetriko idealaren hurbilketa

```
a = (PVmax - PVmin) / 2
Ku = 4*d / (pi*a)
Tu = trantsizio baliokideen arteko batez besteko periodoa
```

Lehen oszilazio osoa hasierako trantsitorio gisa baztertzen da. SetCycles() bidez konfiguratutako kopurua ondorengo zikloi dagokie, batez bestekoetan erabiltzen direnak.

Emaitzaren iraunkortasuna memorian

GetResult() saiakuntza amaitu ondoren berriro kontsulta daiteke. TCLab v1.4.2 sketch-ean, IDENT RELAY-k Ku, Tu eta Tyreus-Luyben proposamenak berriz inprimatzen ditu emaitza memorian gordeta dagoen bitartean eta beste errele-saiakuntza batek berrezarri edo gainidatzi ez duen bitartean.

Sintonizazio integratua v1.4.2-n. RelayTest Ku eta Tu identifikatzera mugatzen da. PIDTuning-ek Tyreus-Luyben PI eta PID gehitzen ditu aparteko etapa gisa. Liburutegiak ez du sintonizaziorik automatikoki aplikatzen: erabiltzaileak erabakitzen du zein erabili.

TCLab-en benetako baliozkotze-adibidea

TCLab-en baliozkotzean, saiakuntza OP=40 % bias-etik egin zen, % 10eko errele-anplitudearekin, 0,5 °C-ko histeresia eta hiru ziklo baliodunekin. OP % 30 eta % 50 artean oszilatzen zen eta saiakuntza amaitzean jatorrizko bias-a berrezarri zen MAN moduan.

```
Ku = 14.233237
Tu = 180.300 s
PV anplitudea = 0.903066 °C
Errele-anplitudea = 10 %
Ziklo baliodunak = 3
```

Liburutegitik kanpoko egiaztapen gisa, balio horiek Ziegler-Nichols sintonizazioa kalkulatzeko erabili daitezke. PI-rako, $K_c = 0,45 \cdot K_u$ eta $T_i = T_u/1,2$ erabilia, gutxi gorabehera hau lortzen da:

```
Kc = 6.404957
Ti = 150.250 s
Ki = Kc/Ti = 0.042629 1/s
Kd = 0
```

Tyreus-Luyben PIDControl 1.4.2-n integratuta

Ku eta Tu beretatik, PIDTuning-ek zuzenean Ziegler-Nichols klasikoa baino bi aukera motelago eta kontserbadoreago eskaintzen ditu:

```
Tyreus-Luyben PI:
Kc = Ku / 3.2
Ti = 2.2*Tu
Ki = Kc/Ti

Tyreus-Luyben PID:
Kc = Ku / 3.3
Ti = 2.2*Tu
Td = Tu / 6.3
Ki = Kc/Ti
Kd = Kc*Td
```

Benetako saiakuntzan $K_u=14,233237$ eta $T_u=180,300$ s izanik, gutxi gorabehera hau lortzen da:

```
Tyreus-Luyben PI: Kc=4.447887 Ti=396.660 s Ki=0.011213 Kd=0
Tyreus-Luyben PID: Kc=4.313102 Ti=396.660 s Td=28.619 s Ki=0.010874 Kd=123.437
```

Aplikazioa esplizitua izaten jarraitzen du. PIDTuning::Apply() edo SetTunings() funtzioek erabiltzaileak eskatzen duenean bakarrik aldatzen dute sintonizazioa.

14. Saiakuntzen egoerak, erroreak eta portaera segurua

StepTestState

Egoera	Esanahia
IDLE	Ez dago saiakuntza aktiborik.

Egoera	Esanahia
WAITING_STABLE	Hasierako edo trantsizioko egoera egonkorren zain.
RUNNING_STEP	Urratsa aplikatuta eta erantzunaren zain.
WAITING_FINAL_STABLE	Erantzuna detektatuta; azken egoera egonkorren zain.
FINISHED	Identifikazioa amaituta.
ABORTED	Saiakuntza bertan behera utzita.

StepTestError

Errorea	Arrazoia
NONE	Errorerik ez.
INVALID_CONFIG	Konfigurazio baliogabea.
PV_LOW / PV_HIGH	PV saiakuntzaren mugetatik kanpo.
OP_LIMIT	Eskatutako OP saiakuntzaren mugetatik kanpo.
TIMEOUT	Gehieneko denbora gaindituta.
NO_RESPONSE	Ez da baliozko erantzunik detektatu.
INVALID_MODEL	Erantzuna egon da, baina ezin izan da baliozko eredurik lortu.
USER_ABORT	Aplikazioak/erabiltzaileak bertan behera utzita.

RelayTestState

Egoera	Esanahia
IDLE	Ez dago saiakuntza aktiborik.
RUNNING	Errelea martxan.
FINISHED	Ku/Tu kalkulatuta.
ABORTED	Saiakuntza bertan behera utzita.

RelayTestError

Errorea	Arrazoia
NONE	Errorerik ez.
INVALID_CONFIG	Parametro bateraezinak edo erreferentzia tartetik kanpo.
PV_LOW / PV_HIGH	PV mugetatik kanpo.
OP_LIMIT	Bias edo OP high/low mugetatik kanpo.
TIMEOUT	Zikloak badaude, baina eskatutakoak ez dira garaiz osatzen.

Errorea	Arrazoia
NO_OSCILLATION	Denbora agortzen da zikloak osatu gabe.
USER_ABORT	Aplikazioak/erabiltzaileak bertan behera utzita.

StepTest-ek eta RelayTest-ek kontrolagailua MAN modura itzultzen dute eta jatorrizko bias-a berrezartzen dute amaitzean edo bertan behera uztean. Reset() funtzioak egoera, erroreak eta emaitzak garbitzen ditu, objektuaren konfigurazioa mantenduta. TCLab sketch-eko RESET globalak, aldiz, bere balio lehenetsiak berriz aplikatzen ditu esplizituki.

15. TCLab sketch-a: eskuratzea, iragazketa, segurtasuna eta baliozkotzea

Liburutegia TCLab batean baliozkotu da Arduino UNO R4 WiFi erabiliz. Sketch hau ez da nukleo algoritmikoaren parte; hardwarea, segurtasuna, kontsola eta grafikoak integratzen dituen proba-aplikazioa da.

Elementua	Konfigurazioa
T1	A0
T2	A2
Q1	D3
Q2	D5
Seriea	115200 baud
ADC	14 bit, 3,300 V-ko kanpo-erreferentzia
Q1 100 %	PWM maximoa 200/255
Q2	PWM maximoa 100/255, uneko saiakuntzan 0an mantenduta
I/O periodoa	100 ms
Grafikoaren periodoa	500 ms

Eskuratze-katea

```

ADC
-> kanala aldatu ondorengo lehen bihurketa baztertu
-> 10 bihurketaren batez bestekoa
-> MedianFilter<5>
-> LowPassFilter(tau)
-> PID / StepTest / RelayTest-ek erabilitako PV

```

Sentsorearen bihurketak TMP36 erlazioa jarraitzen du:

Tenperatura

$$T [C] = (V - 0.5) * 100$$

Funtzionamendu-tartea eta babes termikoa

Kontzeptua	Uneko balioa
Funtzionamendu arrunteko SP	10..80 °C
Trip PV/T2 iragazian	>= 80 °C
Trip PV/T2 gordinean	>= 82 °C
Alarmaren berrezarpena	< 78 °C T1 eta T2-n
Trip ondorengo ekintza	MAN, OP=0; ez da automatikoki AUTOra itzultzen

TCLab-en babes termikoa StepTest eta RelayTest-en mugetatik independentea da. Hardwarean idatzi aurretik OP-ren azken clamp bat ere aplikatzen da.

16. TCLab sketch-aren serieko komando-interfazea

Ez nahastu APlA eta kontsola. Hurrengo komandoak ez dira PIDControl-en nukleoaren parte. TCLab sketch-ak inplementatutako testu-interfazea dira, liburutegia Serial/Arduino COM Plotter bidez probatzeko.

Komando orokorrak

Komandoa	Funtzioa	Adibidea
HELP	Laguntza erakusten du.	HELP
STATUS	PID, iragazki eta saiakuntzen egoera erakusten du.	STATUS
RESET	MAN, OP=0 eta sketch-aren balio lehenetsiak berrezartzen ditu.	RESET
STOP	Saiakuntzak bertan behera uzten ditu, MAN ezartzen du eta OP=0.	STOP
MAN [op]	MAN modura pasatzen da eta aukeran OP ezartzen du.	MAN 40
AUTO [sp]	AUTO modura pasatzen da eta aukeran SP ezartzen du.	AUTO 50
SP <valor>	Operadorearen SP eguneratzen du, 10..80 °C-ra mugatuta.	SP 50
OP <valor>	Eskuzko OP eguneratzen du.	OP 35
TUNE <Kc> <Ki> <Kd>	Sintonizazioa eguneratzen du.	TUNE 6.404957 0.042629 0
STRUCT PID PI_D I_PD	Egitura hautatzen du.	STRUCT PI_D
ACTION DIRECT REVERSE	Ekintza hautatzen du.	ACTION REVERSE
TRACK ON OFF	PV Tracking aktibatu/desaktibatzen du.	TRACK ON
OUTLIMITS <min> <max>	PID irteeraren mugak.	OUTLIMITS 0 100
TS <ms>	Laginketa-denbora.	TS 100
FILTER <tau>	Mediana(5) + LPF adierazitako tau-rekin.	FILTER 1.0
FILTER ON / OFF	Lehenetsitako LPF aktibatzen du edo desaktibatzen du mediana mantenduta.	FILTER OFF
IDENT [STEP RELAY]	Memorian gordetako identifikazio-emaitzak berriz erakusten ditu.	IDENT

StepTest komandoak

Komandoa	Funtzioa
STEP START / STEP ABORT	Konfiguraturako saiakuntza hasi edo bertan behera uzten du.
STEP SIZE <percent>	OP urratsaren tamaina.

Komandoa	Funtzioa
STEP N <1..10>	Urrats kopurua.
STEP DIR UP DOWN	Lehen urratsaren noranzkoa.
STEP GAIN POS NEG	Prozesu-irabaziaren espero den zeinua.
STEP STABLE <band> <time> [slope]	Egoera egonkoraren irizpidea.
STEP RESPONSE <deltaPV>	Noranzkoko gutxieneko erantzun-atalasea.
STEP TIME <minutes>	Gehieneko denbora.
STEP LIMITS <opMin> <opMax> <pvMin> <pvMax>	Saiakuntzaren segurtasun-mugak.
STEP RANGE <pvRangeMin> <pvRangeMax> <opRangeMin> <opRangeMax>	Kp normalizaturako ingeniartza-tarte osoak.

Formatu trinko estandarra:

```
STEP(size,n,UP|DOWN,POS|NEG,stableBand,stableTime,maxSlope,response,maxMin,opMin,opMax,pvMin,pvMax)
```

Ingeniartza-tarte aukerakoekin:

```
STEP(...,pvMin,pvMax,pvRangeMin,pvRangeMax,opRangeMin,opRangeMax)
```

Baliozkotutako adibidea:

```
STEP(10,2,UP,POS,0.3,60,0.10,0.4,30,0,60,15,45,0,80,0,100)
```

RelayTest komandoak

Komandoa	Funtzioa
RELAY START / RELAY ABORT	Saiakuntza hasi edo bertan behera uzten du.
RELAY AMP <percent>	d anplitudea bias-aren inguruan.
RELAY HYST <degC>	Kommutazio-histeresia.
RELAY CYCLES <n>	Batez bestekatzeko ziklo baliodunen kopurua.
RELAY REF CURRENT	Hasieran uneko PV erabiltzen du erreferentzia gisa.
RELAY REF <degC>	Erreferentzia finko bat erabiltzen du.
RELAY TIME <minutes>	Gehieneko denbora.
RELAY LIMITS <opMin> <opMax> <pvMin> <pvMax>	Segurtasun-mugak.
RELAY(amp,hyst,cycles,CURRENT ref,maxMin,opMin,opMax,pvMin,pvMax)	RelayTest komando bakarrean konfiguratu eta abiarazten du.

Formatu trinkoa v1.4.2-n. TCLab sketch-ak RELAY(...) interpretatzen du eta parametroak RelayTestConfig bihurtzen ditu. Saiakuntzaren logika, baliozkotzea eta kalkulua PIDControl barruan jarraitzen dute. STEP(...) lehendik duen formatu trinkoan mantentzen da.

RelayTest formatu trinkoa

TCLab-en baliozkotutako formatu trinkoa hau da:

```
RELAY(10,0.5,3,CURRENT,15,20,60,15,48)
```

Honen baliokidea da: ± 10 %OP errele-anplitudea, $\pm 0,5$ °C-ko histeresia, hiru ziklo balioudun, uneko PV-ren berdina den erreferentzia, 15 min-ko timeout-a, 20..60 % arteko OP baimendua eta 15..48 °C arteko PV baimendua.

Emaitzak IDENT bidez berreskuratzea

StepTest edo RelayTest amaitu ondoren, kalkulaturako emaitza memorian gordeta geratzen da. Serieko portuaren log-a garbitu edo galtzen bada, emaitza berriro eska daiteke saiakuntza errepikatu gabe:

```
IDENT
IDENT STEP
IDENT RELAY
```

IDENT-ek gordetako StepTest-eko Kp, T0 eta Tp erakusten ditu, edo gordetako RelayTest-eko Ku eta Tu. Eskuragarri dauden sintonizazio-proposamenak ere berriz inprimatzen ditu. Memoria lurrunkorra da: RESET-ek, plaka berrabiarazteak/elikadura galtzeak edo mota bereko beste saiakuntza batek datu horiek ezabatu edo gainidazten ditu.

Arduino COM Plotter-erako trama

```
#SP PV OP T2
```

Trama '#'-rekin hasten da, zuriunea erabiltzen du bereizle gisa eta lau aldagai ditu.

17. Gomendatutako lan-fluxuak

Funtzionamendu normala

```
pid.SetOutputLimits(0.0, 100.0);
pid.SetSampleTime(100);
pid.SetStructure(PIDType::PI_D);
pid.SetDirection(PIDAction::REVERSE);
pid.SetPVTracking(true);

pid.SetManualOutput(40.0);
pid.SetMode(PIDMode::MAN);

// Behar denean:
pid.SetMode(PIDMode::AUTO);
pid.SetSetpoint(50.0);
```

FOPDT identifikazioa + IMC/Lambda

1. Prozesua MAN moduan egonkortu.
2. StepTest-en mugak eta tarteak konfiguratu.
3. test.Start() exekutatu eta loop()-ean test.Update() deitu.
4. FOPDTModel model = test.GetModel() lortu.
5. IMC/Lambda PI proposamena eta Lambda PID proposamena kalkulatu.
6. Kp, T0, Tp eta saiakuntzaren koherentzia berrikusi.
7. Hautatutako sintonizazioa esplizituki aplikatu.
8. MANetik AUTOrak pasatu PV Tracking erabiliz.
9. TCLab sketch-aren log-a galtzen bada, IDENT STEP erabili emaitza memorian dagoen bitartean.

Errele-saiakuntza

1. Prozesua MAN moduan egonkortu bias egoki batekin.
2. Anplitudea eta histeresia aukeratu.
3. bias+d eta bias-d mugen barruan daudela egiaztatu.
4. relay.Start(config) edo RELAY(...) komando trinkoa exekutatu.
5. loop()-ean relay.Update() eta pid.Compute() deitu.
6. Ku eta Tu lortu.
7. Tyreus-Luyben PI eta PID kalkulatu PIDTuning bidez.
8. Saiakuntzak bias-a berrezartzen du eta MAN moduan geratzen da.
9. TCLab sketch-aren log-a galtzen bada, IDENT RELAY erabili emaitza memorian dagoen bitartean.

Segurtasuna. Identifikazio-saiakuntzek plantaren edo maketaren muga fisiko absolutuak baino kontserbadoreagoak diren muga propioak izan behar dituzte. Gainera, hardware/aplikazio babes independente batek aktibo jarraitu behar du.

18. APIaren erreferentzia osoa

PIDControl

Metodoa	Deskribapena
PIDControl(...)	Eraikitzailea PV/OP/SP erakusleekin, Kc/Ki/Kd, egitura, ekintza eta tracking-arekin.
Compute()	MAN/tracking edo AUTO kalkulua exekutatzen du dagokionean.
SetMode()/GetMode()	MAN edo AUTO aldatu/kontsultatzen du.
SetManualOutput()/GetManualOutput()	Eskuzko OP ezarri/kontsultatzen du.
SetTunings()	Kc, Ki eta Kd ezartzen ditu; parametro negatiboak baztertzen ditu.
GetKc()/GetKi()/GetKd()	Parametroak itzultzen ditu.
SetStructure()/GetStructure()	PID, PI_D edo I_PD.
SetDirection()/GetDirection()	DIRECT edo REVERSE.
SetPVTracking()/GetPVTracking()	Tracking-a aktibatu/kontsultatzen du.
SetSetpoint()/GetOperatorSetpoint()	Operadorearen kontsigna eguneratu/kontsultatzen du.
SetOutputLimits()	OP-ren mugak konfiguratzeko ditu.
GetOutputMinimum()/Maximum()	Mugak kontsultatzen ditu.
SetSampleTime()/GetSampleTime()	Kalkulu-periodoa ms-tan konfiguratu/kontsultatzen du.

StepTest

Metodoa	Deskribapena
SetOutputLimits()	OP mugak saiakuntzan.
SetPVLimits()	PV mugak saiakuntzan.
SetOutputRange()/SetPVRRange()	Normalizaziorako tarte osoak.
HasEngineeringRanges()	Bi tarteak konfiguratuta dauden adierazten du.
SetStepSize()	Urratsaren tamaina.
SetNumberOfSteps()	Urrats kopurua, lehenetsitako maximoa 10.
SetFirstStepUp()	Hasierako noranzkoa.
SetProcessGain()	POSITIVE/NEGATIVE.
SetStabilityCriteria()	Banda, denbora eta aukerako malda maximoa.
SetMaxStabilitySlope()	Malda maximoa.
SetResponseThreshold()	Erantzun-atalasea.
SetMaxTestTime()/GetMaxTestTime()	Timeout-a minututan.

Metodoa	Deskribapena
Start()/Update()/Abort()/Reset()	Bizi-zikloa.
GetState()/GetError()	Egoera eta errorea.
GetResult()	Emaitza agregatua.
GetModel()	Transferi daitekeen FOPDTModel.
GetStepResult()	Urrats baten emaitza indibiduala.
GetCurrentStabilitySlope()	Egonkortasun-leihoaren uneko malda.
GetBiasOutput()/GetCurrentTargetOutput()	Bias eta target OP.
GetCurrentStepIndex()	Uneko urratsaren indizea.

RelayTest

Metodoa	Deskribapena
SetOutputLimits()/SetPVLimits()	Saiakuntzaren segurtasun-mugak.
SetRelayAmplitude()	d anplitudea.
SetHysteresis()	Histeresia PV-n.
SetCycles()	Batez bestekatzeko ziklo baliodunak.
SetReference()	Erreferentzia finkoa.
UseCurrentPVAsReference()	Erreferentzia = PV hasieran.
SetMaxTestTime()/GetMaxTestTime()	Timeout-a minututan.
Start()	Aurrez gordetako konfigurazioarekin abiarazten da.
Start(const RelayTestConfig&)	Dei bakarrean konfiguratu eta abiarazten da.
Update()/Abort()/Reset()	Bizi-zikloa.
IsRunning()/IsFinished()/IsAborted()	Egoera-kontsultak.
GetState()/GetError()/GetResult()	Egoera, errorea eta Ku/Tu.

PIDTuning

Metodoa	Deskribapena
PIDTuning(model)	FOPDTModel-etik abiatutako eraikitzailea.
IMC_PI()	IMC-PI lehenetsitako Tf-rekin.
IMC_PI(Tf)	IMC-PI Tf esplizituarekin.
IMC_PI(IMCSpeed)	IMC-PI Tf preset-arekin.

Metodoa	Deskribapena
DefaultLambda()	Lehenetsitako Tf itzultzen du; izena API bateragarritasunagatik mantentzen da.
GetModel()	Gordetako eredua itzultzen du.
Apply(pid,result)	Baliozko emaitza baten Kc/Ki/Kd aplikatzen ditu.
Lambda_PI() / Lambda_PI(Tf) / Lambda_PI(IMCSpeed)	IMC/Lambda PI araurako alias esplizitua.
Lambda_PID() / Lambda_PID(Tf) / Lambda_PID(IMCSpeed)	Lambda PID kalkulatu du FOPDTtik eta Kc, Ti, Td, Ki eta Kd itzultzen ditu.
TyreusLuybenPI(Ku,Tu)	Tyreus-Luyben PI kalkulatu du Ku eta Tu-tik.
TyreusLuybenPID(Ku,Tu)	Tyreus-Luyben PID kalkulatu du Ku eta Tu-tik.

PIDControlFilters

Klasea	Metodo nagusiak
MovingAverageFilter<N>	Reset(), Reset(x0), Update(x), Value(), IsInitialized(), Count(), WindowSize().
MedianFilter<N>	Reset(), Reset(x0), Update(x), Value(), IsInitialized(), Count(), WindowSize().
LowPassFilter	SetTimeConstant(), GetTimeConstant(), Reset(), Update(x,dt), Value(), IsInitialized().
ComplementaryFilter	SetAlpha(), GetAlpha(), Reset(), Update(), UpdateIntegrated(), Value(), IsInitialized().

19. Mugak eta jardunbide egokiak

- Kontrolagailuak ez ditu plantaren fisika edo unitateak ezagutzen; PV eta OP eskalatzea aplikazioaren ardura da.
- Prozesuaren babesa PIDetik kanpo inplementatu behar da eta kalkulaturako irteeraren gainetik lehentasuna izan behar du.
- StepTest-ek gutxi gorabehera monotonoa den eta probaturako eremuan FOPDT bidez irudika daitekeen erantzuna suposatzen du.
- Prozesu integratzaileek, oszilazio handikoek, alderantzizko erantzuna dutenek edo oso ez-linealek beste identifikazio-metodo bat behar dezakete.
- StepTest-eko ProcessGain zeinuak OP→PV erlazio fisikoarekin bat etorri behar du.
- RelayTest-ek prozesuaren zeinua PIDAction-etik ondorioztatzen du: REVERSE positibotzat hartzen da eta DIRECT negatibotzat.
- Errelearen histeresia txikiegia bada zaratak konmutazioa eragin dezake; handiegia bada saiakuntza deformatzen du.
- Gehiegizko iragazketak behatutako dinamika aldatzen du eta T0 handitu dezake.
- AUTO moduan sintonizazioa, egitura, ekintza edo mugak aldatzeak bumpless portaera du, baina aldaketa garrantzitsu baten ondoren prozesua gainbegiratzea komeni da.
- Identifikazio-emaitzak saiakuntza egin den eragiketa-eremurako baliozkoak dira; ez dute linealtasun globala bermatzen.

Ereduaren kalitateari buruz. StepTest baliozko eredu batekin amaitzeak algoritmoak kalkulatu ahal izan duela esan nahi du. Aplikazio zorrotzetan komeni da hainbat urrats edo saiakuntza alderatzea eta Kp, T0 eta Tp-ren errepikagarritasuna ebaluatzea sintonizazio bat behin betikotzat hartu aurretik.

20. Paketearen egitura eta biltegirako prestaketa

```
PIDControl/
|-- library.properties
|-- keywords.txt
|-- README.md
|-- CHANGELOG.md
|-- src/
|   |-- PIDControl.h / .cpp
|   |-- StepTest.h / .cpp
|   |-- RelayTest.h / .cpp
|   |-- PIDTuning.h / .cpp
|   `-- PIDControlFilters.h
|-- examples/
|   |-- BasicPID/
|   |-- ManualAuto/
|   |-- StepTestIdentification/
|   |-- StepTestAnalogIO/
|   |-- RelayAutoTune/
|   |-- IMCTuning/
|   `-- FilteringPV/
|-- docs/
`-- extras/
```

Instalazioa

PIDControl Arduino Library Manager-en eskuragarri dago eta Arduino IDEko liburutegi-kudeatzailetik zuzenean instala daiteke `PIDControl` bilatuz. Eskuz ere instala daiteke GitHub-etik edo proiektuaren orritik ZIP fitxategia deskargatuta eta Sketch > Include Library > Add .ZIP Library hautatuta.

- <https://github.com/deurus/PIDControl>
- <https://garikoitz.info/blog/descargas/PIDControl/PIDControl-1.4.2.zip>

A eranskina. Gutxieneko adibide osoa

```
#include <PIDControl.h>

double SP = 50.0;
double PV = 0.0;
double OP = 0.0;

const bool pvTracking = true;

PIDControl pid(
    &PV, &OP, &SP,
    1.0, 0.4, 0.0,
    PIDType::PI_D,
    PIDAction::REVERSE,
    pvTracking
);

void setup()
{
    pid.SetSampleTime(100);
    pid.SetOutputLimits(0.0, 100.0);
    pid.SetManualOutput(35.0);
    pid.SetMode(PIDMode::MAN);
}

void loop()
{
    PV = PVIrakurri();

    pid.Compute();

    OPIdatzi(OP);
}
```

Adibideak nahita uzten ditu kanpoan PVIrakurri() eta OPIdatzi(). Bereizketa hori diseinuaren parte da: PIDControl-ek ingeniarietza-unitateekin lan egiten du eta ez dago sentsore edo eragingailu mota jakin bati lotuta.

B eranskina. Laburpen azkarra

Beharra	Klasea / funtzioa
PV bat kontrolatu	PIDControl
Manual/automatikoan lan egin	SetMode(), SetManualOutput()
MAN→AUTO jauziak saihestu	PV Tracking + bumpless
Egitura aldatu	PIDType::PID / PI_D / I_PD
Kontrol-noranzkoa aldatu	PIDAction::DIRECT / REVERSE
Windup-a saihestu	Barneko integrazio baldintzatua
Zarata iragazi	PIDControlFilters
Kp/T0/Tp identifikatu	StepTest
Irabazia normalizatu	SetPVRange() + SetOutputRange()
PI IMC bidez sintonizatu	PIDTuning
Ku/Tu lortu	RelayTest
RelayTest dei bakarrean konfiguratu	RelayTestConfig + Start(config)
PI FOPDTtik lortu	PIDTuning::IMC_PI() / Lambda_PI()
PID FOPDTtik lortu	PIDTuning::Lambda_PID()
PI/PID Ku/Tutik lortu	PIDTuning::TyreusLuybenPI()/PID()
Sketch-aren identifikazioa berreskuratu	IDENT / IDENT STEP / IDENT RELAY

Arkitektura honekin, sentso-re-irakurketa zaratatsu batetik kontrolagailu sintonizatu batera pasa daiteke, kontrol-logika hardware zehatzari lotu gabe.