advanced-pid Documentation

Release latest

eadali

Jun 29, 2022

CONTENTS

CHAPTER

ONE

ADVANCED-PID

An advanced PID controller in Python. The derivative term can also be used in practice thanks to built-in first-order filter. Detailed information can be found here.

Usage is very simple:

```
from advanced_pid import PID

# Create PID controller
pid = PID(Kp=2.0, Ki=0.1, Kd=1.0, Tf=0.05)

# Control loop
while True:
    # Get current measurement from system
    timestamp, measurement = system.get_measurement()

    # Calculate control signal by using PID controller
    reference = 1.0
    control = pid(timestamp, reference - measurement)

    # Feed control signal to system
    system.set_input(control)
```

Complete API documentation can be found here.

1.1 Usage

Biggest advantage of advanced-pid, the derivative term has a built-in first-order filter. advanced-pid package includes a toy mass-spring-damper system model for testing:

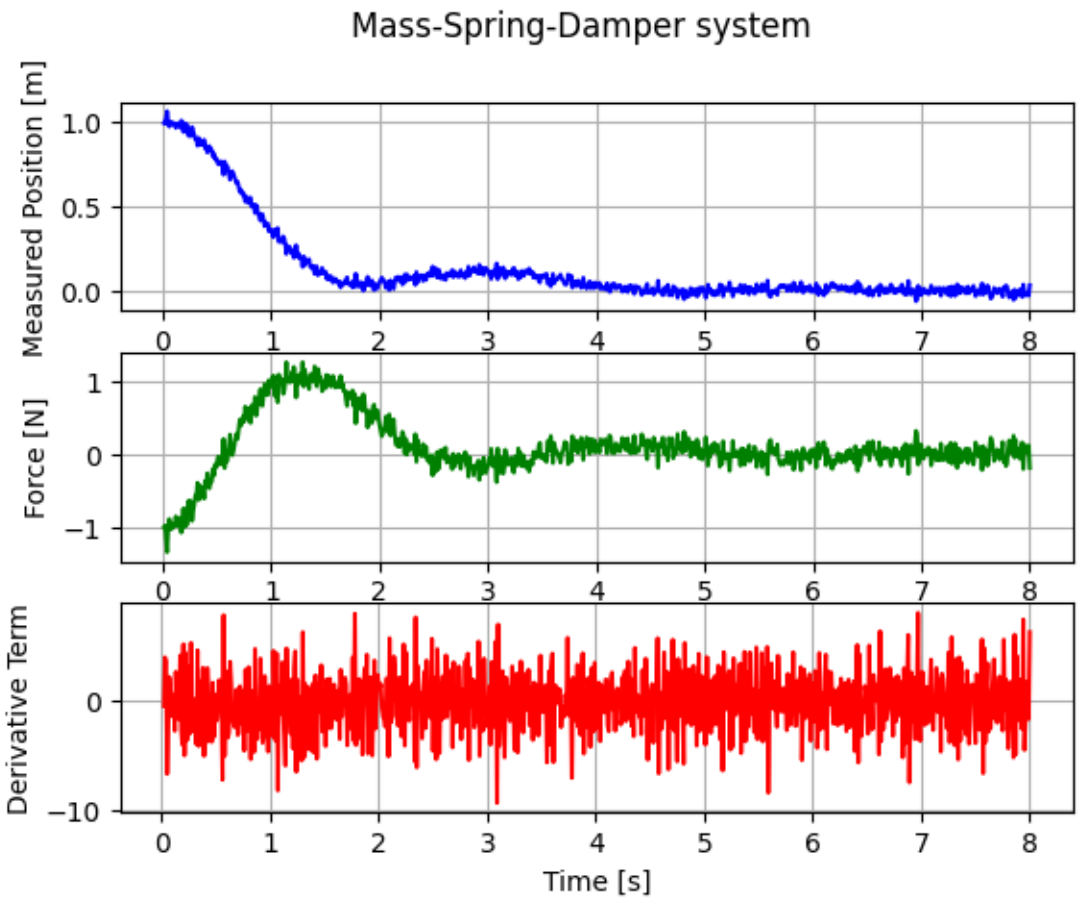
```
from advanced_pid import PID
from advanced_pid.models import MassSpringDamper
from matplotlib import pyplot as plt
from numpy import diff

# Create a mass-spring-damper system model
system = MassSpringDamper(mass=1.0, spring_const=1.0, damping_const=0.2)
system.set_initial_value(initial_position=1.0, initial_velocity=0.0)

# Create PID controller
```

(continues on next page)

(continued from previous page)

```
pid = PID(Kp=1.0, Ki=0.0, Kd=2.0, Tf=0.5)

# Control loop
time, meas, cont = [], [], []
for i in range(800):
    # Get current measurement from system
    timestamp, measurement = system.get_measurement()

    # Calculate control signal by using PID controller
    control = pid(timestamp, -measurement)

    # Feed control signal to system
    system.set_input(control)

    # Record for plotting
    time.append(timestamp)
    meas.append(measurement)
    cont.append(control)

# Plot result
fig, (ax1, ax2, ax3) = plt.subplots(3, 1)
fig.suptitle('Mass-Spring-Damper system')
ax1.set_ylabel('Measured Position [m]')
ax1.plot(time, meas, 'b')
ax1.grid()
ax2.set_ylabel('Force [N]')
ax2.plot(time, cont, 'g')
ax2.grid()
ax3.set_xlabel('Time [s]')
ax3.set_ylabel('Derivative Term')
ax3.plot(time[1:], diff(meas)/diff(time), 'r')
ax3.grid()
plt.show()
```

As It can be seen in the figure, derivative term cannot be use without a filter:



1.2 Installation

To install, run:

```
pip3 install advanced-pid
```

1.3 Tests

To run tests, run:

```
python -m unittest tests.test_pid
```

1.4 License

Licensed under the MIT License.

CHAPTER

TWO

COMPLETE API DOCUMENTATION

class advanced_pid.**PID**(*Kp*, *Ki*, *Kd*, *Tf*)

An advanced PID controller with first-order filter on derivative term.

Parameters

- **Kp** (*float*) – Proportional gain.
- **Ki** (*float*) – Integral gain.
- **Kd** (*float*) – Derivative gain.
- **Tf** (*float*) – Time constant of the first-order derivative filter.

__call__(*t*, *e*)

Call integrate method.

Parameters

- **t** (*float*) – Current time.
- **e** (*float*) – Error signal.

Returns
Control signal.

Return type
float

get_gains()

Get PID controller gains.

Returns
Gains of PID controller (Kp, Ki, Kd, Tf).

Return type
tuple

get_initial_value()

Get PID controller states.

Returns
Initial states of PID controller (t0, e0, i0)

Return type
tuple

`get_output_limits()`

Get PID controller output limits for anti-windup.

Returns
Output limits (lower, upper).

Return type
tuple

`integrate(t, e)`

Calculates PID controller output.

Parameters

- **t** (*float*) – Current time.
- **e** (*float*) – Error signal.

Returns
Control signal.

Return type
float

`set_gains(Kp, Ki, Kd, Tf)`

Set PID controller gains.

Parameters

- **Kp** (*float*) – Proportional gain.
- **Ki** (*float*) – Integral gain.
- **Kd** (*float*) – Derivative gain.
- **Tf** (*float*) – Time constant of the first-order derivative filter.

`set_initial_value(t0, e0, i0)`

Set PID controller states.

Parameters

- **t0** (*float or None*) – Initial time. None will reset time.
- **e0** (*float or None*) – Initial error. None will reset error.
- **i0** (*float or None*) – Inital integral. None will reset integral.

`set_output_limits(lower, upper)`

Set PID controller output limits for anti-windup.

Parameters

- **lower** (*float or None*) – Lower limit for anti-windup,
- **upper** (*flaot or None*) – Upper limit for anti-windup.

INDEX

Symbols

G

I

P

S