

---

# **PID\_control Documentation**

***Release 0.0.1***

**Philipp Schuette, Daniel Schuette**

**May 19, 2019**



**CONTENTS:**

<b>1</b>	<b>Indices and tables</b>	<b>7</b>
	<b>Python Module Index</b>	<b>9</b>
	<b>Index</b>	<b>11</b>



```
class PID_control.AnimatedPendulum (phi0, phi0_dot, alpha, beta, mu, max_control, frequency,  
                                     deadband, set_point, precision, t_start, t_end, N, L, f)
```

Class especially tailored to creating, solving and finally animating a controlled inverted pendulum. While the Pendulum class method `animate()` can be called after solving the pendulum ODE with `solve()`, the AnimatedPendulum class wraps these steps. Simply create an instance and call the `animate()` method.

```
__init__ (phi0, phi0_dot, alpha, beta, mu, max_control, frequency, deadband, set_point, precision,  
          t_start, t_end, N, L, f)
```

#### Parameters

- **phi0** (*float*) – initial angle value
- **phi0\_dot** (*float*) – initial angle velocity value
- **alpha** (*float* > 0) – proportional control parameter
- **beta** (*float* > 0) – derivative control parameter
- **mu** (*float* > 0) – integral control parameter
- **max\_control** (*float* > 0) – controller output bound
- **frequency** (*int* >= 1) – controller speed parameter
- **deadband** (*float*) – minimum difference between calculated control outputs
- **set\_point** (*float*) – desired value of controlled system
- **precision** (*int* > 0) – measurement precision of controller input
- **t\_start** (*float*) – starting time for pendulum dynamics
- **t\_end** (*float* > *t\_start*) – ending time for pendulum dynamics
- **N** (*int* > 0) – number of support points
- **L** (*float* > 0) – pendulum length
- **f** (*function*) – right hand side for pendulum ODE

**Output** Creates an instance of a pendulum, waiting to be animated

```
__module__ = 'PID_control'
```

```
animate (anim_name)
```

Animates an instance of AnimatedPendulum.

**Parameters** **anim\_name** (*string*) – name for the mp4 file, in which the animation gets stored

```
class PID_control.PIDControl (alpha, beta, mu, frequency, max_control, set_point, deadband)
```

Class implementation of a PID controller. This class is initialize from given data like parameters for P, I and D, controller speed with respect to the modelled system and a bound for controller ouput. Especially the latter two are quite important in practical applications. The set point, that the controller tries to reach, can also be adjusted manually. A parameter deadband reduces strain on possible machinery behind the controller; the control output is only changed, if the newly calculated output differs from the previous one by at least the amount specified by deadband.

```
__init__ (alpha, beta, mu, frequency, max_control, set_point, deadband)
```

Initialize PIDControl class.

#### Parameters

- **alpha** (*float* > 0) – proportional control parameter
- **beta** (*float* > 0) – derivative control parameter
- **mu** (*float* > 0) – integral control parameter

- **frequency** (*int*  $\geq 1$ ) – controller speed parameter
- **max\_control** (*float*  $> 0$ ) – controller output bound
- **set\_point** (*float* between 0 and  $2\pi$ ) – desired value of controlled system
- **deadband** (*float*  $> 0$ ) – minimum difference for controller adjustment

```
>>> import numpy as np; from PID_control import *
>>> ALPHA = 4.4; BETA = 2.0; MU = 1.2; MAX_CONTROL = 2.6
>>> FREQUENCY = 30; DEADBAND = 0.01; SET_POINT = -0.0*np.pi
>>> controller = PIDControl(
... ALPHA, BETA, MU, FREQUENCY, MAX_CONTROL, SET_POINT, DEADBAND
... )
>>> print(controller)
PID Controller with alpha = 4.4, beta = 2.0, mu = 1.2
```

**\_\_module\_\_** = 'PID\_control'

**\_\_repr\_\_** ()

Return string representation of PID controller.

**derivative\_output** (*x1*, *x2*, *t1*, *t2*)

Method returning the derivative or D controller output, depending on the attribute beta. A numerical approximation of the derivative value is necessary to compute the output. The trapezoid rule was chosen for that.

#### Parameters

- **x1** (*float*) – system value at time t1
- **x2** (*float*) – system value at time t2
- **t1** (*float*) – last time point
- **t2** (*float*) – current time point

**Output** derivative control value

**integral\_output** (*x1*, *x2*, *t1*, *t2*)

Method returning the integral or I controller output, depending on the controller attribute mu. A numerical approximation of the integral value is necessary to compute the output.

#### Parameters

- **x1** (*float*) – system value at time t1
- **x2** (*float*) – system value at time t2
- **t1** (*float*) – last time point
- **t2** (*float*) – current time point

**Output** integral control value

**proportional\_output** (*x*, *t*)

Method returning the proportional or P controller output, depending on the controller attribute alpha.

#### Parameters

- **x** (*float*) – system value at time t
- **t** (*float*) – current time point

**Output** Proportional control value

**total\_output** (*x1, x2, t1, t2, precision=4*)

Method returning the total controller output in response to system value *x* at time *t*. The output is bounded by the `max_control` attribute and controller adjustment speed is bounded by the `frequency` attribute. Controller does not adjust, if successive outputs differ by less than the `deadband` attribute. Limited measurement precision is included by rounding controller input.

#### Parameters

- **x1** (*float*) – system value at time *t1*
- **x2** (*float*) – system value at time *t2*
- **t1** (*float*) – last time point
- **t2** (*float*) – current time point
- **precision** (*int > 0*) – system measurement precision

**Output** total control value

**class** `PID_control.Pendulum` (*t\_start, t\_end, N, f, L=0.1, G=9.81*)

Class implementation of a simple ODE solver for the inverted pendulum. Intended to be used in conjunction with the `PIDControl` class. The solved ODE is of the form  $x'(t) = f(x(t)) + u(t)$ .

Initialize with time parameters *t\_start*, *t\_end*, number of support points *N*. Initial values are given to the `solve` method for flexibility in calculating solutions for different initial conditions. Right hand side can be adjusted, even though sine and identity are the most reasonable choices.

**\_\_init\_\_** (*t\_start, t\_end, N, f, L=0.1, G=9.81*)

Initialize linear or nonlinear pendulum.

#### Parameters

- **t\_start** (*float*) – starting time for ODE solving
- **t\_end** (*float > t\_start*) – final time for ODE solving
- **N** (*int > 0*) – number of numerical support points for ODE solving
- **f** (*function*) – right hand side for the second order pendulum ODE
- **L** (*float*) – pendulum length parameter in [m]
- **G** (*float > 0*) – gravitational constant in [m/s<sup>2</sup>]

**Output** object representing the second order pendulum ODE with right hand side *f*

**\_\_module\_\_** = `'PID_control'`

**\_\_repr\_\_** ()

Return string representation of inverted pendulum.

**animate** (*anim\_name*)

Method to animate solutions generated with `Pendulum` class. One needs to call the `solve()` method, before `plot()` makes sense.

**Parameters** **anim\_name** (*string*) – name for the mp4 file, in which the animation gets stored

**get\_func\_values** ()

A convenience method that returns the calculated function values or an empty list if `solve` was never called on this pendulum.

**Output** array containing angle solution values (floats)

### **get\_support\_values()**

A convenience method that returns the support values (== time points) or an empty list if *solve* was never called on this pendulum.

**Output** array containing support values (floats)

### **plot** (*file\_name*, *parameter=False*)

Method to plot solutions generated with Pendulum class. One needs to call the *solve()* method, before *plot()* can be called. PID Parameters can be written in the filename. This might be useful for numerical experiments with several distinct sets of parameters.

#### **Parameters**

- **file\_name** (*string*) – name for the .png file, in which the solution gets stored
- **parameter** (*bool*) – if true, controller parameters are written in the filename

**Todo** fix naming scheme (Linear vs Nonlinear; testing for  $(\lambda x: x) == \text{self.f}$  is not implemented very nicely).

### **solve** (*phi0*, *phi0\_dot*, *alpha*, *beta*, *mu*, *max\_control*, *frequency*, *deadband*, *set\_point*, *precision*)

Method solving the ODE for given physical initial conditions, i.e. initial angle and velocity, and with PID controller, that has the given parameters.

#### **Parameters**

- **phi0** (*float*) – initial value
- **phi0\_dot** (*float*) – initial angular velocity
- **alpha** (*float* > 0) – proportional control parameter
- **beta** (*float* > 0) – derivative control parameter
- **mu** (*float* > 0) – integral control parameter
- **max\_control** (*float* > 0) – controller output bound
- **frequency** (*int* >= 1) – controller speed parameter
- **deadband** (*float*) – minimum difference between calculated control outputs
- **set\_point** (*float*) – desired value of controlled system
- **precision** (*int* > 0) – measurement precision of controller input

**Output** numerically calculates the attributes (float arrays) *phi*, *output\_array*, *P\_array*, *I\_array* and *D\_array*

### **solve\_from\_angles** (*phi0*, *phi1*, *alpha*, *beta*, *mu*, *max\_control*, *frequency*, *deadband*, *set\_point*, *precision*)

Method solving the ODE for given numerical initial conditions, i.e. two initial angles. Works exactly like the *solve()* method.

#### **Parameters**

- **phi0** (*float*) – first initial value
- **phi1** (*float*) – second initial value
- **alpha** (*float* > 0) – proportional control parameter
- **beta** (*float* > 0) – derivative control parameter
- **mu** (*float* > 0) – integral control parameter
- **max\_control** (*float* > 0) – controller output bound



- **frequency** (*int*  $\geq 1$ ) – controller speed parameter
- **deadband** (*float*) – minimum difference between calculated control outputs
- **set\_point** (*float*) – desired value of controlled system
- **precision** (*int*  $> 0$ ) – measurement precision of controller input

**Output** Numerically calculates the attributes (float arrays) phi, output\_array, P\_array, I\_array and D\_array

```
>>> import numpy as np; from PID_control import *
>>> ALPHA = 4.4; BETA = 2.0; MU = 1.2; MAX_CONTROL = 2.6
>>> FREQUENCY = 30; DEADBAND = 0.01; SET_POINT = -0.0*np.pi
>>> PRECISION = 5; t_start = 0.0; t_end = 45.0; N = 9000; LENGTH = 0.1
>>> phi0 = 0.5 * np.pi; phi0_dot = 0.3 * np.pi
>>> pendulum = Pendulum(t_start, t_end, N, np.sin, L=LENGTH)
>>> pendulum
Inverted Pendulum of Length 0.1
>>> phi1 = phi0 + phi0_dot*pendulum.h
>>> pendulum.solve(
... phi0, phi0_dot, ALPHA, BETA, MU, MAX_CONTROL, FREQUENCY, DEADBAND,
... SET_POINT, PRECISION
... )
>>> x = pendulum.phi[10]
>>> pendulum.solve_from_angles(
... phi0, phi1, ALPHA, BETA, MU, MAX_CONTROL, FREQUENCY, DEADBAND,
... SET_POINT, PRECISION
... )
>>> y = pendulum.phi[10]
>>> print(np.abs(x - y) < 5e-5)
True
```



## INDICES AND TABLES

- `genindex`
- `modindex`
- `search`



## PYTHON MODULE INDEX

### p

`PID_control`, [1](#)



## Symbols

[\\_\\_init\\_\\_\(\) \(PID\\_control.AnimatedPendulum method\), 1](#)  
[\\_\\_init\\_\\_\(\) \(PID\\_control.PIDControl method\), 1](#)  
[\\_\\_init\\_\\_\(\) \(PID\\_control.Pendulum method\), 3](#)  
[\\_\\_module\\_\\_ \(PID\\_control.AnimatedPendulum attribute\), 1](#)  
[\\_\\_module\\_\\_ \(PID\\_control.PIDControl attribute\), 2](#)  
[\\_\\_module\\_\\_ \(PID\\_control.Pendulum attribute\), 3](#)  
[\\_\\_repr\\_\\_\(\) \(PID\\_control.PIDControl method\), 2](#)  
[\\_\\_repr\\_\\_\(\) \(PID\\_control.Pendulum method\), 3](#)

## A

[animate\(\) \(PID\\_control.AnimatedPendulum method\), 1](#)  
[animate\(\) \(PID\\_control.Pendulum method\), 3](#)  
[AnimatedPendulum \(class in PID\\_control\), 1](#)

## D

[derivative\\_output\(\) \(PID\\_control.PIDControl method\), 2](#)

## G

[get\\_func\\_values\(\) \(PID\\_control.Pendulum method\), 3](#)  
[get\\_support\\_values\(\) \(PID\\_control.Pendulum method\), 3](#)

## I

[integral\\_output\(\) \(PID\\_control.PIDControl method\), 2](#)

## P

[Pendulum \(class in PID\\_control\), 3](#)  
[PID\\_control \(module\), 1](#)  
[PIDControl \(class in PID\\_control\), 1](#)  
[plot\(\) \(PID\\_control.Pendulum method\), 4](#)  
[proportional\\_output\(\) \(PID\\_control.PIDControl method\), 2](#)

## S

[solve\(\) \(PID\\_control.Pendulum method\), 4](#)  
[solve\\_from\\_angles\(\) \(PID\\_control.Pendulum method\), 4](#)

## T

[total\\_output\(\) \(PID\\_control.PIDControl method\), 2](#)