Handling System Limits
PID Control with Saturation

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems
The Controller Output

- The output of the PID is somewhat proportional to the error (in a direct, integral or derivative “way”)

- If the error is large, the controller output may be very large (and also increasing in the presence of the integrator)

- But, in real life, can we provide a “driving signal” to a system that is as large as we want?

- Are systems subject to certain limits that cannot be overcome?
In the “cart example”, the force is due to the power of the motors that, in turn, is generated according to the voltage applied to motors themselves. Increasing the voltage, increases motor power and thus the pushing force. But can we increase such a voltage indefinitely? NO! There are two kind of limits:

1. The electronics driving the motor cannot provide a voltage greater than the power supply.
2. Supposing that the former limit does not occur, if we overcome the limits for what the motors are designed, we easily burn them!
Handling Limits

- In other words, we need to saturate the controller output according to a certain limit $OUT_{max}$.

- This objective is achieved by including a saturation block that ensure the output is always in the range $[-OUT_{max}, OUT_{max}]$. 

The PID Controller with Saturation

Physical System

PID Controller

Saturator

Target

$K_P$

$K_I$ ∫

$K_D \frac{d}{dt}$

$OUT_{max}$
From the implementation point of view, a saturation block is simply a couple of "ifs"

```python
... if output > OUT_MAX:
    output = OUT_MAX
elif output < -OUT_MAX:
    output = -OUT_MAX
# else the output is unchanged
... ```
Let us consider that in our Cart, the motors are not able to provide a push greater than $0.5 \, N$.

Let’s see the implementation of the position control with saturation.

```python
... self.controller = PIDSat(0.2, 0, 0, 0.5) # Kp = 0.2, saturation 0.5 Newton ...

class PIDSat:
    def __init__(self, kp, ki, kd, saturation):
        ...
        self.saturation = saturation

    def evaluate(self, delta_t, target, current):
        ...
        if output > self.saturation:
            output = self.saturation
        elif output < -self.saturation:
            output = -self.saturation
        return output
```
Cart Position Control with Saturation

Without saturation

With saturation

- Target
- Current Position
- Controller Output

Corrado Santoro

PID Control with Saturation
Also in the case of speed control we must consider the presence of system limits and thus saturation.

Let’s consider the cart with 0.5 N of maximum push.

Let’s test the speed control algorithm using the same parameters of the case without saturation.

```python
self.controller = PIDSat(3.0, 2.0, 0.0, 0.5)  # Kp = 3, Ki = 2, Sat = 0.5 N
self.target_speed = 1.5  # 1.5 m/s
```

...
The system is constantly in saturation and there is no way to achieve the target speed of 1.5 m/s
Let’s change our motors with more powerfull ones that are able to provide up to 2 $N$

Without saturation
$K_P = 3, K_I = 2$

With saturation (2 $N$)
$K_P = 3, K_I = 2$

It works!! But...an overshot appeared!!! Why?
Speed Control with Saturation

With saturation (2 \( N \))

\[ K_p = 3, \; K_I = 2 \]

The Overshot...

- Even if we used the same parameters, in the presence of saturation the overall system is different, so a different behaviour is expected.
- Above all, the saturator is a non-linear block.
- Indeed, the overshot is due to the integrator that accumulates the error.
Speed Control with Saturation

The Anti-Wind-up Optimisation

- Accumulating the error is necessary to obtain an adequate long-term output able to let the system reach the target.

- But, when we are in the “saturation area”, does it make sense to accumulate the error in any case?

- After all, since we have reached the system limits, increasing the accumulated value (above the system limits) does not help in any way.

- Worstly, if the accumulated value is too high (and the target is overcome) we must wait more time for its reduction (and this is the overshoot!)
The Anti-Wind-up Optimisation

So, let’s check when we are in the saturation area and, if this is the case, do not call the integrator (see standard.py, class PIDSat)

def evaluate(self, delta_t, target, current):
    error = target - current
    derivative = (error - self.prev_error) / delta_t
    self.prev_error = error

    if not(self.in_saturation):
        self.i.evaluate(delta_t, target, current)

    output = self.p.evaluate(target, current) + self.i.output + derivative * self.kd

    if output > self.saturation:
        output = self.saturation
        self.in_saturation = True
    elif output < -self.saturation:
        output = -self.saturation
        self.in_saturation = True
    else:
        self.in_saturation = False
    return output
Cart Speed Control with Saturation

Without saturation

$K_P = 3, K_I = 2$

With saturation (2N) and Anti-Wind-up

$K_P = 3, K_I = 2$

Corrado Santoro

PID Control with Saturation
Handling System Limits
PID Control with Saturation

Corrado Santoro

ARSLAB - Autonomous and Robotic Systems Laboratory
Dipartimento di Matematica e Informatica - Università di Catania, Italy
santoro@dmi.unict.it

Robotic Systems