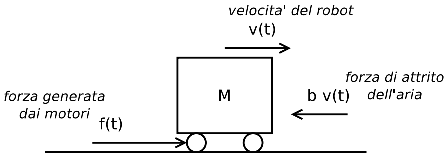
# Principles of System Control


Corrado Santoro

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


Robotic Systems

# Modelling the Cart



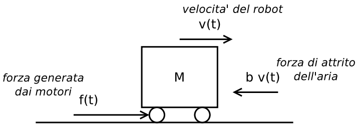

Let's start (once again) from the model based on differential equations:

$$\begin{cases} \dot{v} & = & -\frac{b}{M} v + \frac{1}{M} f \\ \dot{p} & = & v \end{cases}$$

## Controlling the Cart: Questions

1. Given a certain speed $\overline{v}$, what is the force $f$ that we must apply to let the cart travelling at the speed $\overline{v}$?
2. Given a certain position $\overline{p}$, at what time instant we must **stop** the cart in order to let it stop at $\overline{p}$?

# Controlling the Cart




## The Analytical Way

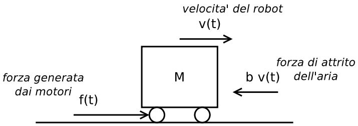
$$\begin{cases} \dot{v} = -\frac{b}{M}v + \frac{1}{M}f \\ \dot{p} = v \end{cases}$$

If we consider the use of a *constant* force $F$ and the cart not moving at $t = 0$, i.e. $v(0) = 0$, we can solve the equations analytically:

$$v(t) = \frac{F}{b}(1 - e^{-\frac{b}{M}t})$$

$$p(t) = \frac{F}{b}(1 - e^{-\frac{b}{M}t})t$$

# Controlling the Cart

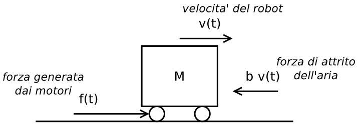



## The Algorithmical Way

- Given a certain speed $\overline{v}$, what is the force $f$ that we must apply to let the cart travelling at the speed $\overline{v}$?

1. **Measure** the current speed $v$
2. **Compute the error** with respect to target speed $error = \overline{v} - v$
3. Given the error use a **proper function** $F = control(error)$ that is able to **reduce and cancel the error**
4. **Apply** $F$
5. Go to step 1

# Controlling the Cart



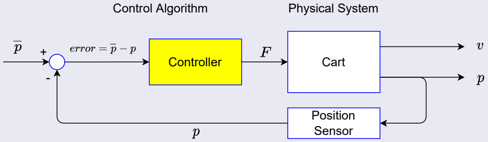

## The Algorithmical Way

- Given a certain position $\overline{p}$, at what time instant we must **stop** the cart in order to let it stop at $\overline{p}$?

1. **Measure** the current position $p$
2. **Compute the error** with respect to target position $error = \overline{p} - p$
3. Given the error use a **proper function** $F = control(error)$ that is able to **anticipate the cart inertia** (and thus reduce and cancel the error)
4. **Apply** $F$
5. Go to step 1

## The Control System Model: Feedback

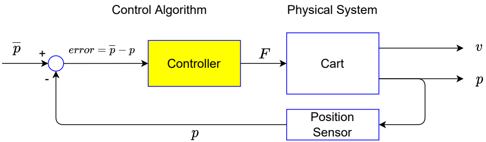
- The algorithms above can be represesed as the following *data-flow diagram*:




- This is the typical scheme to control dynamic systems and is called **feedback**
- The advantage is that the exact model of the system **is not needed** but only **its behaviour, in a qualitative way**
- The problem here is instead in the **control block** that must be properly designed

# Position Control

# Controlling the Cart




## Position Control

- We can make the following "generic" assumptions:
  1. If we are *far* from the target position (*error* is large), we can apply a large $F$
  2. As soon as we *approach* the target, it's better to **reduce** $F$ accordingly, thus anticipating the behaviour of the system and stop the cart in the target position
- In other words, we can try to control the system by applying a $F$ that is **directly proportional** to the *error*:

$$F = K_P \, error$$

with $K_P$ a constant determined in a sperimental way

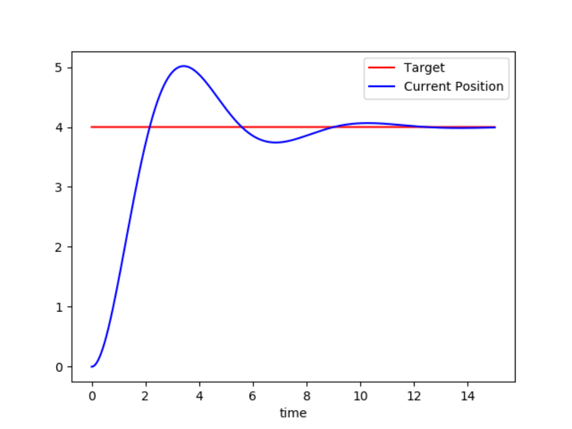
# Controlling Cart Position

examples/simple_control/cart_position_control.ipynb

# Controlling Cart Position

## Effect of $K_P$

$K_P = 1.0$

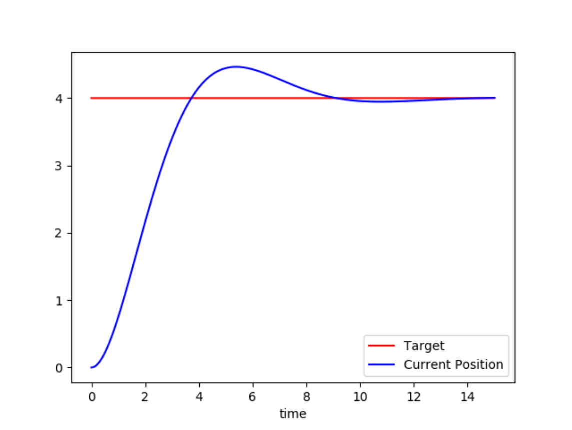



Too much!!! The cart overcomes the target and go back

# Controlling Cart Position

## Effect of $K_P$

$K_P = 0.5$

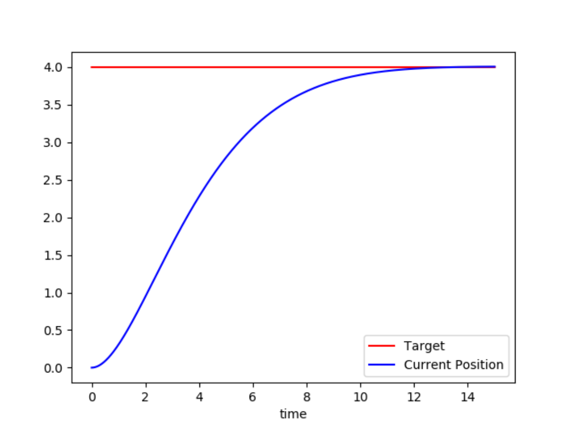



Still too much!!! The cart overcomes the target and go back

# Controlling Cart Position

## Effect of $K_P$

$K_P = 0.2$




Good enough!!!

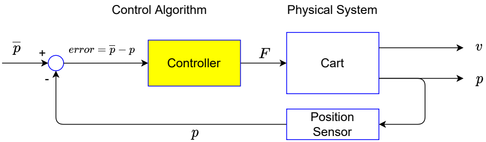
# Controlling Cart Position

## Effect of $K_P$

- In a **Proportional Controller**, $K_P$ controls the "speed" (*dyamics*) of the system
- If $K_P$ is small, the system reaches "slowly" the target
- If $K_P$ is large, the system is "fast" to reach the target but if it is "too much", the target is overcome and the system oscillates
- therefore...
- for each system to be controlled, there is a $K_P$ limit $L$; if $K_P > L$, the system oscillates
- we cannot have a system "fast" and "not oscillating", but always a compromise between these two aspect

# Controlling the Ball

# Controlling the Ball




## Controlling the (Godot) Ball

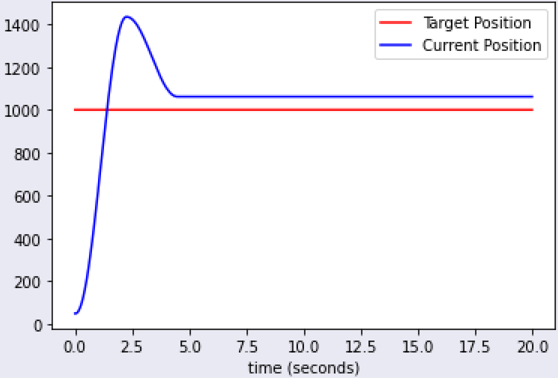
- We use the same algorithm to ensure that the ball reaches a certain position
- We consider a target position of 1000 *pix*

see examples/simple_control/godot_ball_position_control.ipynb

# Controlling Ball Position

## Effect of $K_P$

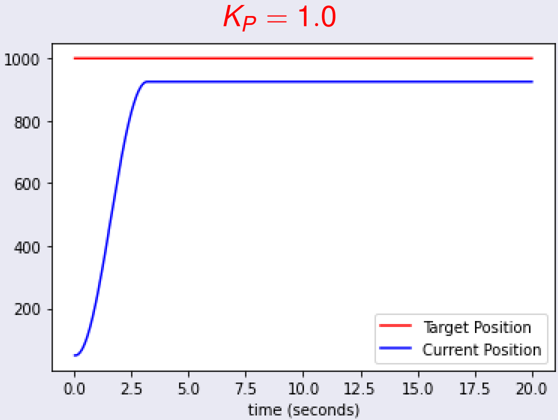
$K_P = 2.0$




The target is overcome and the ball does not go back!!!

# Controlling Ball Position

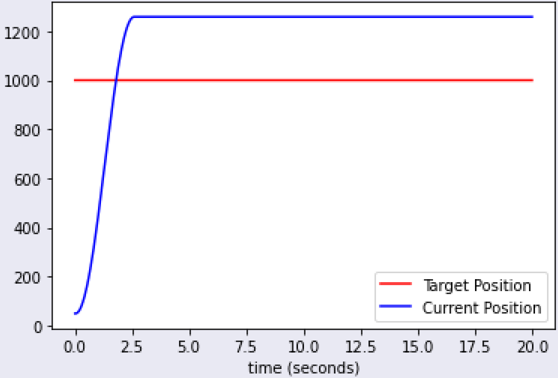
## Effect of $K_P$




The target is never reached

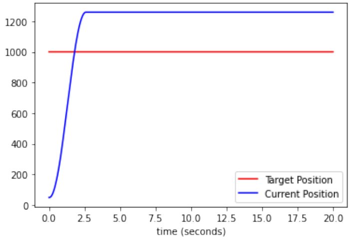
## Effect of $K_P$

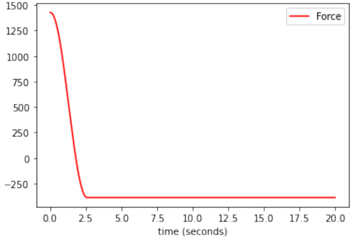
$K_P = 1.5$




The target is never reached also in this case...Why???

# Real Systems vs Modeled Systems

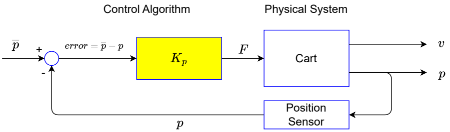



## Position Control

- We observe that, at a certain time instant, the force is $\neq 0$ (because there is still an error), but the ball does not move
- This is due to the fact that the force is not enough to overcome static frictions
- Indeed this is what happens in **real systems**
- But not in the cart modeled, since we did not consider static friction forces
- **What shall we do?**

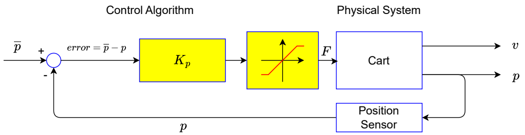
# Real Systems vs Modeled Systems




## Infinite Force or Limited Force?

- Another aspect of the schema above is related to the output of the controller
- The use of $out = K_P(target - current)$ implies that the output is as large as the error, but **can the *out* be any value** (also very large)?
- Indeed, considering that the out is the force that we want the motors to apply, it cannot be any value
- But, any motor (actuator) can provide a **maximum power** and thus a **maximum force**

# Real Systems vs Modeled Systems

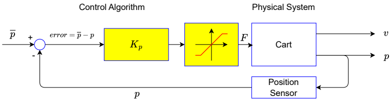


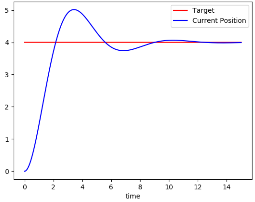

## Infinite Force or Limited Force?

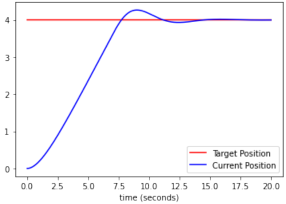
- Any motor (actuator) can provide a **maximum power** and thus a **maximum force**
- So we must include, in the control chain, a **saturator**, i.e. a block that limits the output of the controller in the interval $[-MAX, MAX]$, where $MAX$ is the limit of the system input
- The saturator is simply a couple of "if"s applied to the proportional output

(see examples/simple_control/cart_position_control_saturation.ipynb)

# Real Systems vs Modeled Systems



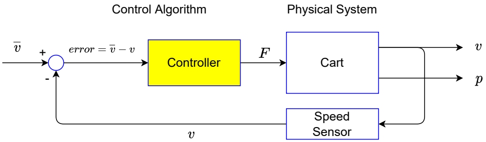




## Saturation

- Without saturation (left) vs. With saturation (right), $K_p = 1.0$, $MAX = 0.5\ N$
- We notice that with saturation the overall system takes more time to reach the target
- This is a natural consequence since reducing time implies to have more “power”

Controlling the speed of the cart

# Controlling the Speed

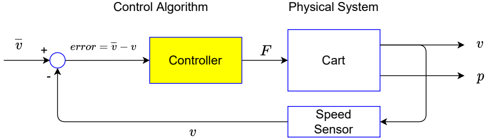



## Speed Control

$$F = K_p \; error$$

- Is the proportional controller enough for speed control ?
- Let's analyse the output of the controller w.r.t. the trend of the error
- When $error \neq 0$ we must "push" the cart and thus generate a $F \neq 0$
- But what happens when the **target speed** is **reached**?
- In this case, $error = 0$ thus, according to the formula above, $F = 0$, **the cart stops!!!!!!**

# Controlling the Speed

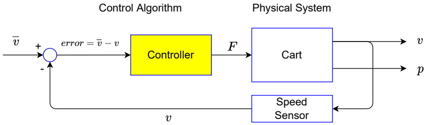



## Thinking analytically ...

- Let's assume that the cart is moving (thanks to a certain $F$) and that, at a certain point, the target speed $\overline{v}$ is reached
- We have $error = 0$, meaning that our force is "good enough" to push the cart at the target speed
- But to maintain that speed we should **not change** $F$
- In other words, when $error = 0$, the $F$ must be **constant**!!
- If we think to the "basic systems" (proprtional, integrator, derivator), that condition is met by an **integrator**:

$$F(t) = \int_0^t error(\tau)\, d\tau$$

# Controlling the Speed



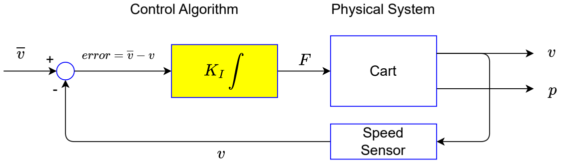

## Thinking practically ...

- When the *error* $> 0$ is large, we must **largely increase the** $F$ in order to gain speed
- When the *error* $> 0$ is small, we must **increase the** $F$ **of a small amount** in order to not overcome the target speed
- If *error* $< 0$ the target speed has been overcome, and thus we must **reduce** $F$
- If *error* $= 0$, we **must not change** the $F$
- In other words, $F$ must be a **weighted accumulator** of *error*:

$$F(k+1) = F(k) + const \cdot error(k)$$

- Once again, this is an **integrator**

# Controlling the Speed



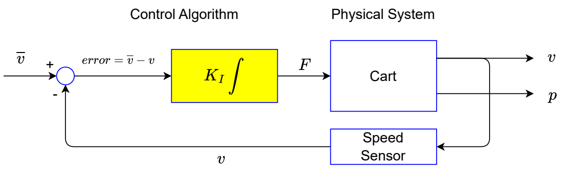

## Speed Control - The Integral Controller

$$F(t) = K_I \int_0^t error(\tau) d\tau$$

- We can use an integrator including a constant $K_I$ that is able to **weight** the contribution of the integral

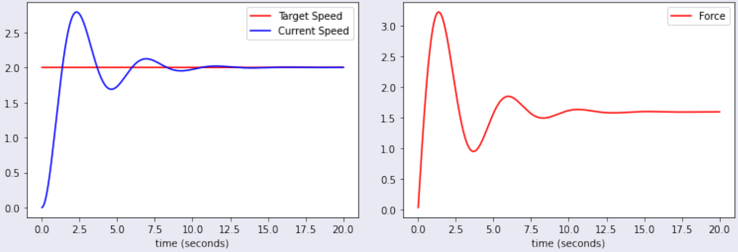
(see examples/simple_control/cart_speed_control.ipynb)

# Controlling Cart Speed



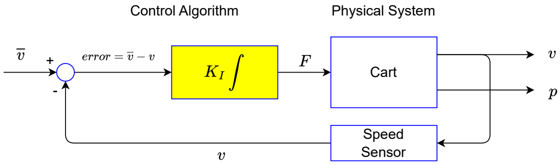

## Effect of $K_I$

$K_I = 2.0$



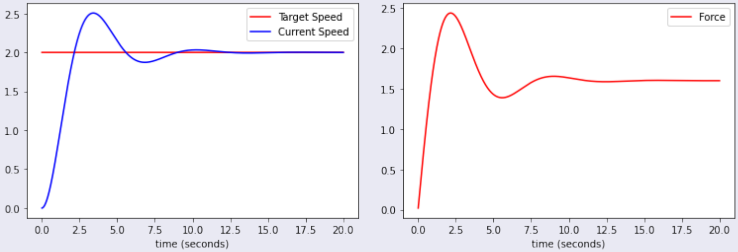

Too much!!! The cart overcomes the target and decelerates

# Controlling Cart Speed



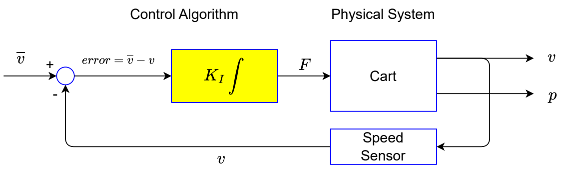

## Effect of $K_I$

$K_I = 1.0$



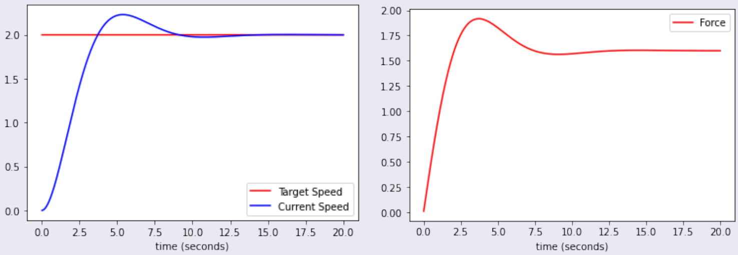

Still too much!!!

# Controlling Cart Speed



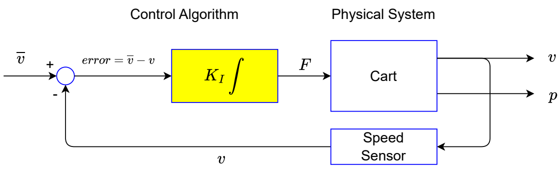

## Effect of $K_I$

$K_I = 0.5$



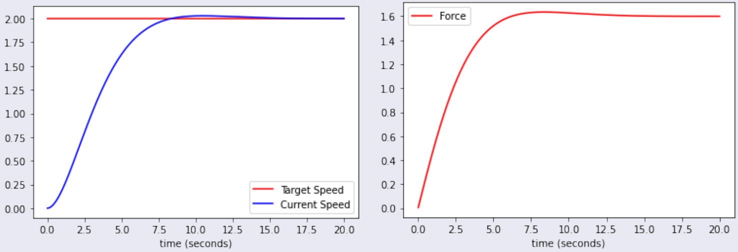

Quite good (but not so enough)

# Controlling Cart Speed




## Effect of $K_I$

$K_I = 0.25$



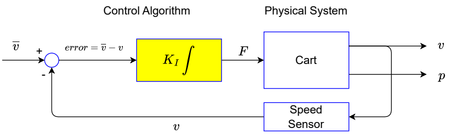

Good enough!!

## Controlling the speed of a rotating ball

See:

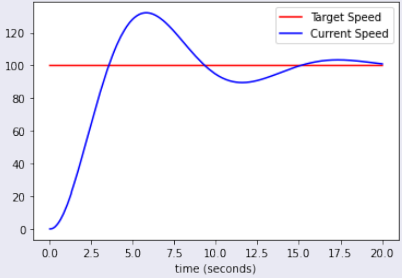
- godot/rolling_ball
- examples/simple_control/godot_ball_speed_control.ipynb

# Controlling Ball Speed



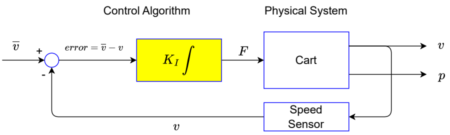

## Effect of $K_I$

$K_I = 0.5$



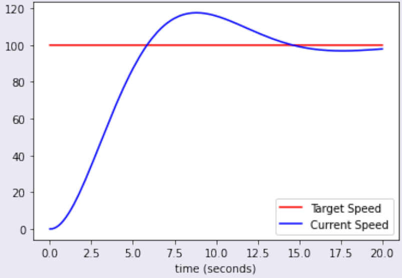

mmmmm....

# Controlling Ball Speed



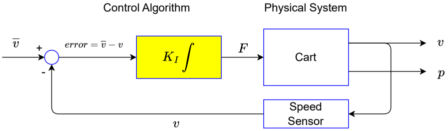

## Effect of $K_I$

$K_I = 0.25$




mmmmm....

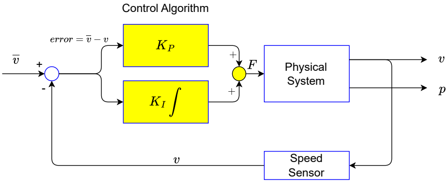
# The Integral Controller




## Effect of $K_I$

- The integrator is an **accumulator** of the error
- The constant $K_I$ controls the **rate** of accumulation
- If $K_I$ is **high**, the output of the controller **increases fastly**: this is good when the error is high, but bad when the error becomes small (too much accumulation)
- If $K_I$ is **low**, the output of the controller **increases slowly**: this is bad when the error is low, but good when the error becomes small

# The Proportional-Integral Controller

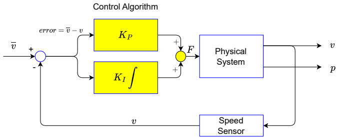



## PI Control

- **We can combine the effects of both P and I controllers**
- The P controller reacts immediatelly, but does not have memory
- It can be used when the error is **large** in order to speed-up the control
- The I controller reacts in the long term, it has memory
- It can be used when the P controller has no more effect (error is **small**), given that it has accumulated sufficient control action
- Let's see the effect....

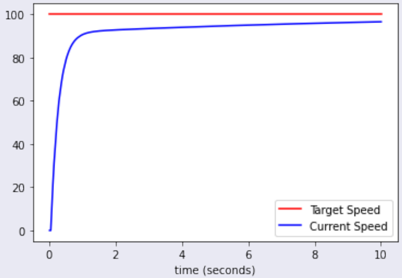
(see examples/simple_control/godot_ball_speed_control_PI.ipynb)

# The Proportional-Integral Controller



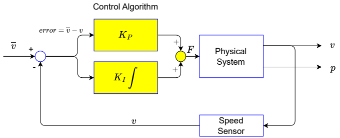

## PI Control

$K_P = 5, K_I = 0.5$



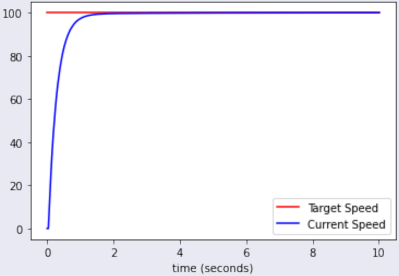

In the initial part the response is "fast", in the long term is "slow", let's increase $K_I$

# The Proportional-Integral Controller



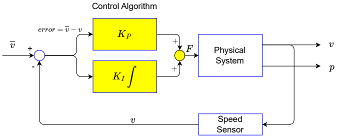

## PI Control

$K_P = 5, K_I = 2$



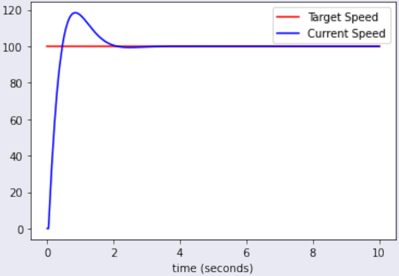

In the initial part the response is "fast", in the long term is "good", let's see if we can have a better behaviour...

# The Proportional-Integral Controller



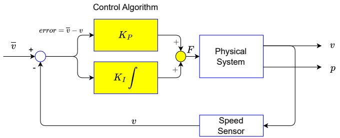

## PI Control

$K_P = 5, K_I = 10$



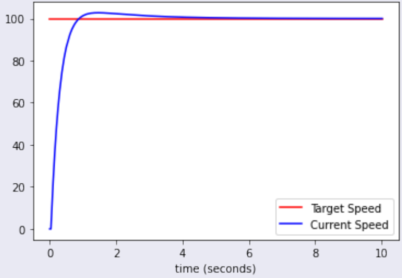

An overshot appears... too much $K_I$

# The Proportional-Integral Controller



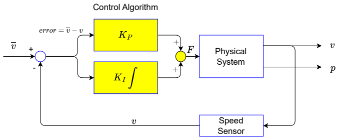

## PI Control

$K_P = 5, K_I = 3$



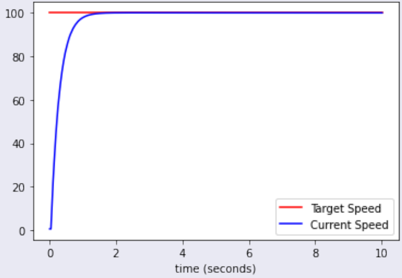

Still the overshot...

# The Proportional-Integral Controller



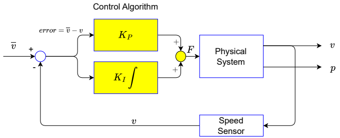

## PI Control

$K_P = 5, K_I = 2.1$



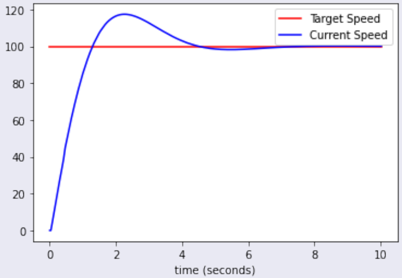

**Good!**

# The Proportional-Integral Controller



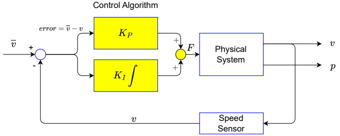

## PI Control

$K_P = 1.5, K_I = 2.1$



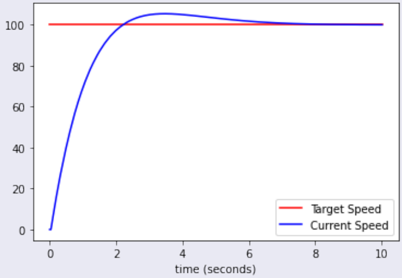

$K_P$ is small, so the system is slower than previous, and here the contribution of $K_I$ is too much

# The Proportional-Integral Controller



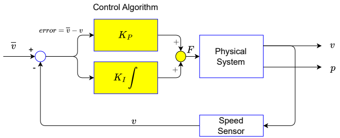

## PI Control

$K_P = 1.5, K_I = 1$



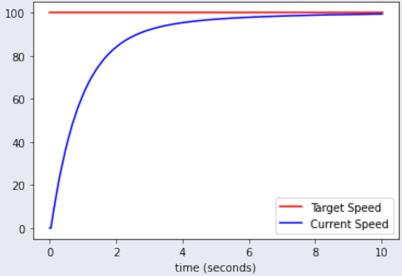

The system slower but still too much $K_I$

# The Proportional-Integral Controller



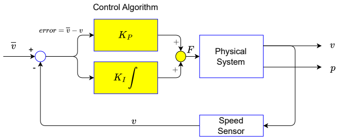

## PI Control

$K_P = 1.5, K_I = 0.5$



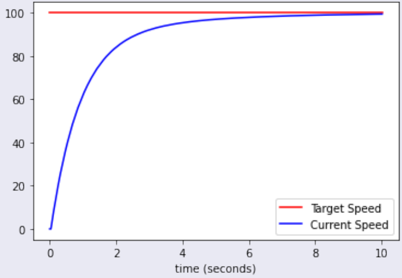

Too slow!!!

# The Proportional-Integral Controller



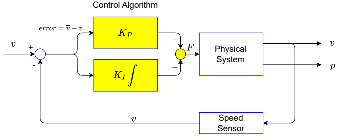

## PI Control

$K_P = 1.5, K_I = 0.5$



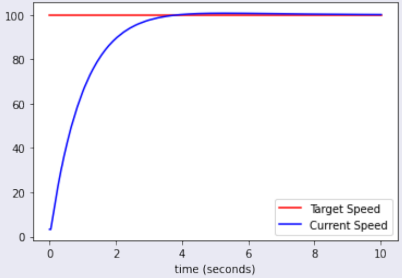

Too slow!!!

# The Proportional-Integral Controller



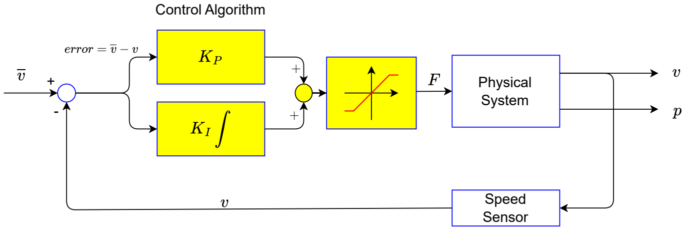

## PI Control

$K_P = 1.5, K_I = 0.7$



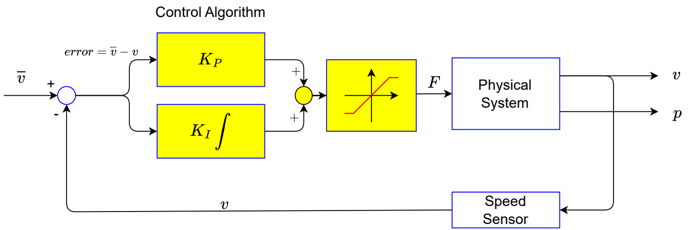

Good enough!!

# The Role of Saturation




## PI Control + Saturation

- Also in PI controllers a saturation block is worth, since the system cannot overcome certain limits and the controller output must be limited accordingly
- But the use of saturator with and integrator has some side effects that must be considered



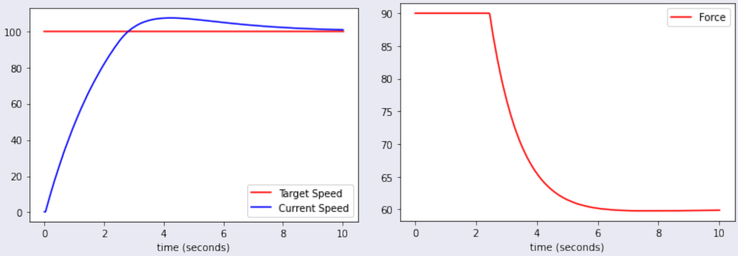

## PI Control + Saturation

- Also in PI controllers a saturation block is worth, since the system cannot overcome certain limits and the controller output must be limited accordingly
- But the use of saturator with and integrator has some side effects that must be considered
- Let us consider the last set-up but with a saturation of 90 $N$

# The Role of Saturation

## PI Control + Saturation

$K_P = 1.5, K_I = 0.7, SAT = 90\ N$

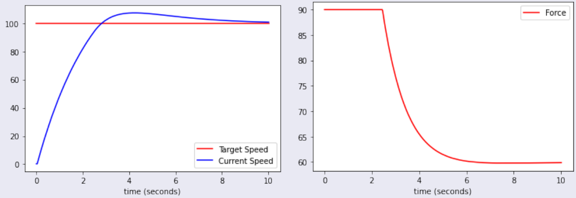



- The role of saturator is clear
- Saturation appears in the first part (indeed the error is high so the output is high)
- An overshot appears, why??

# The Anti Wind-up Optimisation

## PI Control + Saturation

$K_P = 1.5, K_I = 0.7, SAT = 90\ N$



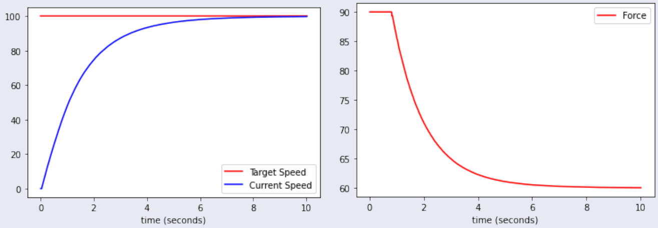

- The overshot is due to the fact that, in the first part, the controller tries to "push the system towards the target", but there is a limit thus the system **cannot perform as desired**
- The error **does not decrease as expected**
- It is worth to accumulate the error, given that there is no way to a have more performances???
- The Anti Wind-up optimisation, checks if the output is saturated and, in this case, **avoids integrating the error** until we exit from the saturation phase

# The Anti Wind-up Optimisation

## PI Control + Saturation

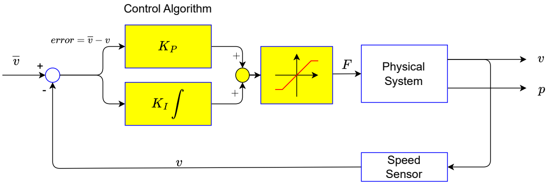
(see examples/simple_control/godot_ball_speed_PI_sat_aw_control.ipynb)

$K_P = 1.5, K_I = 0.7, SAT = 90\ N$, **Anti wind-up**




- The system is in saturation for less time than the previous case (this is good!!)
- The overshot disappers
- Parameters can be tuned in order to have a better response (if needed)

# Summary




- The **feedback** is right way to "control a system", i.e. to make the system behave as desired
- A simple **proportional controller** can do the job but not in all cases
- If, when $error = 0$, we need a constant output $\neq 0$, an **integrator** must be added
- The actions of P and I controllers can be combined to have better response performances
- The P controller acts immediately (and thus works well in the first part)
- The I controller acts after (and thus works well in the long term)
- Saturation is always needed (any real system has limits)

# Principles of System Control


Corrado Santoro

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


Robotic Systems