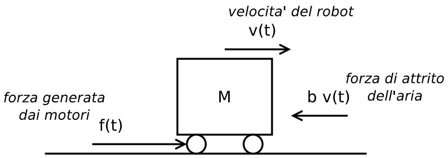
Controlling a Cart

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



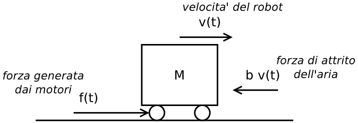
Let's start (once again) from the model based on differential equations:

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

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 (1)

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

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

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:

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

Controlling the Cart

The Analytical Way (2)

Since the speed is given as $\overline{v}$, we have:

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

If we invert the relation above to solve it, we must determine both F and t; in other words, the question should be changed as:

1. Given a certain speed $\overline{v}$, what is the force F that we must apply to let the cart travelling at the speed $\overline{v}$ **after a given time** $\overline{T}$?

$$F = \overline{v}\frac{b}{1 - e^{-\frac{b}{M}\overline{T}}}$$

Controlling the Cart

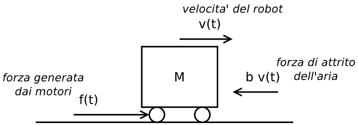
The Analytical Way (3)

$$\dot{v} = -\frac{b}{M} v + \frac{1}{M} F \qquad F = \overline{v} \frac{b}{1 - e^{-\frac{b}{M}\overline{T}}}$$

Yeah!! Problem solved?? Ehm...**NO!**:

- The differential equation is a model, so the mathematical relations are **approximation** of the real object
- For example, once we stop the cart by applying $f = 0$, according to the model the speed reaches 0 for $t \rightarrow +\infty$ (see the exponential factor), while, in real word, the speed reaches 0 in a **finite time**
- The problem depends on b and M, which can be estimated but not measured with precision (above all b), thus leading to a *high approssimation*
- Once the target speed $\overline{v}$ has been reached at time $\overline{T}$, we can guarantee the it will be maintained for $t > \overline{T}$?

Controlling the Cart

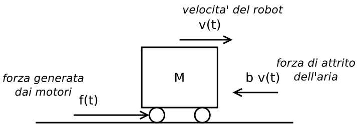


The Algorithmical Way

- 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



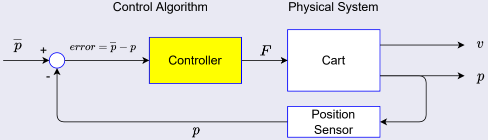
The Algorithmical Way

- 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

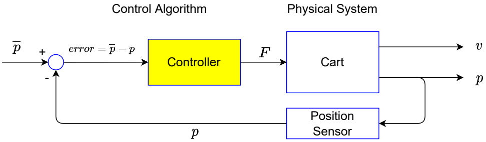
- The algorithms above can be represed as the following *data-flow diagram*:



- 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

Let's implement our controller
lib/controllers/standard.py

```
class Proportional:

    def __init__(self, kp):
        self.kp = kp

    def evaluate(self, target, current):
        error = target - current
        return self.kp * error
```

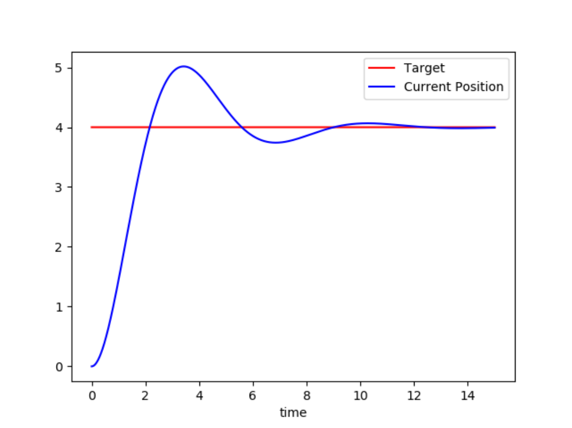
tests/test_position_control_cart_gui.py

```
from controllers.standard import *
...
class CartRobot(RoboticSystem):

    def __init__(self):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg, friction = 0.8
        self.cart = Cart(1, 0.8)
        self.controller = Proportional(0.2) # Kp = 0.2
        self.target_position = 4 # 4 meters

    def run(self):
        F = self.controller.evaluate(self.target_position, self.get_pose())
        self.cart.evaluate(self.delta_t, F)
        return True
```

Controlling Cart Position

Effect of K_P

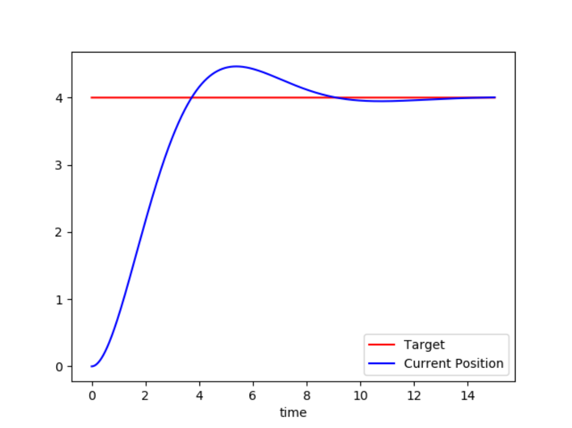
$K_P = 1.0$



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

Effect of K_P

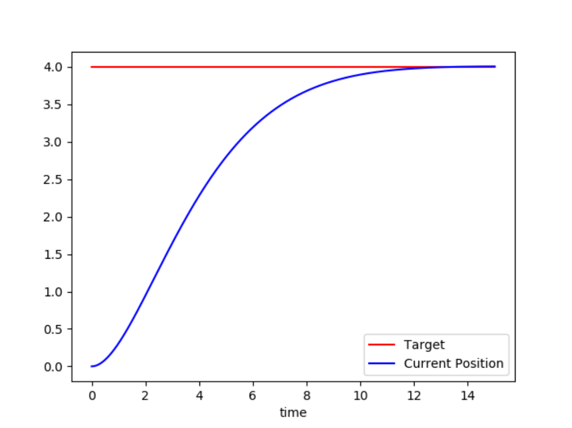
$K_P = 0.5$



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

Effect of K_P

$K_P = 0.2$



Good enough!!!

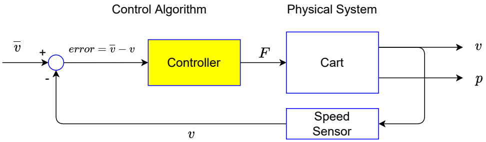
Controlling Cart Position

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

Speed Control

Controlling the Cart



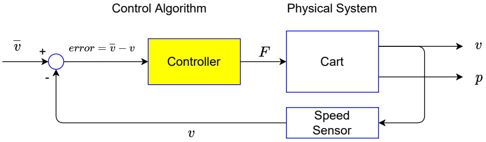
Speed Control

- We can think to the way in which we drive our car and use the accelerator pedal to reach and maintain a certain speed:
  1. We push the pedal, from "0" to a certain point, thus increasing F
  2. When we reach the target speed (i.e. the error is "0"), we keep the foot on the pedal in order to provide a (more-or-less) constant F able to maintain the target speed
- In other words, we can try to control the system by applying a F that **increases** as soon as the $error \neq 0$, using an *increasing factor* **directly proportional** to the *error*:

$$F = F + K_I \, error$$

with K_I a constant determined in a sperimental way

Controlling the Cart

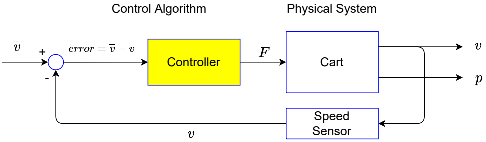


Speed Control - The Integral Controller

$$F = F + K_I \, error$$

- If $error > 0$, F increases at a rate determined by K_I
- If $error < 0$, F decreases at a rate determined by K_I
- If $error = 0$, F does not change
- Our control action (the F) "accumulates" the error for each iteration
- In other words, the F is somewhat proportional to the **integral** of the error

Controlling the Cart



Speed Control - The Integral Controller

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

by using the approximation $d\tau \simeq \Delta T$:

$$F(t) = K_I \sum_{i=0}^{\frac{t}{\Delta T}} error(i\Delta T)\Delta T$$

or, recursively:

$$F(t + \Delta T) = F(t) + K_I error(t) \Delta T$$

with $F(0) = 0$

Controlling Cart Speed

Let's implement our integral controller

lib/controllers/standard.py

```
class Integral:

    def __init__(self, ki):
        self.ki = ki
        self.output = 0

    def evaluate(self, delta_t, target, current):
        error = target - current
        self.output = self.output + self.ki * error * delta_t
        return self.output
```

Controlling Cart Speed

Let's implement our integral controller

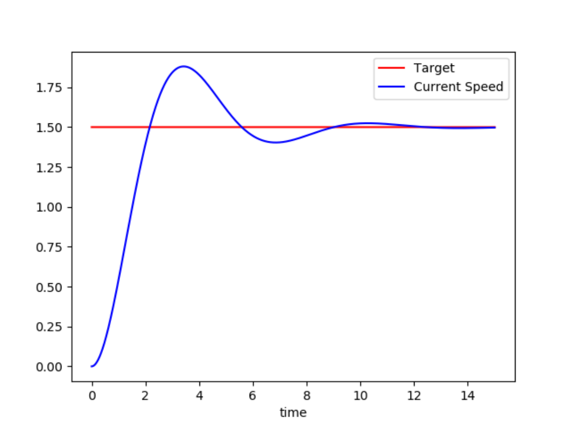
tests/test_speed_control_cart_gui_plot.py

```
from controllers.standard import *
...
class CartRobot(RoboticSystem):

    def __init__(self):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg, friction = 0.8
        self.cart = Cart(1, 0.8)
        self.controller = Integral(0.2) # Ki = 0.2
        self.target_speed = 1.5 # 1.5 m/s

    def run(self):
        F = self.controller.evaluate(self.delta_t,
                                     self.target_speed, self.get_speed())
        self.cart.evaluate(self.delta_t, F)
        ...
```

Controlling Cart Speed

Effect of K_I

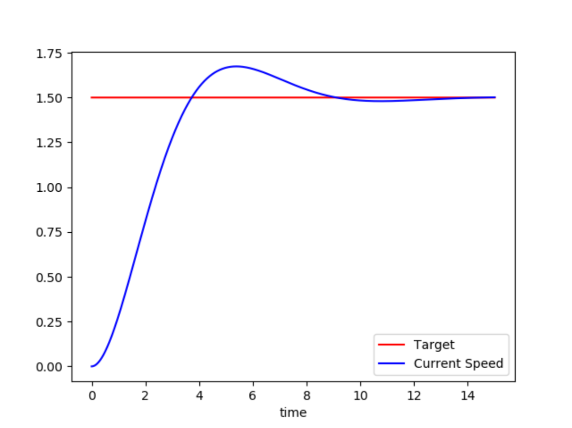
$K_I = 1.0$



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

Effect of K_I

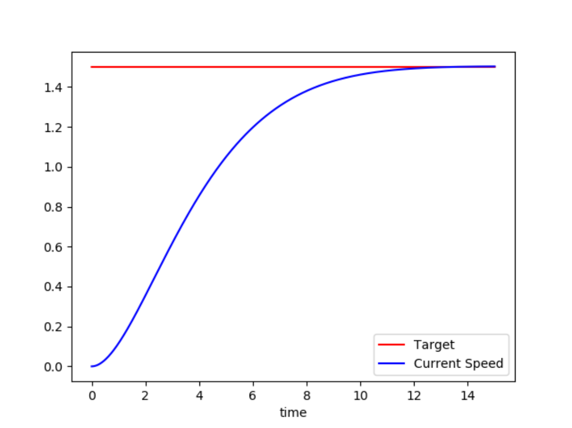
$K_I = 0.5$



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

Effect of K_I

$K_I = 0.2$



Good enough!!! ...but maybe too slow??

Speed Control - The Integral Controller

$$F = F + K_I \, \Delta T \; error$$

- Here the effect of the intergral controller is to “accumlate” the control action
- At the beginning, the accumulated value is low, so the control action is not so strong
- To make the control action “strong enough”, we must wait that the accumulated value becomes enough high, but this is done late in time
- Can we speed-up the control action in other ways rather than increasing K_I?
- Can we **combine** another control action able to respond “very fast”?

Recalling Actions

Proportional Action

$$F = K_P \; error$$

Responds **immediatelly**

Integral Action

$$F = F + K_I \; \Delta T \; error$$

Responds when the **accumulated action is enough**

Proportional-Integral Actions

$$\begin{aligned} INT &= INT + error \Delta T \\ F &= K_P \; error + K_I \; INT \end{aligned}$$

Let's combine **both actions** in order to gain the advantages of both

The “PI” Controller

lib/controllers/standard.py

```
...
class ProportionalIntegral:

    def __init__(self, kp, ki):
        self.p = Proportional(kp)
        self.i = Integral(ki)

    def evaluate(self, delta_t, target, current):
        return self.p.evaluate(target, current) + \
               self.i.evaluate(delta_t, target, current)

...
```

Controlling Cart Speed

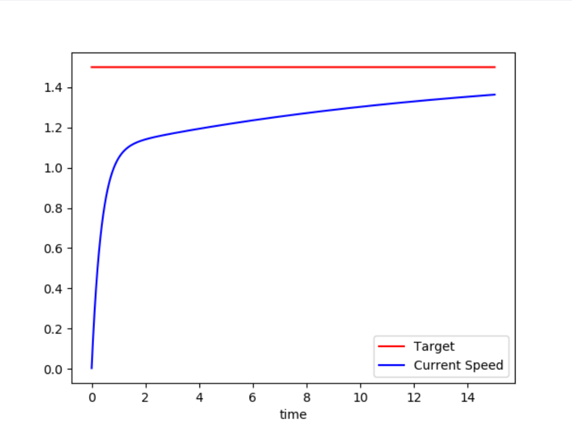
Let's implement our integral controller

tests/test_speed_pi_control_cart_gui_plot.py

```
from controllers.standard import *
...
class CartRobot(RoboticSystem):

    def __init__(self):
        super().__init__(1e-3) # delta_t = 1e-3
        # Mass = 1kg, friction = 0.8
        self.cart = Cart(1, 0.8)
        self.controller = ProportionalIntegral(3.0, 2.0)
                                        # kp = 3.0, ki = 2.0
        self.target_speed = 1.5 # 1.5 m/s

    def run(self):
        F = self.controller.evaluate(self.delta_t,
                                     self.target_speed, self.get_speed())
        self.cart.evaluate(self.delta_t, F)
        ...
```

Effect of K_P and K_I

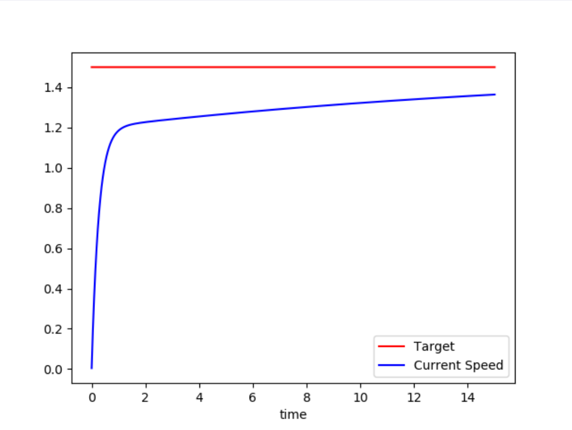
$K_P = 2.0, K_I = 0.2$



Controlling Cart Speed

Effect of K_P and K_I

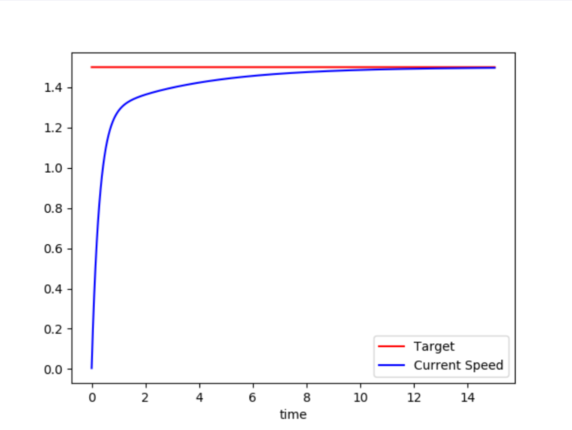
$K_P = 3.0, K_I = 0.2$



Controlling Cart Speed

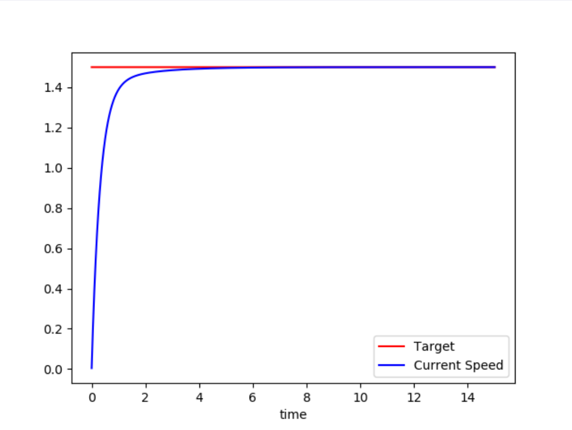
Effect of K_P and K_I

$K_P = 3.0, K_I = 1.0$



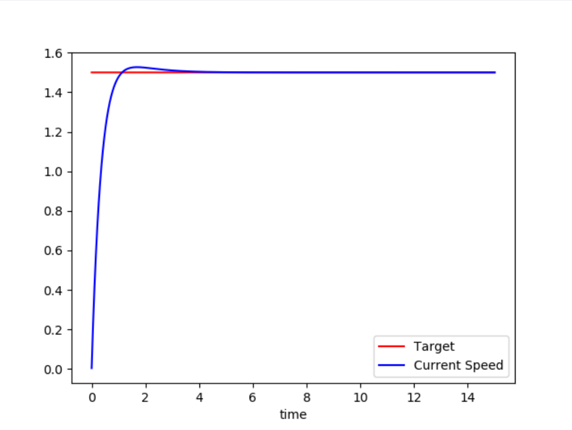
Effect of K_P and K_I

$K_P = 3.0, K_I = 2.0$



Effect of K_P and K_I

$K_P = 3.0, K_I = 3.0$



Controlling a Cart

Corrado Santoro

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

Robotic Systems