ETH zürich
Chair of Software Engineering

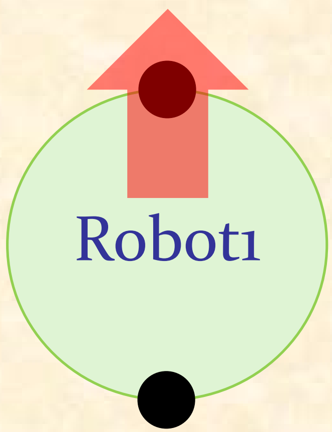
Robotics Programming Laboratory

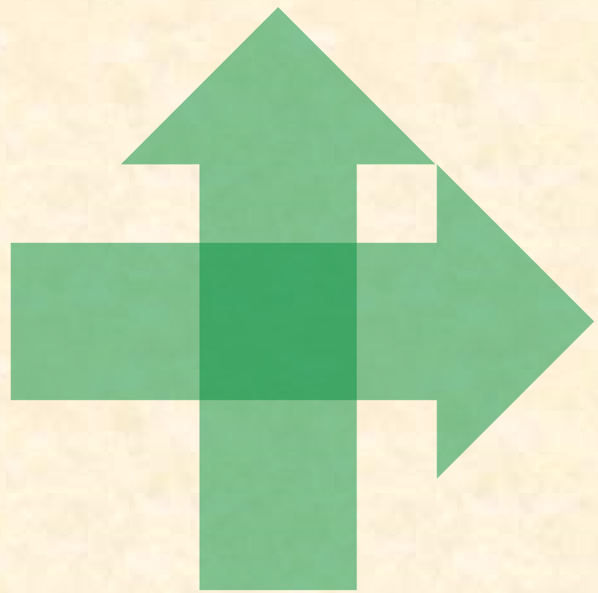
Bertrand Meyer
Jiwon Shin

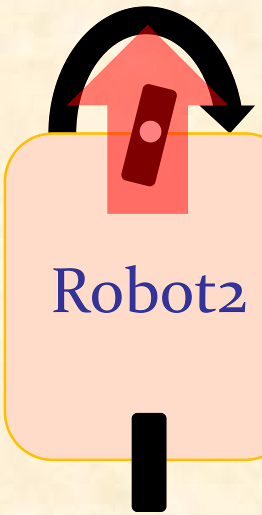
Lecture 3:

Control

Go forward, go right







Holonomic

DDOF=DOF

Nonholonomic

DDOF<DOF

DOF: Ability to achieve various poses

DDOF: Ability to achieve various velocities

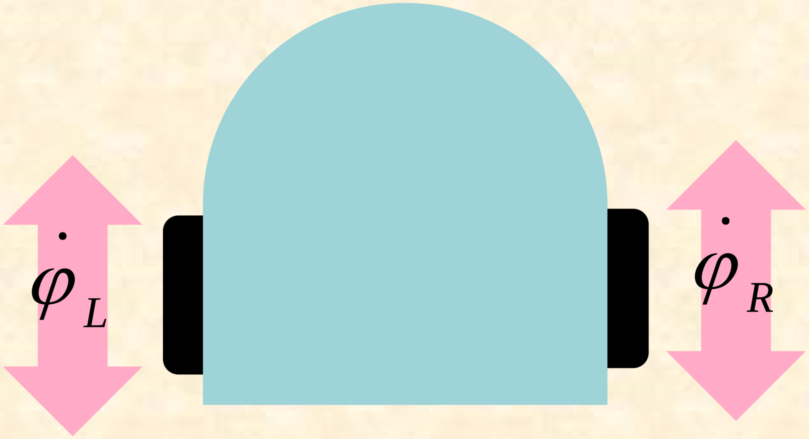
Differential drive

Forward: $\dot{\varphi}_L = \dot{\varphi}_R > 0$

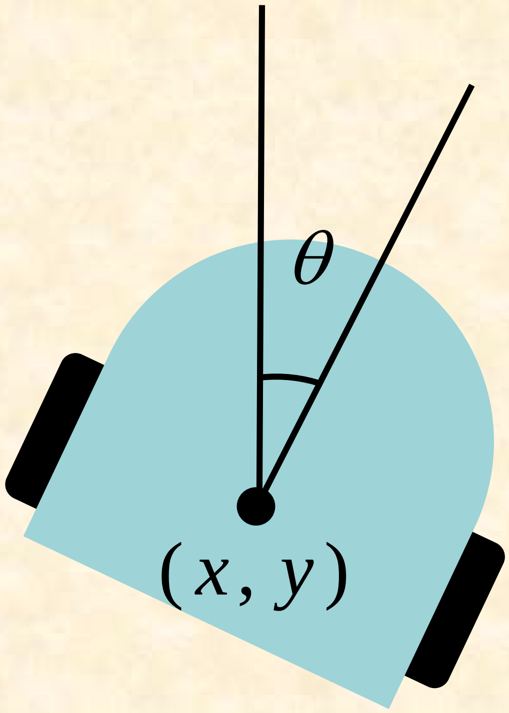
Backward: $\dot{\varphi}_L = \dot{\varphi}_R < 0$

Right turn: $\dot{\varphi}_L > \dot{\varphi}_R$

Left turn: $\dot{\varphi}_L < \dot{\varphi}_R$



Differential drive



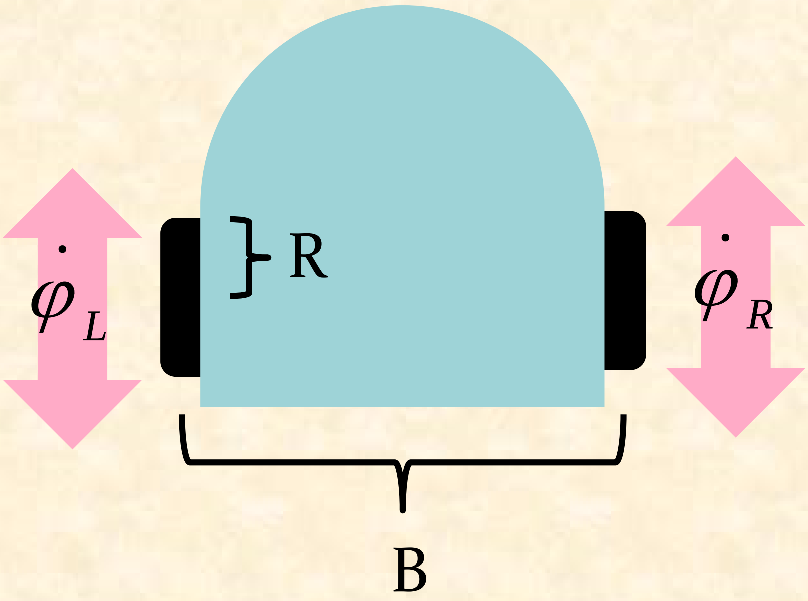
Input: (v, ω)

$$\dot{x} = v \cos \theta$$

$$\dot{y} = v \sin \theta$$

$$\dot{\theta} = \omega$$

Differential drive



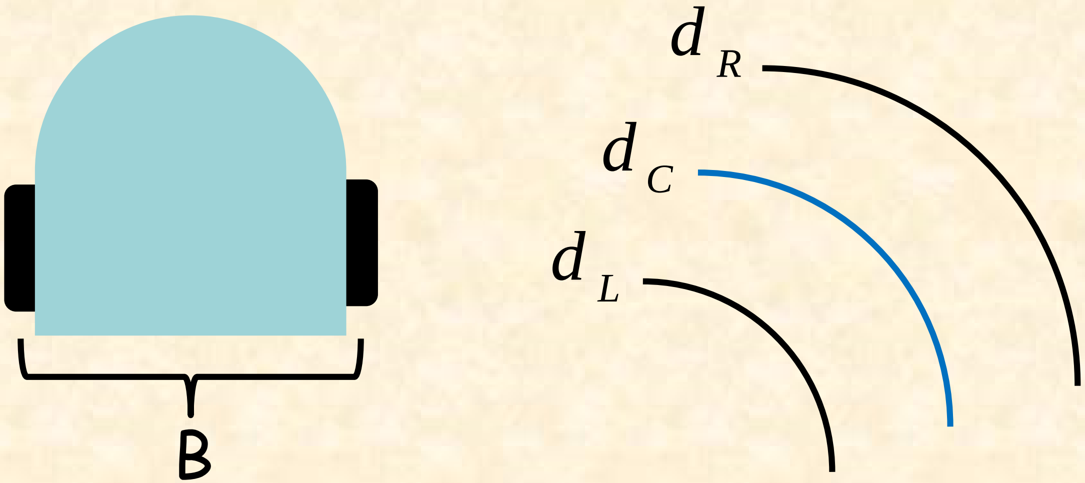
$$\dot{x} = R\frac{(\dot{\varphi}_L + \dot{\varphi}_R)}{2}\cos\theta$$

$$\dot{y} = R\frac{(\dot{\varphi}_L + \dot{\varphi}_R)}{2}\sin\theta$$

$$\dot{\theta} = \frac{R}{B}(\dot{\varphi}_R - \dot{\varphi}_L)$$

Odometry: intuition

Odometry for small t



$$d_C = \tfrac{1}{2}(d_L + d_R)$$

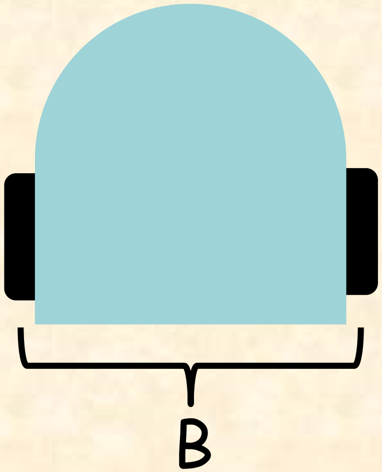
$$\theta_C = \frac{d_R - d_L}{B}$$

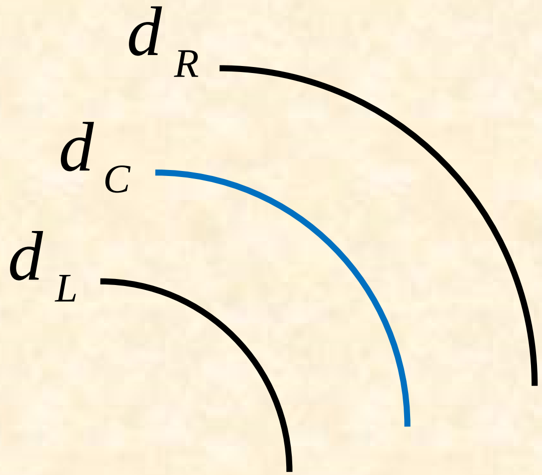
$$x(t) = x(t-1) + d_C \cos\theta(t)$$

$$y(t) = y(t-1) + d_C \sin\theta(t)$$

$$\theta(t) = \theta(t-1) + \theta_C$$

More accurate odometry for small t





$$d_C = \tfrac{1}{2}(d_L + d_R)$$

$$\theta_C = \arctan\left(\frac{d_R - d_L}{B}\right)$$

$$x(t) = x(t-1) + d_C \cos(\theta(t-1) + \tfrac{1}{2}\theta_C)$$

$$y(t) = y(t-1) + d_C \sin(\theta(t-1) + \tfrac{1}{2}\theta_C)$$

$$\theta(t) = \theta(t-1) + \theta_C$$

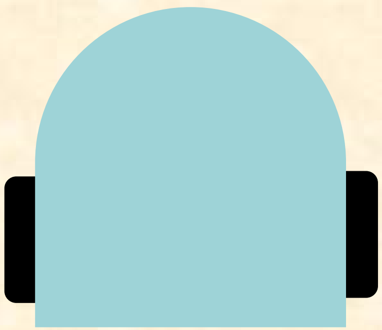
Wheel encoder

How do we get the distance each wheel has moved?

- If the wheel has N ticks per revolution:

$$\triangle n_{tick} = n_{tick}(t) - n_{tick}(t-1)$$

$$d = 2\pi R \frac{\triangle n_{tick}}{N}$$

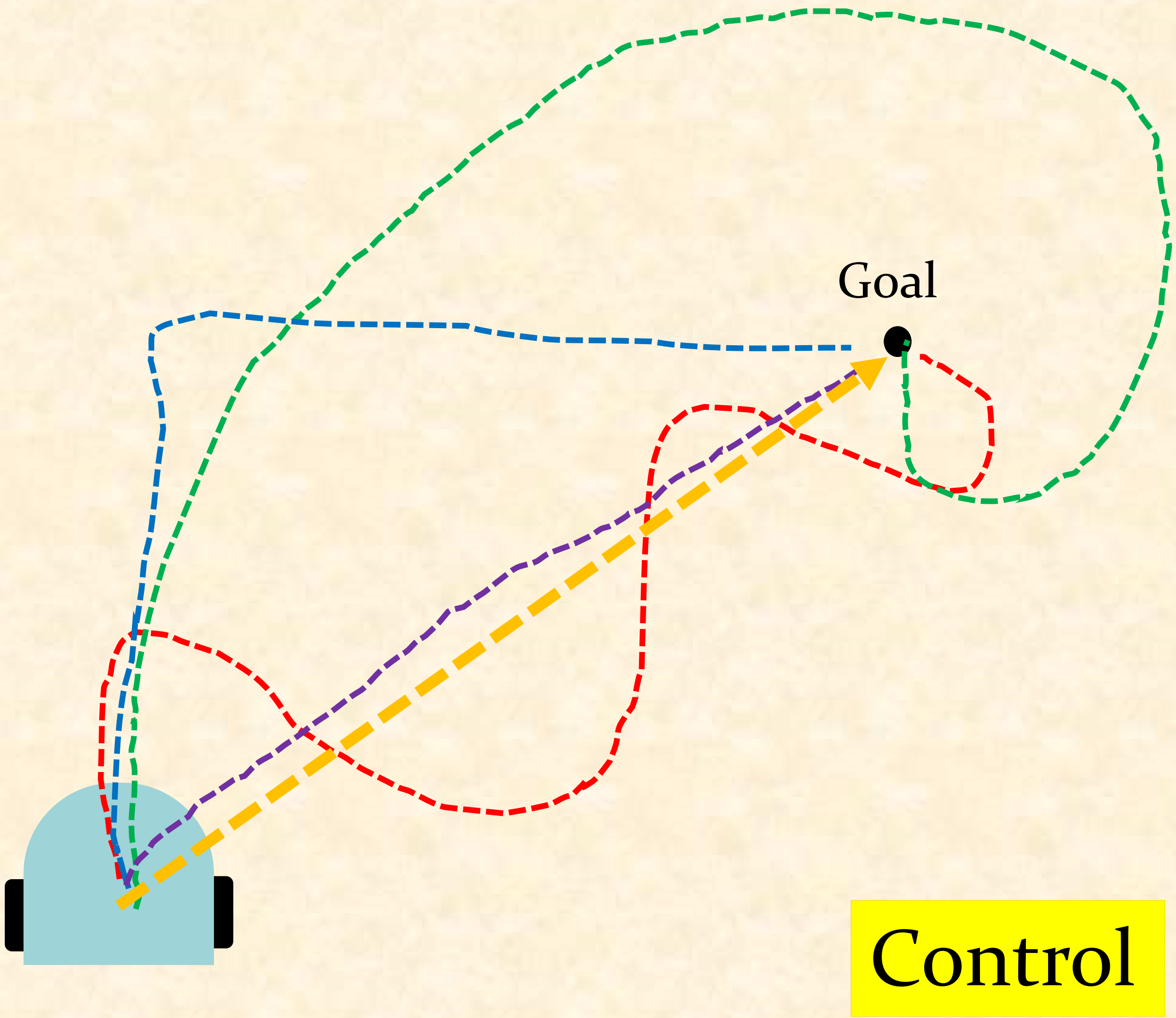


- Thymio:

$$d = \dot{d} \triangle t$$

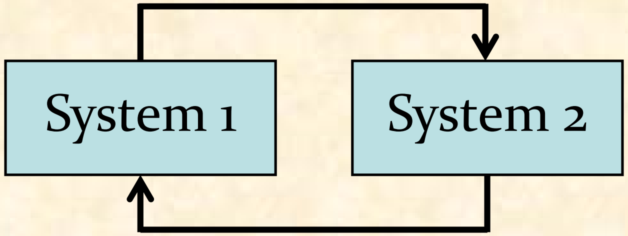
Drift

Go to goal

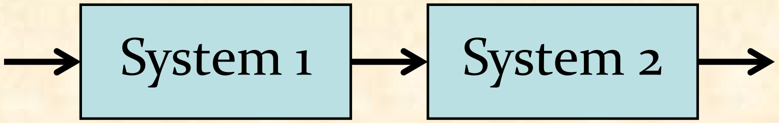


Feedback

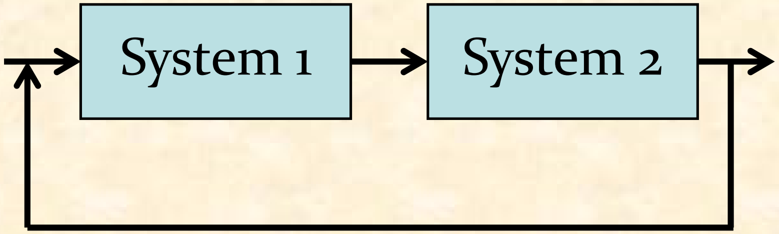
A collection of two or more dynamical systems, in which each system influences the other, resulting in strongly-coupled dynamics



- Open loop: the systems are not interconnected (no feedback)

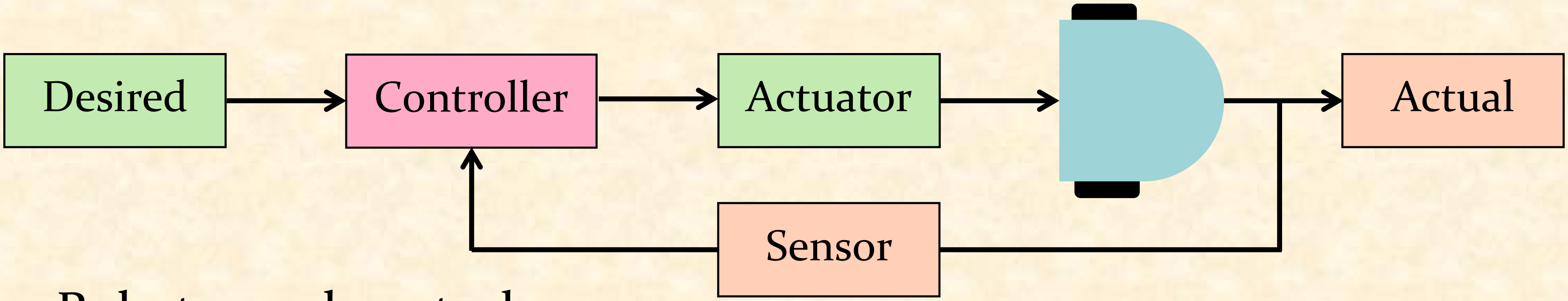


- Closed loop: the systems are interconnected (with feedback)



Control

The use of algorithms and feedback in engineered systems



Robot speed control

- Actuator: set the robot's speed
- Sensor: sense the robot's actual speed
- Control goals: set the robot's speed such that:
  - Stability: the robot maintains the desired speed
  - Performance: the robot responds quickly to changes
  - Robustness: the robot tolerates perturbation in dynamics

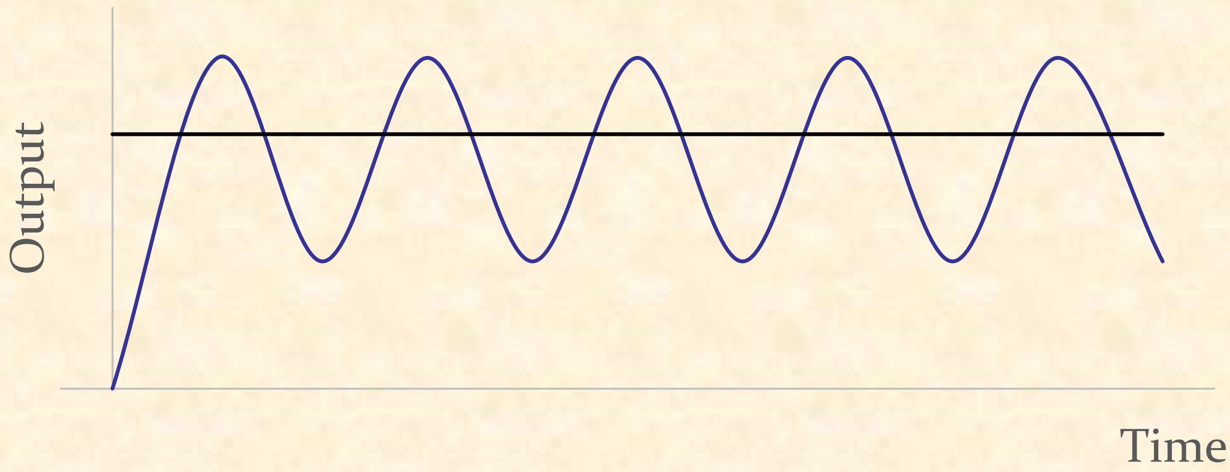
On-off controller

$$u = \begin{cases} u_{max} & \text{if } e > 0 \\ u_{min} & \text{if } e < 0 \end{cases}$$

```
error := set_point – measured
if error > 0.0 then
        output := max
else
        if error < 0.0 then
                output := min
        end
end
```

On-off controller

$$u = \begin{cases} u_{max} & \text{if } e > 0 \\ u_{min} & \text{if } e < 0 \end{cases}$$



Proportional controller

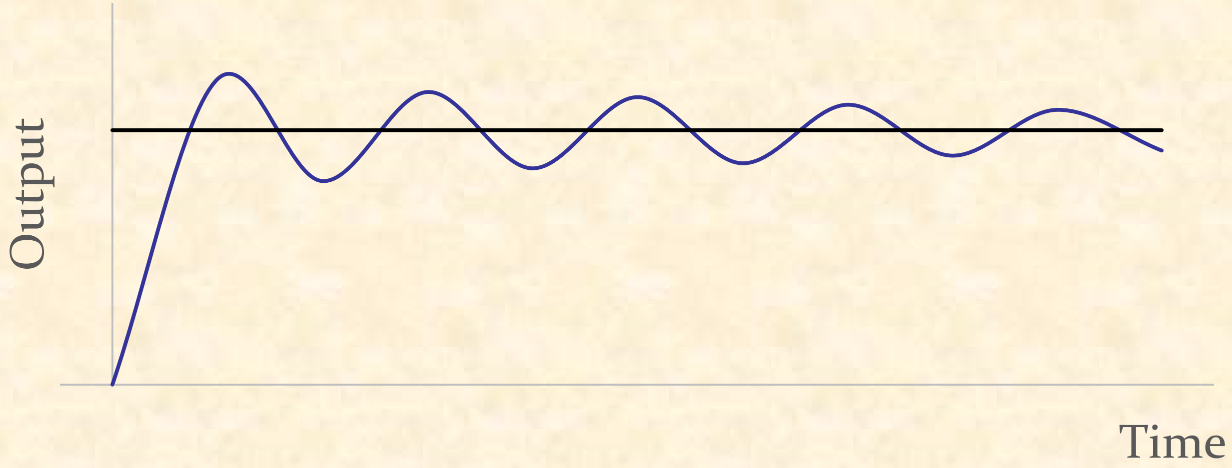
$$u(t) = k_p e(t)$$

```
error := set_point – measured

output := k_p * error
```

Proportional controller

$$u(t) = k_p e(t)$$



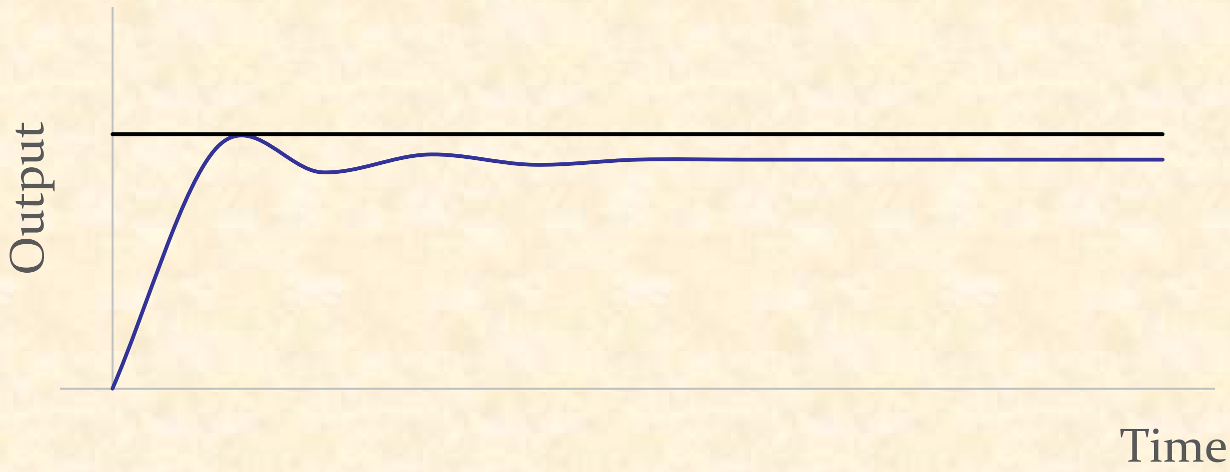
Proportional derivative controller

$$u(t) = k_p e(t) + k_d \frac{de(t)}{dt}$$

```
error := set_point – measured
proportional := k_p * error
derivative := k_d * (error - previous_error)/dt
output := proportional + derivative
previous_error := error
```

Proportional derivative controller

$$u(t) = k_p e(t) + k_d \frac{de(t)}{dt}$$



Proportional integral derivative controller

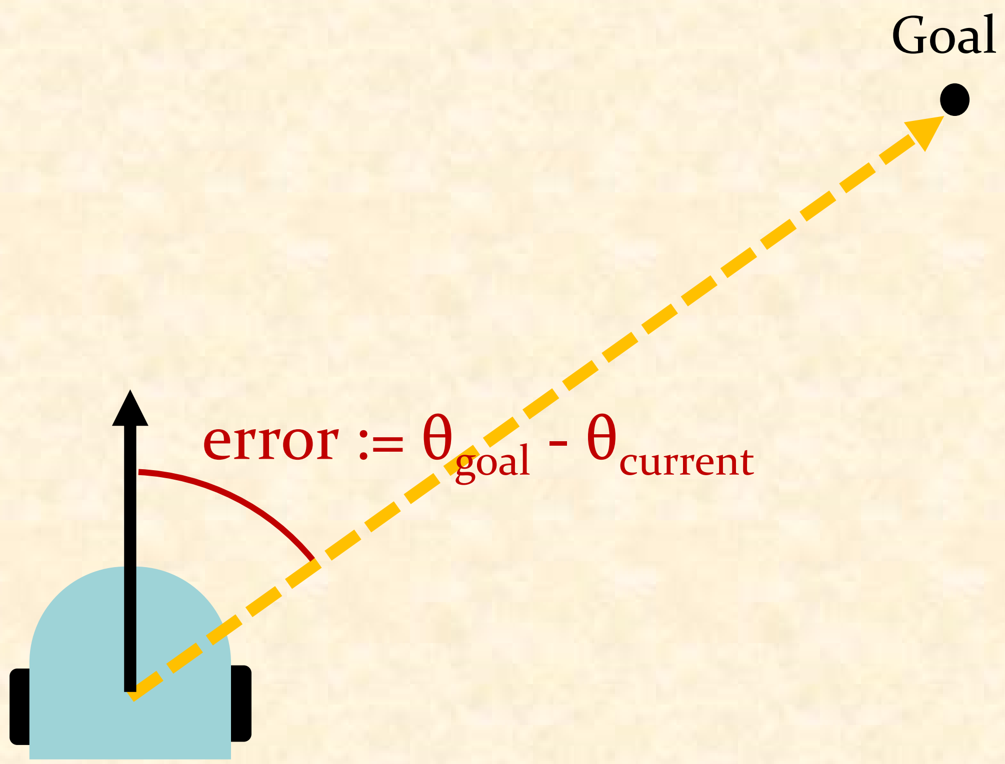
$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$

```
error := set_point – measured
accumulated_error := accumulated_error + error * dt
proportional := k_p * error
integral := k_i * accumulated_error
derivative := k_d * (error - previous_error)/dt
output := proportional + integral + derivative
previous_error := error
```

Proportional integral derivative controller

$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$

Go to goal



Control gains

Ziegler-Nicols method

- Set K_i and K_d to 0.
- Increase K_p until K_u at which point the output starts to oscillate.
- Use K_u and the oscillation period T_u to set the control gains.

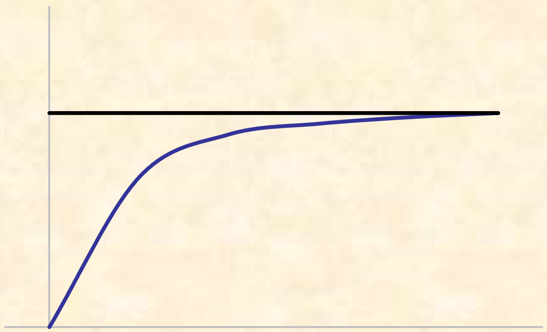
Control Type	K_p	K_i	K_d
P	$0.50K_u$	-	-
PI	$0.45K_u$	$1.2K_p/T_u$	-
PID	$0.60K_u$	$2K_p/T_u$	$K_pT_u/8$

Manual tuning!

P, PI, PID, ...?

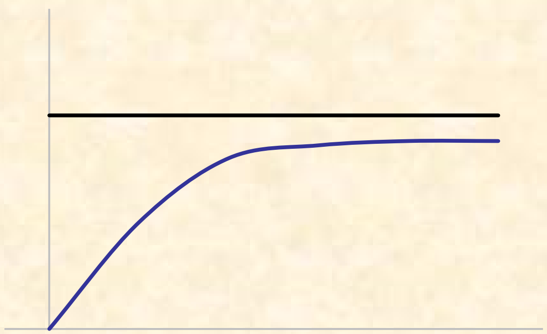
$$u(t) = k_p e(t) + k_i \int_0^t e(\tau)\, d\tau + k_d \frac{de(t)}{dt}$$

a.



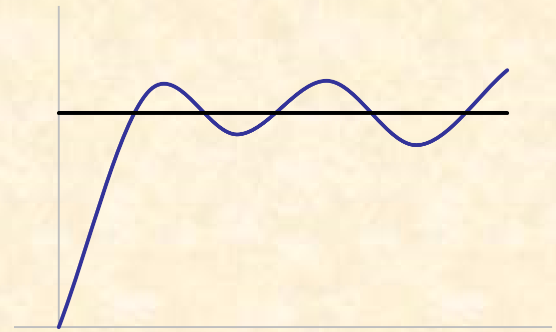
$k_p, k_i, k_d \neq 0$

b.



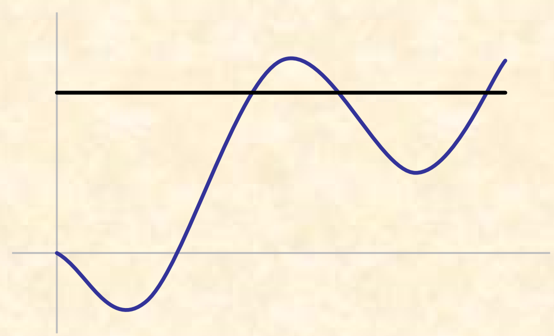
$k_i = 0$

c.



$k_d = 0$

d.



$k_p = 0$

Software engineering tips

```
make_with_gains (control_gains: ARRAY[REAL_64] )
    do
        k_p := control_gains[1]
        k_i := control_gains[2]
        k_d := control_gains[3]
    end
```

```
make
    do
        k_p := 1.0
        k_i := 0.1
        k_d := 0.3
    end
```

Software engineering tips

```
make
    do
        create robot.make_with_gains ( << 1.0, 0, 0.1 >> )
    end
```

```
make
    local
        control_gains: ARRAY[REAL_64]
        file: PLAIN_TEXT_FILE
        index: INTEGER
    do
        create control_gains.make_filled (0.0, 1, 3)
        create file.make_open_read (“param.txt”)
        from index := 1 until index > 3 or file.exhausted loop
            file.read_double
            control_gains.put (file.last_double, index)
            index := index + 1
        end
        file.close
        create robot.make_with_gains (control_gains)
    end
```

Software engineering tips

```
update_velocity ( … )
   …
   e := desired_angle – current_angle
   acc_e := acc_e + e * dt
   p := k_p * e
   i := k_i * acc_e
   d := k_d * (e - prev_e)/dt
   prev_e := e
   output := p + i + d
   …
```

```
update_velocity ( … )
   …
   e := desired_angle – current_angle
   output := pid_controller (e,dt)
   …

pid_controller ( … )
   …
```

Software engineering tips

```
class
  GO_TO_GOAL_CONTROLLER
feature
  update_velocity ( … )
  pid_controller ( … )
  …
```

```
class
  GO_TO_GOAL_CONTROLLER
feature
  update_velocity ( … )
  …

class
  PID_CONTROLLER
feature
  pid_controller ( … )
  …
```