

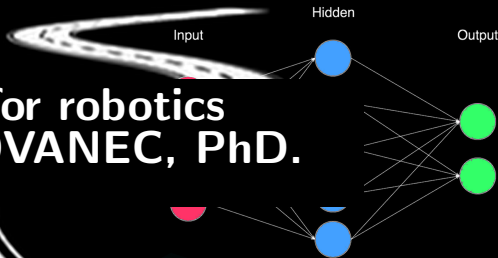
$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

(The New Action Value = The Old Value) + The Learning Rate  $\times$  (The New Information - the Old Information)

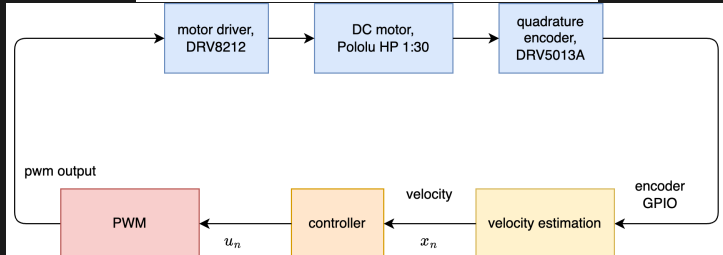
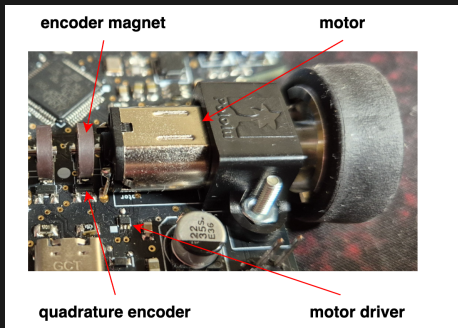


# Algorithms for robotics

## Michal CHOVANEC, PhD.

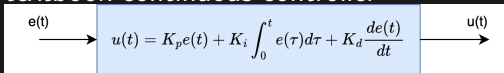


# motor velocity controller

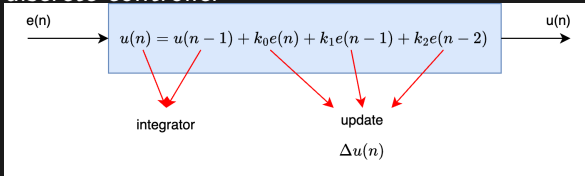


# PID control

- textbook continuous controller



- discrete controller



$$k_0 = K_p + K_i \Delta t + \frac{K_d}{\Delta t}$$

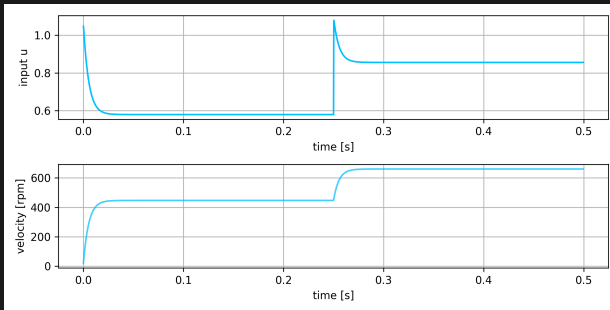
$$k_1 = -K_p - 2 \frac{K_d}{\Delta t}$$

$$k_2 = \frac{K_d}{\Delta t}$$

# PID control - P only

- target value : **1000rpm**
- P-only control causes **steady state error**

$$u(n+1) = u(n-1) + k_p e(n) - k_p e(n-1)$$
$$k_p = 0.01$$



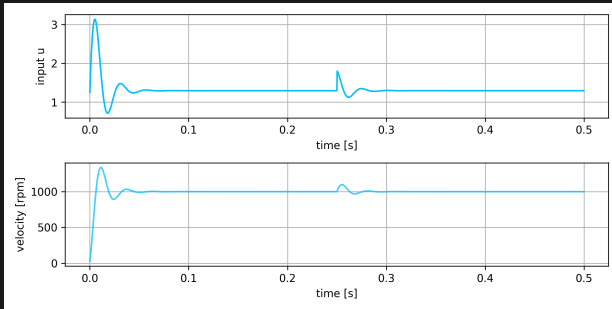
# PID control - PI

- target value : **1000rpm**
- PI control **removes steady state error**
- too high I term causes **oscillations and overshoot**

$$u(n+1) = u(n-1) + (k_p + k_i \Delta t)e(n) - k_p e(n-1)$$

$$k_p = 0.01$$

$$k_i = 0.005$$



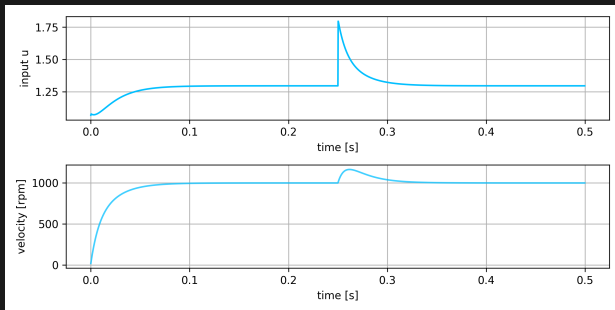
# PID control - PI

- target value : **1000rpm**
- correct tuned PI controller for 1st order system
- **no overshoot, no steady state error**

$$u(n+1) = u(n-1) + (k_p + k_i \Delta t)e(n) - k_p e(n-1)$$

$$k_p = 0.01$$

$$k_i = 0.0002$$



# complete generic discrete PID

- 1 calculate u-change candidate :

$$\Delta \hat{u}(n) = k_0 e(n) + k_1 e(n) + k_2 e(n)$$

- 2 clip maximum allowed u-change, to avoid u-kick :

$$\Delta u(n) = \text{clip}(\Delta \hat{u}(n), -du_{min}, du_{max})$$

- 3 clip maximum allowed u value, to avoid saturation / windup :

$$u(n) = \text{clip}(u(n-1) + \Delta u(n), -u_{min}, u_{max})$$

# complete generic discrete PID

```
def __init__(self, kp, ki, kd, antiwindup = 10**10, du_max=10**10):
    self.k0 = kp + ki + kd
    self.k1 = -kp -2.0*kd
    self.k2 = kd

    self.e0 = 0.0
    self.e1 = 0.0
    self.e2 = 0.0

    self.antiwindup = antiwindup
    self.du_max      = du_max

def forward(self, xr, x, u_prev):
    # error compute
    self.e2 = self.e1
    self.e1 = self.e0
    self.e0 = xr - x

    du = self.k0*self.e0 + self.k1*self.e1 + self.k2*self.e2

    #kick clipping, maximum output value change limitation
    du = numpy.clip(du, -self.du_max , self.du_max)

    #antiwindup, maximum output value limitation
    u = numpy.clip(u_prev + du, -self.antiwindup, self.antiwindup)

    return u
```



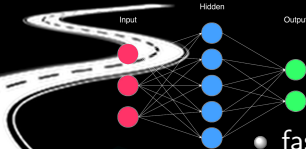
# PID control - issues

- mostly hand tuned parameters
- only single input, single output systems
- no implicit noise filtering

# motor velocity controller

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha [r_{t+1} + \lambda \max_a Q(s_{t+1}, a) - Q(s_t, a_t)]$$

(The New Action Value = The Old Value) + The Learning Rate  $\times$  (The New Information - the Old Information)



- fastest line following
- 3 rounds
- obstacles : broken line, brick, loop, colored line

