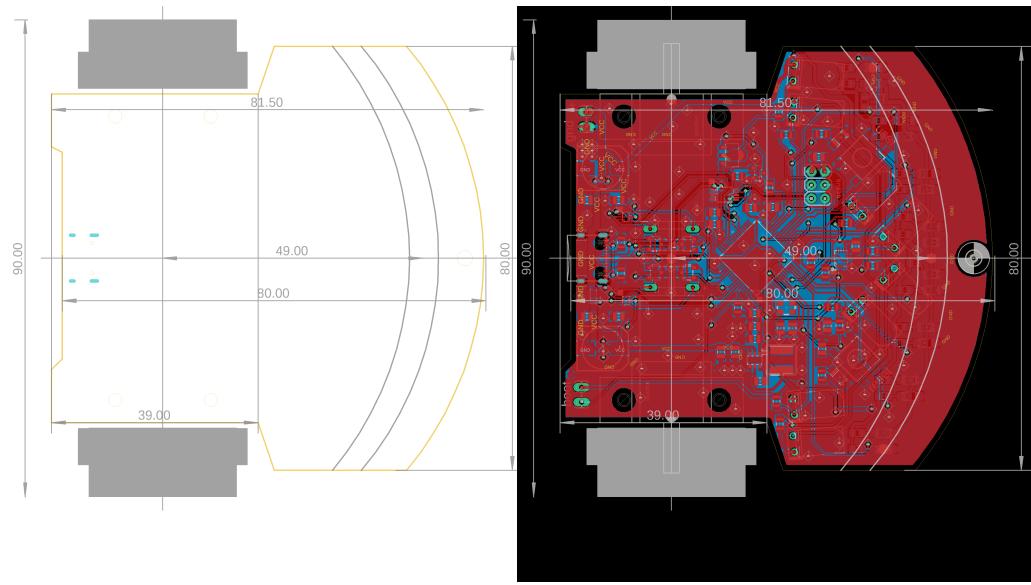


Chapter 1

Hardware description



(a) mechanical dimensions

(b) overall PCB layout

Figure 1.1: Robot PCB

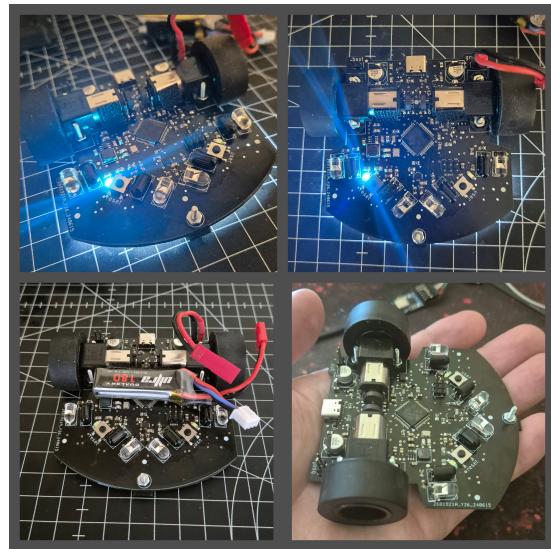


Figure 1.2: Robot photo

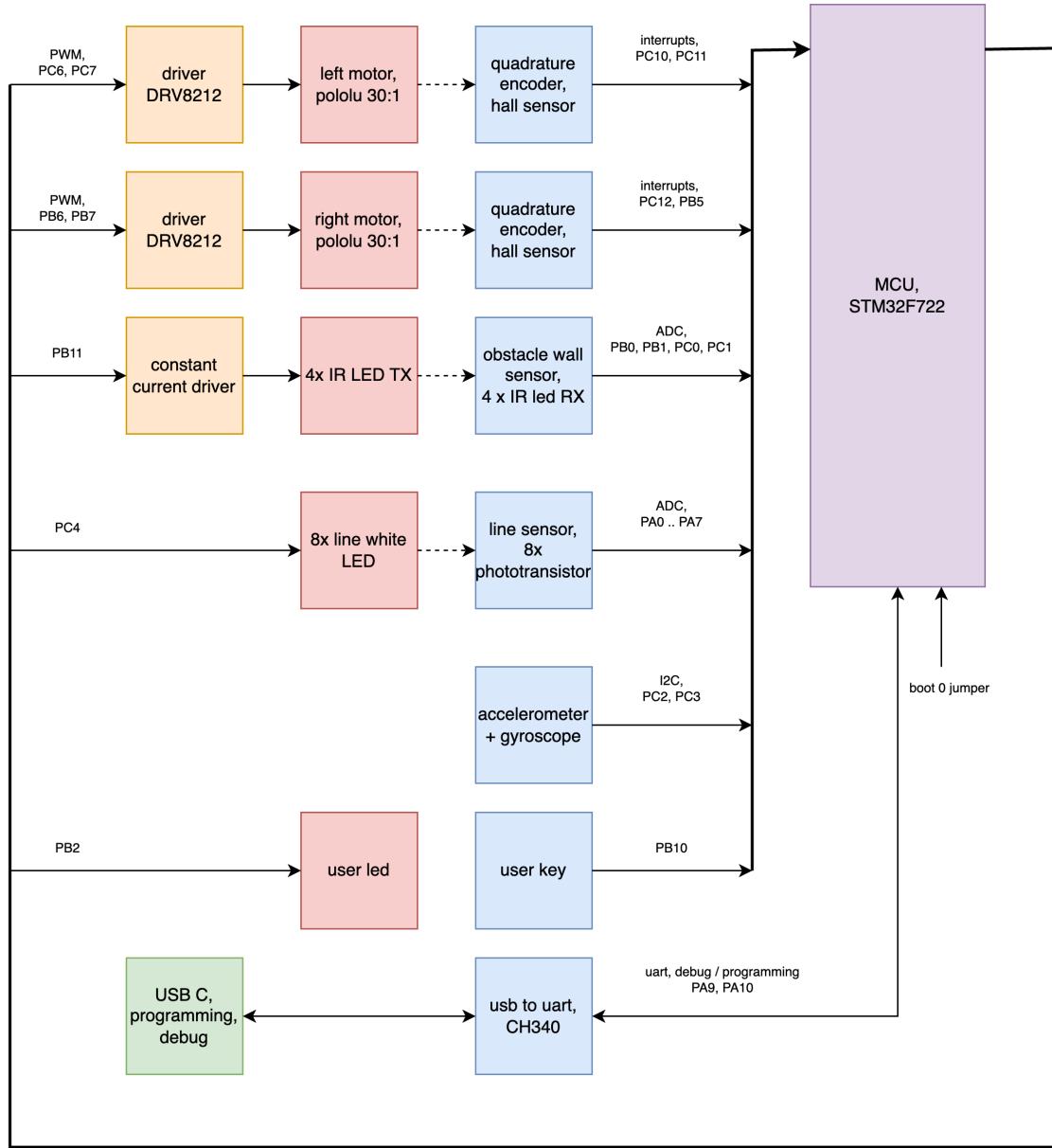


Figure 1.3: block diagram of DC motor version

CHAPTER 1. HARDWARE DESCRIPTION

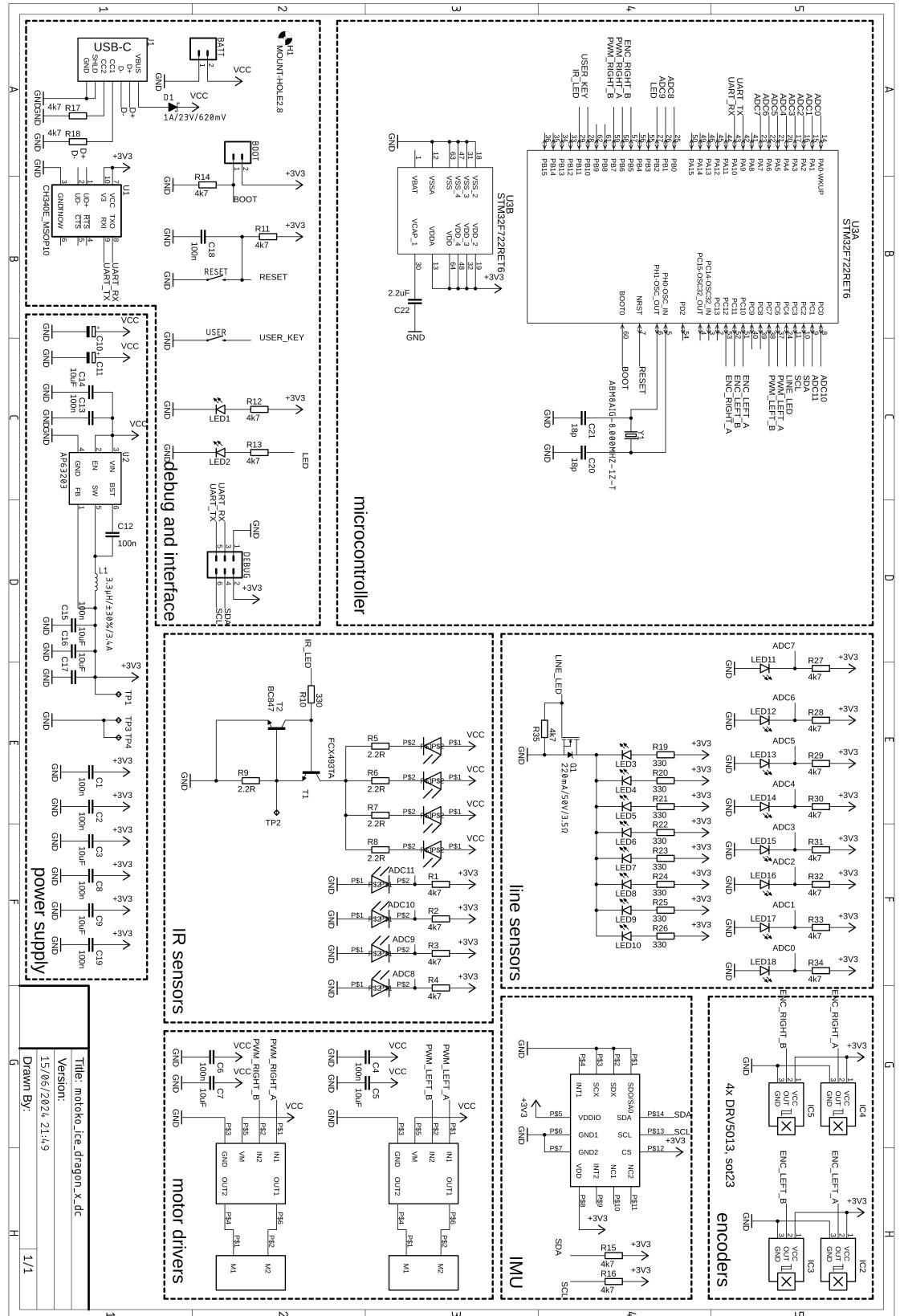


Figure 1.4: schematic diagram of DC motor version

pin number	pin name	function	peripheral	AF number
14	PA0	line sensor ADC0, right	ADC1	analog in
15	PA1	line sensor ADC1	ADC1	analog in
16	PA2	line sensor ADC2	ADC1	analog in
17	PA3	line sensor ADC3	ADC1	analog in
20	PA4	line sensor ADC4	ADC1	analog in
21	PA5	line sensor ADC5	ADC1	analog in
22	PA6	line sensor ADC6	ADC1	analog in
23	PA7	line sensor ADC7, left	ADC1	analog in
25	PB0	IR sensor ADC8, front left	ADC1	analog in
26	PB1	IR sensor ADC9, left	ADC1	analog in
8	PC0	IR sensor ADC10, right	ADC1	analog in
9	PC1	IR sensor ADC11, front right	ADC1	analog in
51	PC10	encoder left A	GPIOC	
52	PC11	encoder left B	GPIOC	
53	PC12	encoder right A	GPIOC	
57	PB5	encoder right B	GPIOB	
37	PC6	PWM left A	TIM3_CH1	AF2
38	PC7	PWM left B	TIM3_CH2	AF2
58	PB6	PWM right A	TIM4_CH1	AF2
59	PB7	PWM right B	TIM4_CH2	AF2
10	PC2	IMU I2C, SDA	GPIOC	
11	PC3	IMU I2C, SCL	GPIOC	
42	PA9	uart TX	UART1	AF7
43	PA10	uart RX	UART1	AF7
27	PB2	user LED	GPIOB	
28	PB10	user Key	GPIOB	
29	PB11	IR LED control	GPIOB	
24	PC4	line LED control	GPIOC	
7	NRST	reset		
60	BOOT0	firmware bootloader control		

Table 1.1: pin mapping and function

Chapter 2

System identification

2.1 State space models

System state is column vector, with N rows

$$x(n) = \begin{bmatrix} x_0(n) \\ x_1(n) \\ \dots \\ x_{N-1}(n) \end{bmatrix} \quad (2.1)$$

For single output system this vector contains single value, e.g. the motor velocity $\omega(n)$

$$x(n) = [\omega(n)] \quad (2.2)$$

If we have 2nd order system, e.g. servo with inertia, the state will contain two elements :

- motor shaft velocity $\omega(n)$, measured in rad/s
- motor shaft angle $\theta(n)$, measured in rad

$$x(n) = \begin{bmatrix} \omega(n) \\ \theta(n) \end{bmatrix} \quad (2.3)$$

More accurate servo model observes also motor current $i(n)$, which gives us three element state vector

$$x(n) = \begin{bmatrix} i(n) \\ \omega(n) \\ \theta(n) \end{bmatrix} \quad (2.4)$$

Robot moving in 2D plane have usually state containing robot position x' , y' , and robot orientation θ

$$x(n) = \begin{bmatrix} x'(n) \\ y'(n) \\ \theta(n) \end{bmatrix} \quad (2.5)$$

The number of rows in state vector $x(n)$ is called **system order**.

The dynamical system is controlled with M inputs, stacked in column input vector $u(n)$

$$u(n) = \begin{bmatrix} u_0(n) \\ u_1(n) \\ \dots \\ u_{M-1}(n) \end{bmatrix} \quad (2.6)$$

The system with single input, e.g. motor with current control, input vector contains single value

$$u(n) = [i(n)] \quad (2.7)$$

Differential drive robot where are controlled two independent motors have two inputs

$$u(n) = \begin{bmatrix} i_{left}(n) \\ i_{right}(n) \end{bmatrix} \quad (2.8)$$

Relation between control input $u(n)$, current state $x(n)$ and next state $x(n+1)$ can be modeled using **linear state space model**

$$x(n+1) = Ax(n) + Bu(n) \quad (2.9)$$

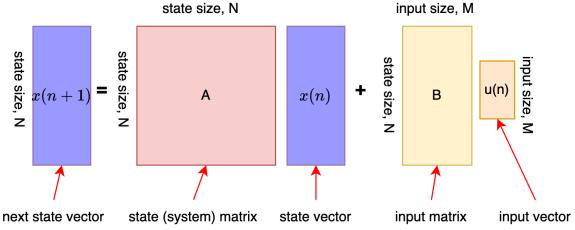


Figure 2.1: State space model matrices shapes

This model can capture dynamics of any linear system. Sometimes the state $x(n)$ can't be measured directly, and we observe only system output $y(n) = Cx(n)$. Where matrix C have k-rows and n-columns. In our case we consider we have access to full state $x(n)$. There is of course the continuous form of state space model

$$\frac{dx}{dt} = A^c x(t) + B^c u(t) \quad (2.10)$$

note : the martices A^c and B^c differs from discrete system matrices A and B . They are related with bilinear transform as

$$S_a = (I - \frac{\Delta T}{2} A^c)^{-1} \quad (2.11)$$

$$S_b = I + \frac{\Delta T}{2} A^c \quad (2.12)$$

$$A = S_a S_b \quad (2.13)$$

$$B = S_a B^c \Delta T \quad (2.14)$$

where ΔT is sampling period. This is mostly used discretisation, from continuous (e.g. physical) model into discrete form. Following python code convert continuous system into discrete system. If system contains output matrix C discretisation doesn't effects its values.

```

1 def c2d(a, b, c, dt):
2     i = numpy.eye(a.shape[0])
3
4     tmp_a = numpy.linalg.inv(i - (0.5*dt)*a)
5     tmp_b = i + (0.5*dt)*a
6
7     a_disc = tmp_a@tmp_b

```

```

8     b_disc = (tmp_a*dt)@b
9
10    return a_disc, b_disc, c

```

Process of system identification is finding matrices A , B (and C). From this model can be synthesised controller, or system able to plan multiple steps ahead.

2.2 System examples

Motor velocity model, can be approximated as 1st order system.

Model will have following parameters

- sampling frequency 4kHz, $\Delta T = 1/4000$
- maximum control input $u_{max} = 2$
- motor constant $k = 17$
- motor time constant $\tau = 29$ milliseconds

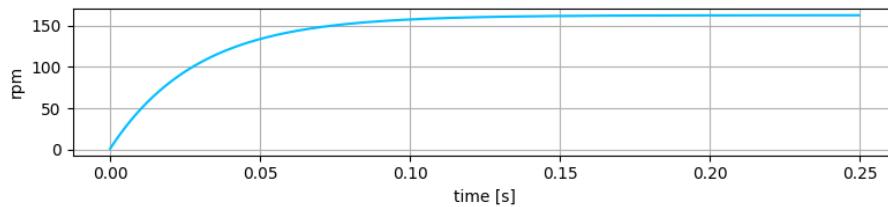


Figure 2.2: Motor step response

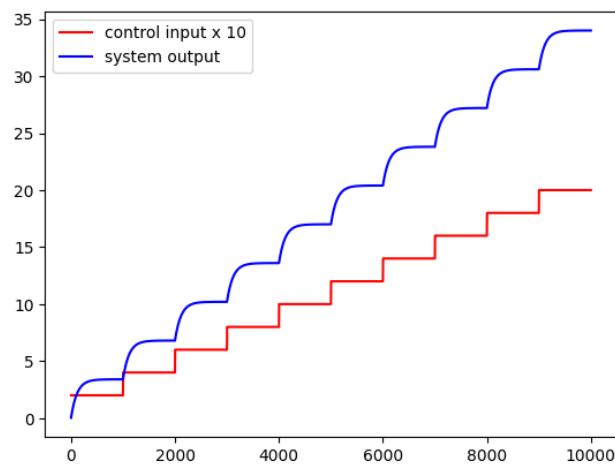


Figure 2.3: Motor constant estimation

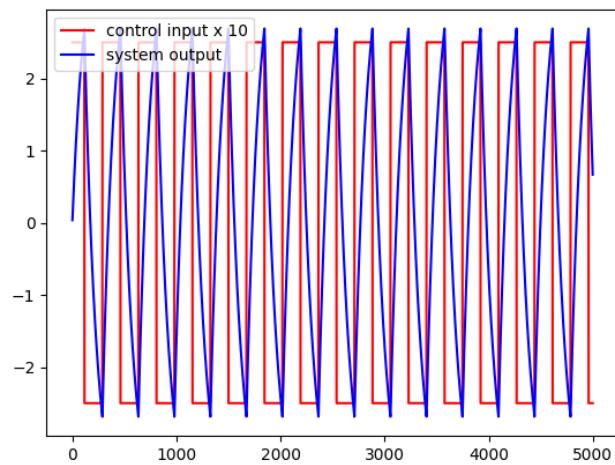


Figure 2.4: Motor time constant estimation

Chapter 3

Motor control

motor velocity controller

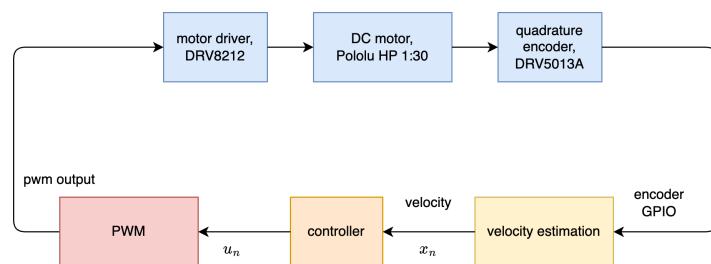


Figure 3.1: Velocity control overview

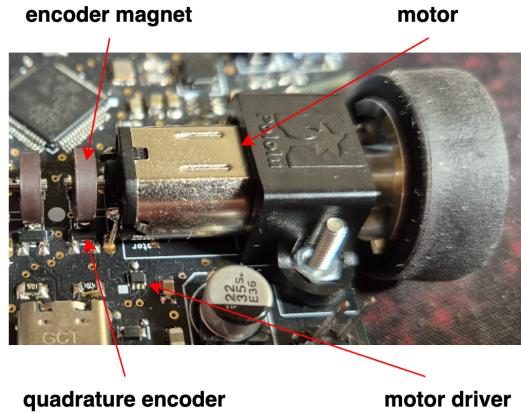


Figure 3.2: Real system photo

3.1 PID control

$$e(t) \rightarrow u(t) = K_p e(t) + K_i \int_0^t e(\tau) d\tau + K_d \frac{de(t)}{dt} \rightarrow u(t)$$

Figure 3.3: Textbook continuous PID controller

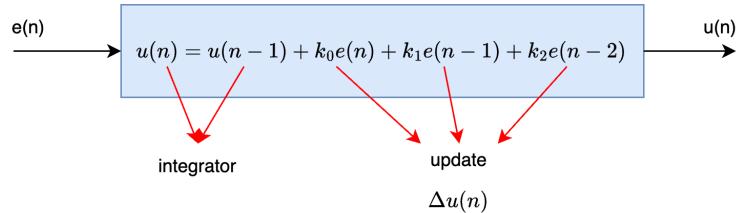


Figure 3.4: Discrete PID controller

$$k_0 = K_p + K_i \Delta T + \frac{K_d}{\Delta T} \quad (3.1)$$

$$k_1 = -K_p - 2 \frac{K_d}{\Delta T} \quad (3.2)$$

$$k_2 = \frac{K_d}{\Delta T} \quad (3.3)$$

PID control - P only controller

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

$$u(n + 1) = u(n - 1) + k_p e(n) - k_p e(n - 1) \quad (3.4)$$

$$k_p = 0.01 \quad (3.5)$$

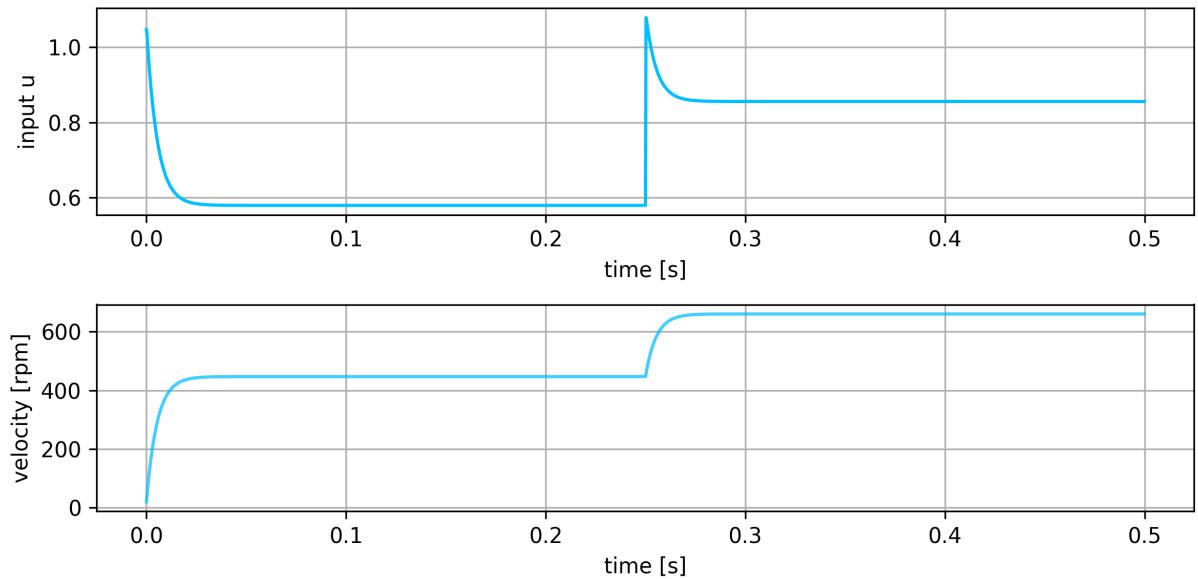


Figure 3.5: P-only controller

PID control - PI

too high I term

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

$$u(n + 1) = u(n - 1) + (k_p + k_i\Delta T)e(n) - k_p e(n - 1) \quad (3.6)$$

$$k_p = 0.01 \quad (3.7)$$

$$k_i\Delta T = 0.005 \quad (3.8)$$

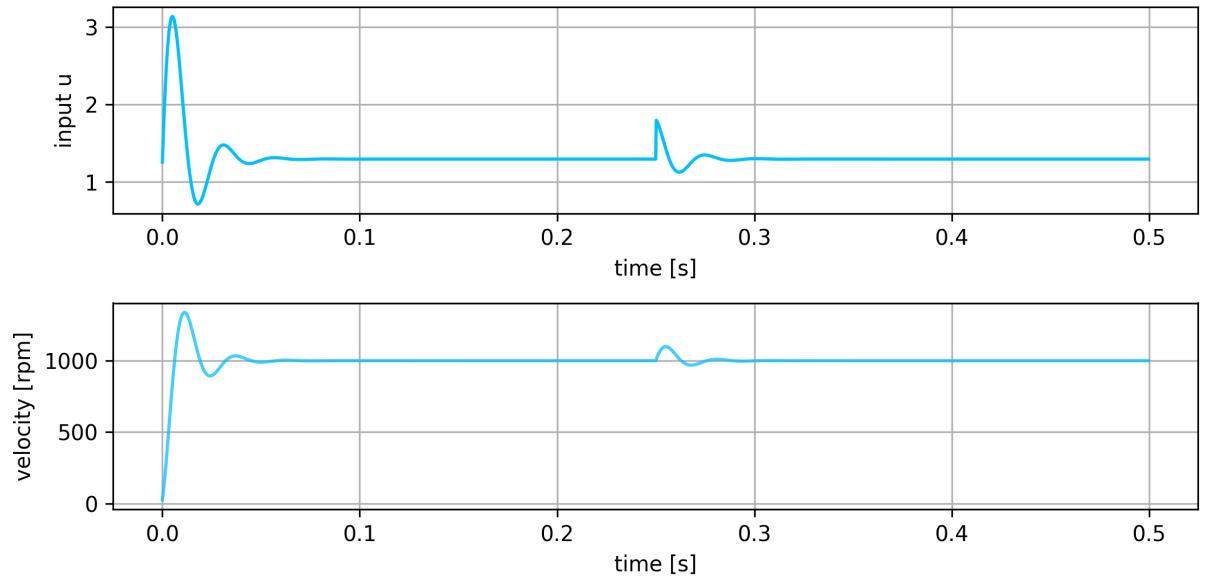


Figure 3.6: PI controller

correct I term

- target value : **1000rpm**
- correct tuned PI controller for 1st order system

- **no overshoot, no steady state error**

$$u(n+1) = u(n-1) + (k_p + k_i \Delta T)e(n) - k_p e(n-1) \quad (3.9)$$

$$k_p = 0.01 \quad (3.10)$$

$$k_i \Delta T = 0.0002 \quad (3.11)$$

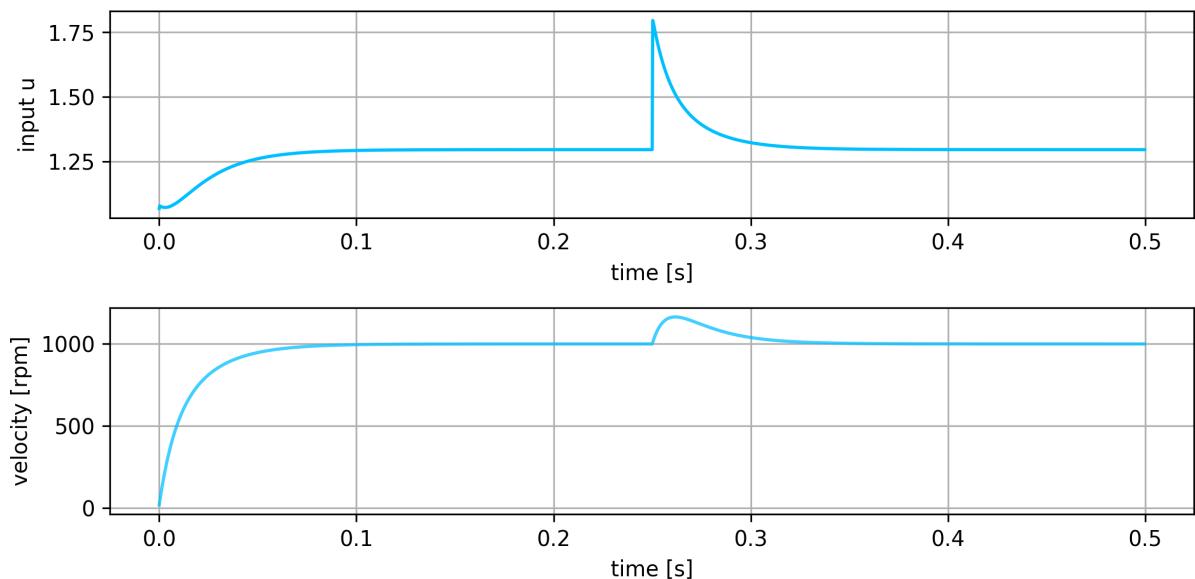


Figure 3.7: PI controller with correct parameters

complete discrete PID algorithm

1. calculate u-change candidate :

$$\Delta\hat{u}(n) = k_0e(n) + k_1e(n) + k_2e(n)$$

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 :

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

Full Python code example for discrete PID controller :

```

1  class PID:
2
3  def __init__(self, kp, ki, kd, antiwindup = 10**10, du_max=10**10):
4      self.k0 = kp + ki + kd
5      self.k1 = -kp -2.0*kd
6      self.k2 = kd
7
8      self.e0 = 0.0
9      self.e1 = 0.0
10     self.e2 = 0.0
11
12     self.antiwindup = antiwindup
13     self.du_max = du_max
14
15 def forward(self, xr, x, u_prev):
16     # error compute
17     self.e2 = self.e1
18     self.e1 = self.e0
19     self.e0 = xr - x
20
21     # u-change computing
22     du = self.k0*self.e0 + self.k1*self.e1 + self.k2*self.e2
23
24     # kick clipping, maximum output value change limitation
25     du = numpy.clip(du, -self.du_max, self.du_max)
26
27     # antiwindup, maximum output value limitation
28     u = numpy.clip(u_prev + du, -self.antiwindup, self.antiwindup)
29
30     return u

```

3.2 Optimal control

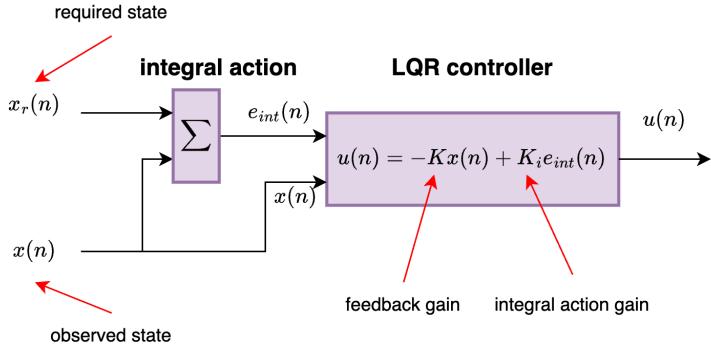


Figure 3.8: LQR controller with integral action

$$u(n) = -Kx(n) + K_ie_{int}(n) \quad (3.12)$$

$$e_{int}(n) = e_{int}(n-1) + x_r(n) - x(n) \quad (3.13)$$

where :

- $x(n)$ is observerd state, column vector, with shape $(N, 1)$
- $x_r(n)$ is required state, column vector, with shape $(N, 1)$
- $u(n)$ is controller output, column vector, with shape $(M, 1)$
- K is feedback gain matrix, with shape (M, N)
- K_i is integral gain matrix, with shape (M, N)

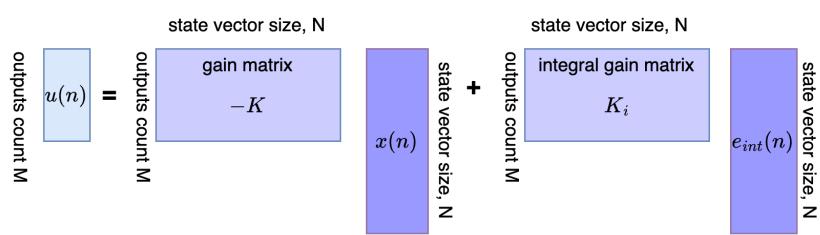


Figure 3.9: Matrices shapes in LQR controller