DC Motor

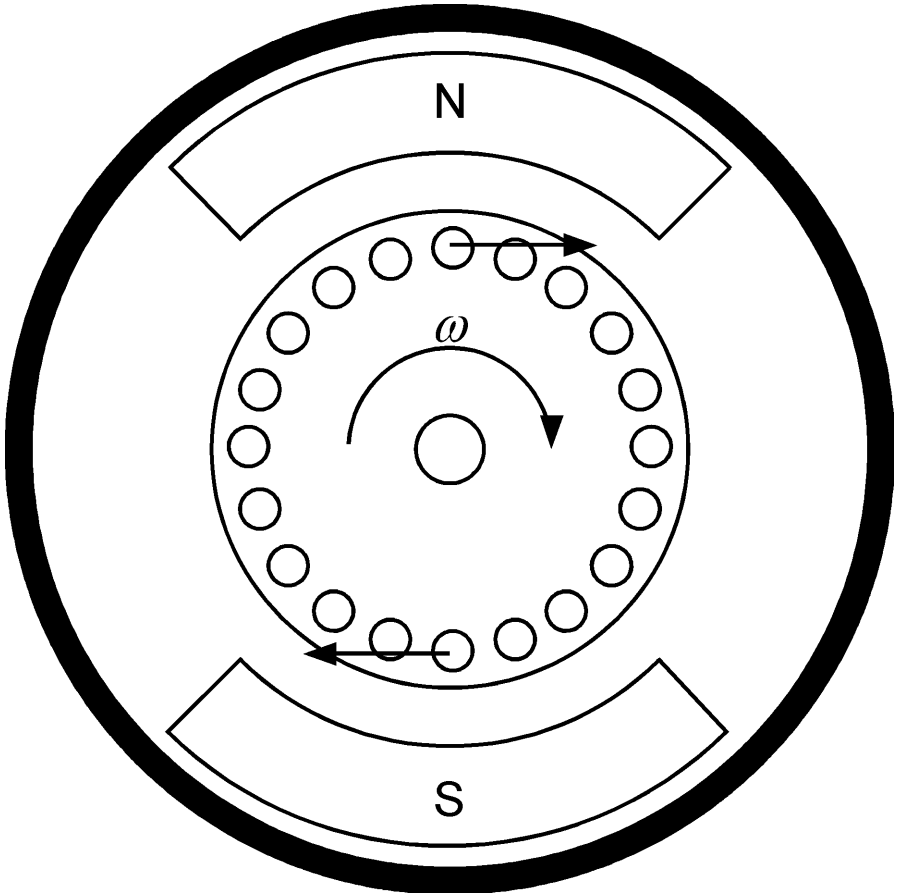
Modeling and Control

직류 모터의 모델

- 자기장(magnetic field) 내에서 전류가 흐르는 도체에는 힘이 작용하며, 이 힘의 세기는 자기장의 세기와 도체에 흐르는 전류의 크기에 비례한다.

- 자기장 내에서 도체를 움직이면 이 도체에는 기전력(emf, electro-motive force)이 발생하며, 이 기전력의 크기는 도체가 움직이는 속도에 비례한다.

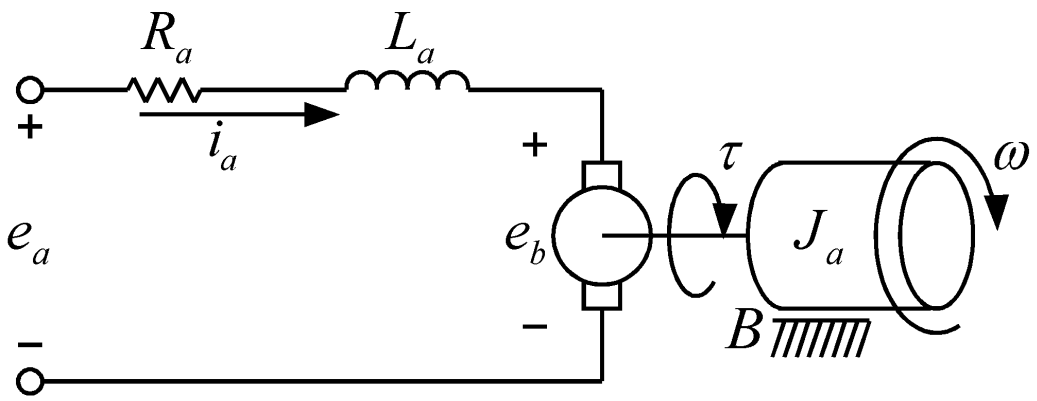
직류 모터의 모델



$$\tau = K_t i_a$$

$$e_b = K_b \frac{d\theta}{dt} = K_b \omega$$

직류 모터의 모델



$$\tau = K_t i_a$$

$$e_b = K_b \frac{d\theta}{dt} = K_b \omega$$

$$e_a = R_a i_a + L_a \frac{di_a}{dt} + e_b = R_a i_a + L_a \frac{di_a}{dt} + K_b \frac{d\theta}{dt}$$

$$\tau = K_t i_a = J_a \frac{d^2\theta}{dt^2} + B \frac{d\theta}{dt}$$

직류 모터의 모델: 전달 함수

$$E_a(s) = R_a I_a(s) + L_a s I_a(s) + K_b s\Theta(s)$$

$$= \left(R_a + L_a s\right) I_a(s) + K_b s\Theta(s)$$

$$K_t I_a(s) = J_a s^2 \Theta(s) + Bs\Theta(s) = \left(J_a s^2 + Bs\right)\Theta(s)$$

$$K_t E_a(s) = \left(R_a + L_a s\right) K_t I_a(s) + K_t K_b s\Theta(s)$$

$$K_t E_a(s) = \left(R_a + L_a s\right)\left(J_a s^2 + Bs\right)\Theta(s) + K_t K_b s\Theta(s)$$

$$\frac{\Theta(s)}{E_a(s)} = \frac{K_t}{\left(R_a + L_a s\right)\left(J_a s^2 + Bs\right) + K_t K_b s}$$

$$\frac{s\Theta(s)}{E_a(s)} = \frac{\Omega(s)}{E_a(s)} = \frac{K_t}{\left(R_a + L_a s\right)\left(J_a s + B\right) + K_t K_b}$$

모터의 기계 파워와 전기 파워

- 손실이 없다고 가정: 기계 파워=전기 파워

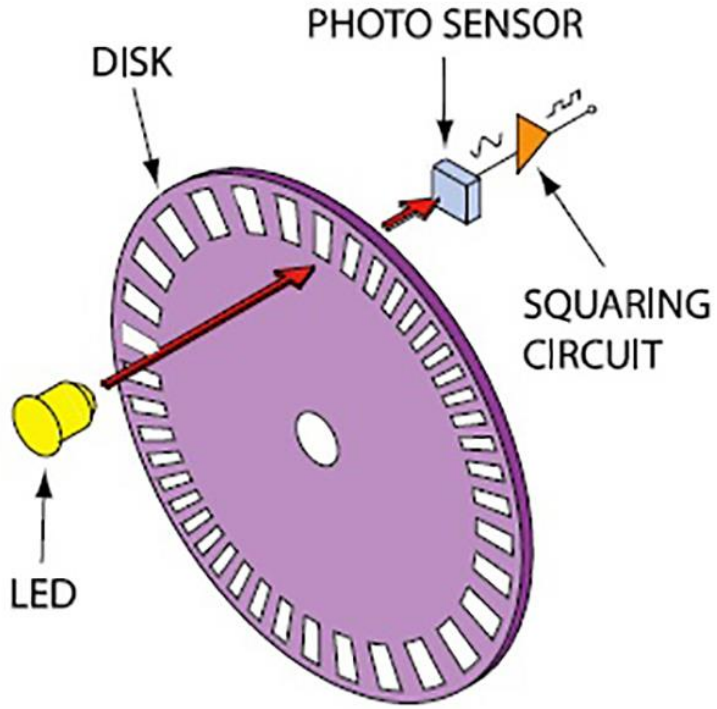
$$P_e = e_a i_a$$

$$P_m = \tau\omega = K_t i_a \frac{e_b}{K_b}$$

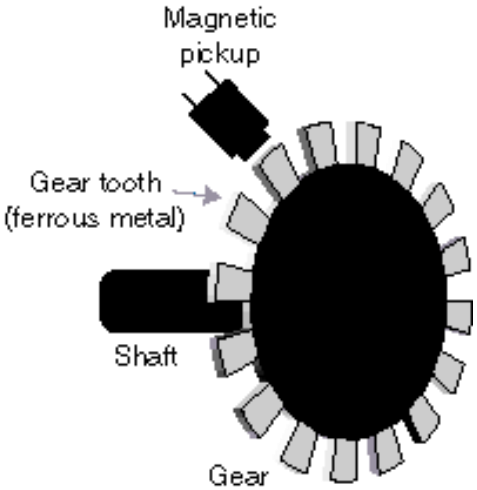
$$P_m = P_e, e_b = e_a \left(R_a = 0, L_a = 0\right)$$

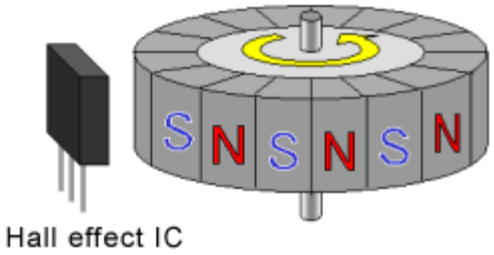
$$\Rightarrow K_t = K_b$$

Optical Encoder



Magnetic Encoder





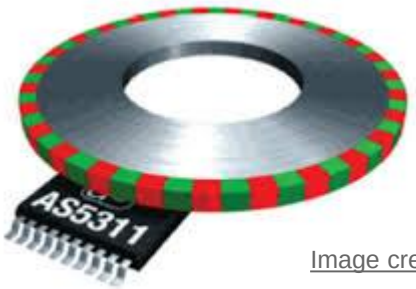
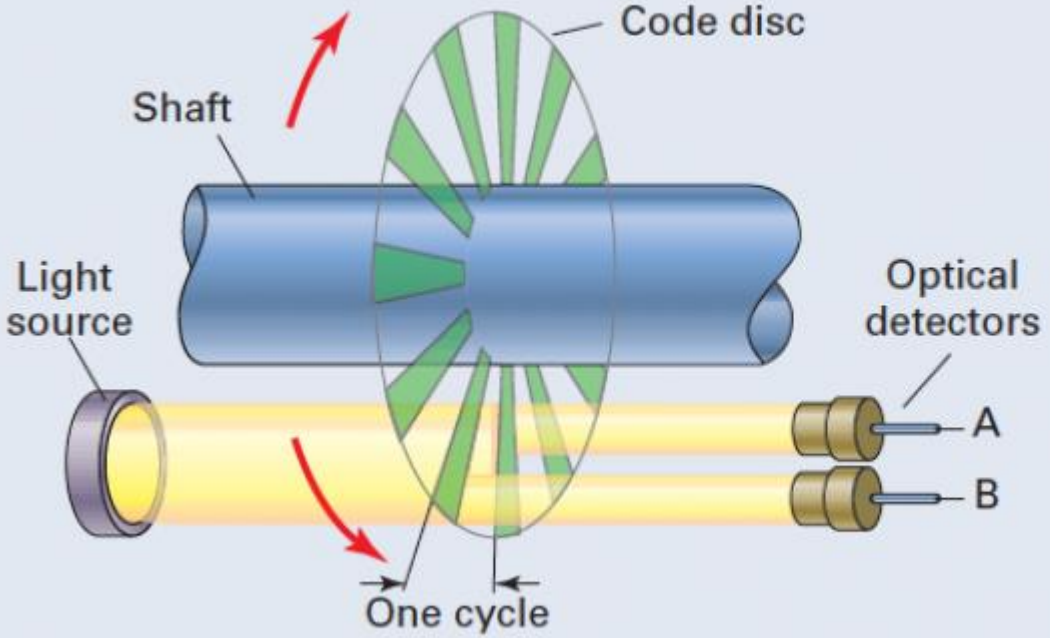
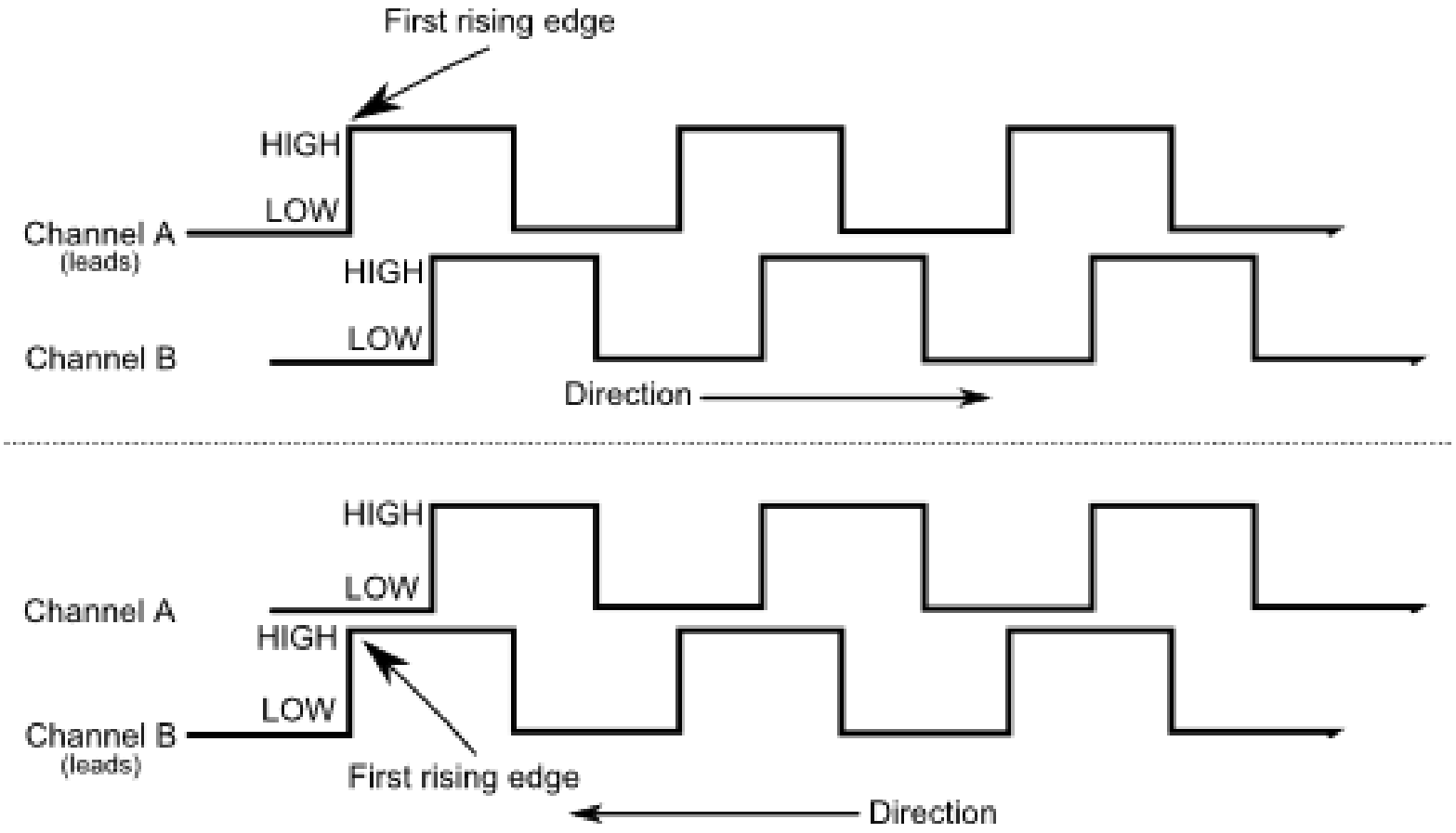


Image credit: ams AG

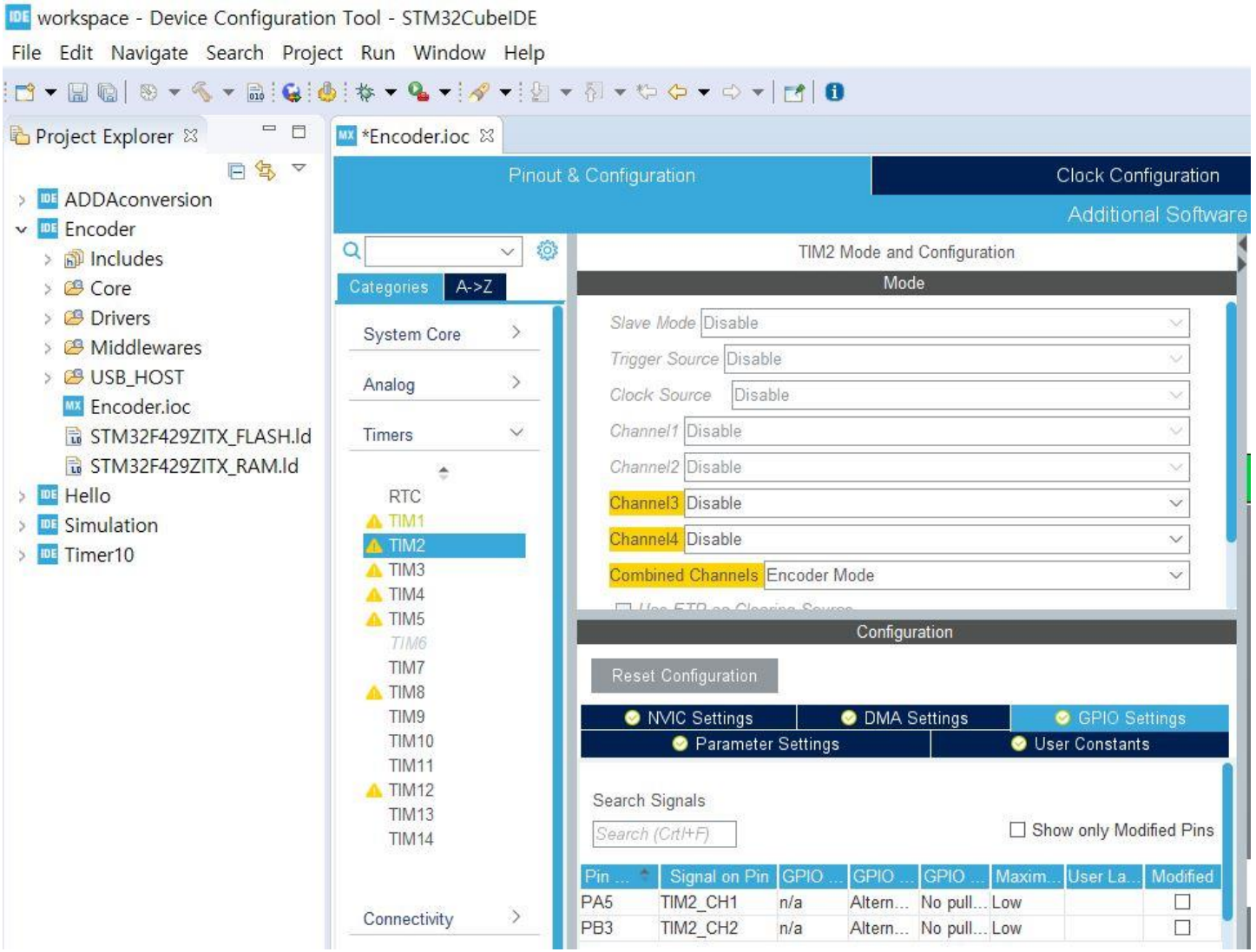
How an optical encoder works



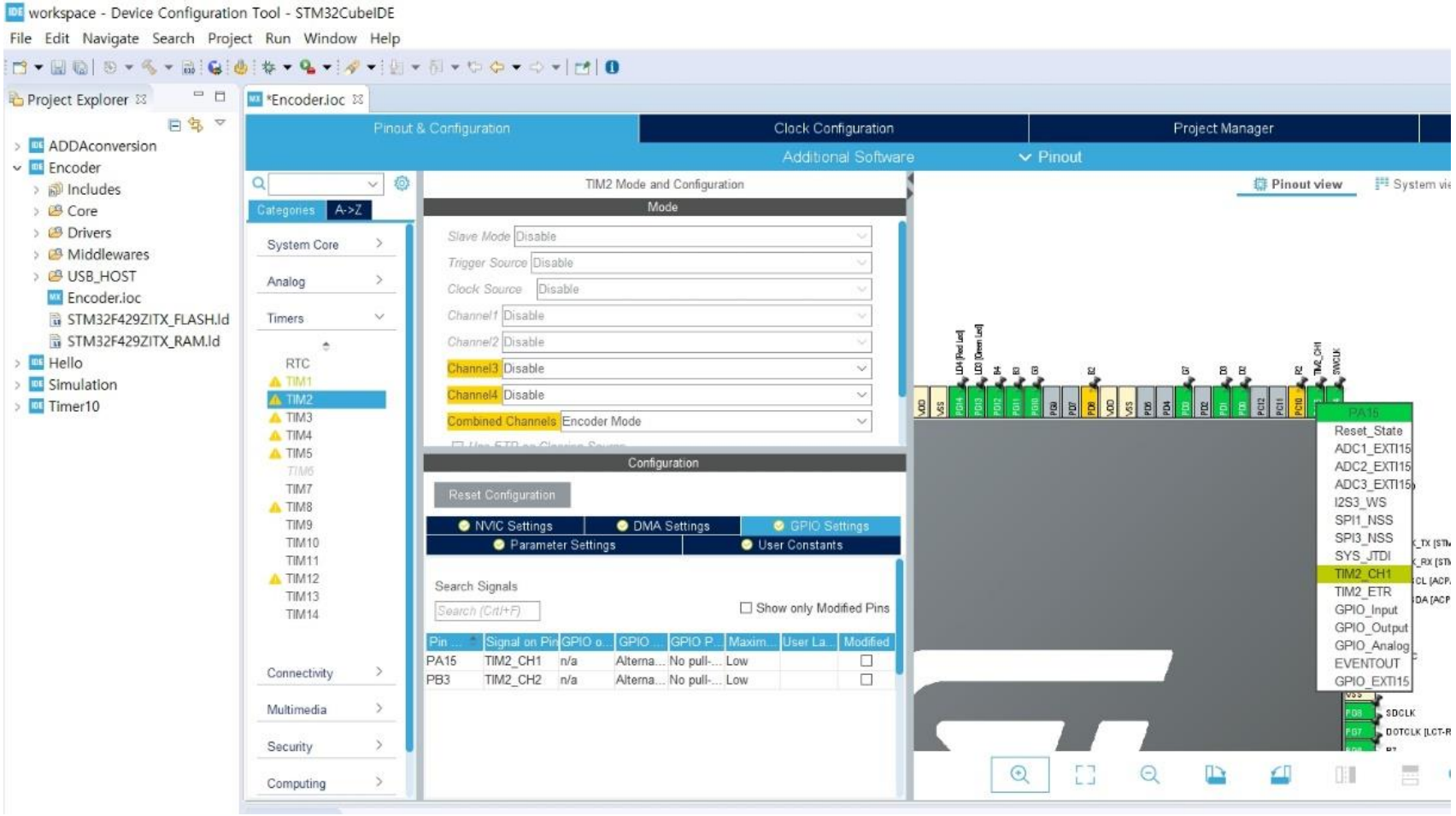
Quadrature encoders combine the output of two sensing channels, A and B, producing a signa that registers movements four times smaller than the cycle of the line pattern printed on code disk.

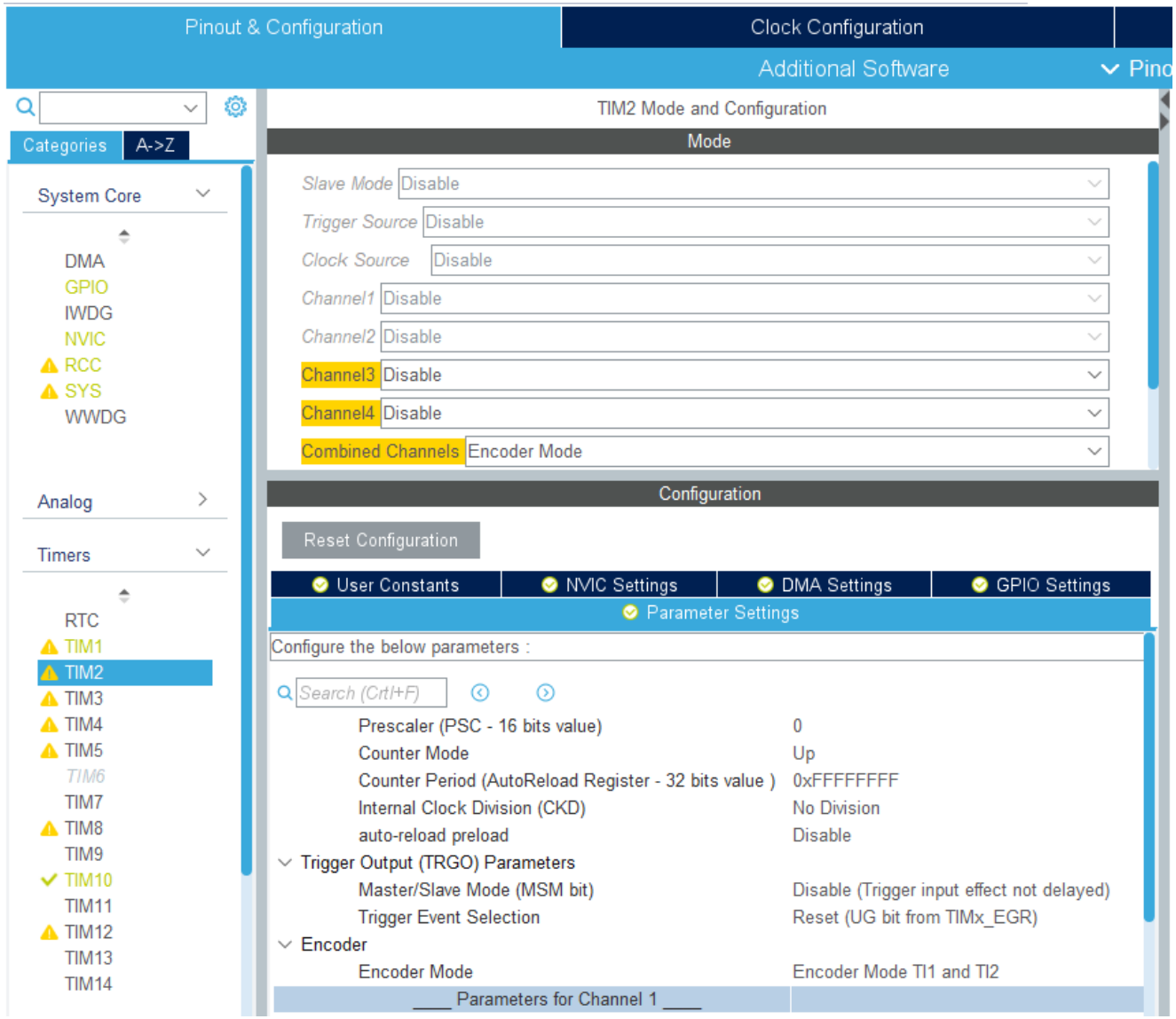

First rising edge
HIGH
LOW
Channel A
(leads)
HIGH
LOW
Channel B
Direction
HIGH
LOW
Channel A
HIGH
LOW
Channel B
(leads)
First rising edge
Direction

Timer 2 configuration



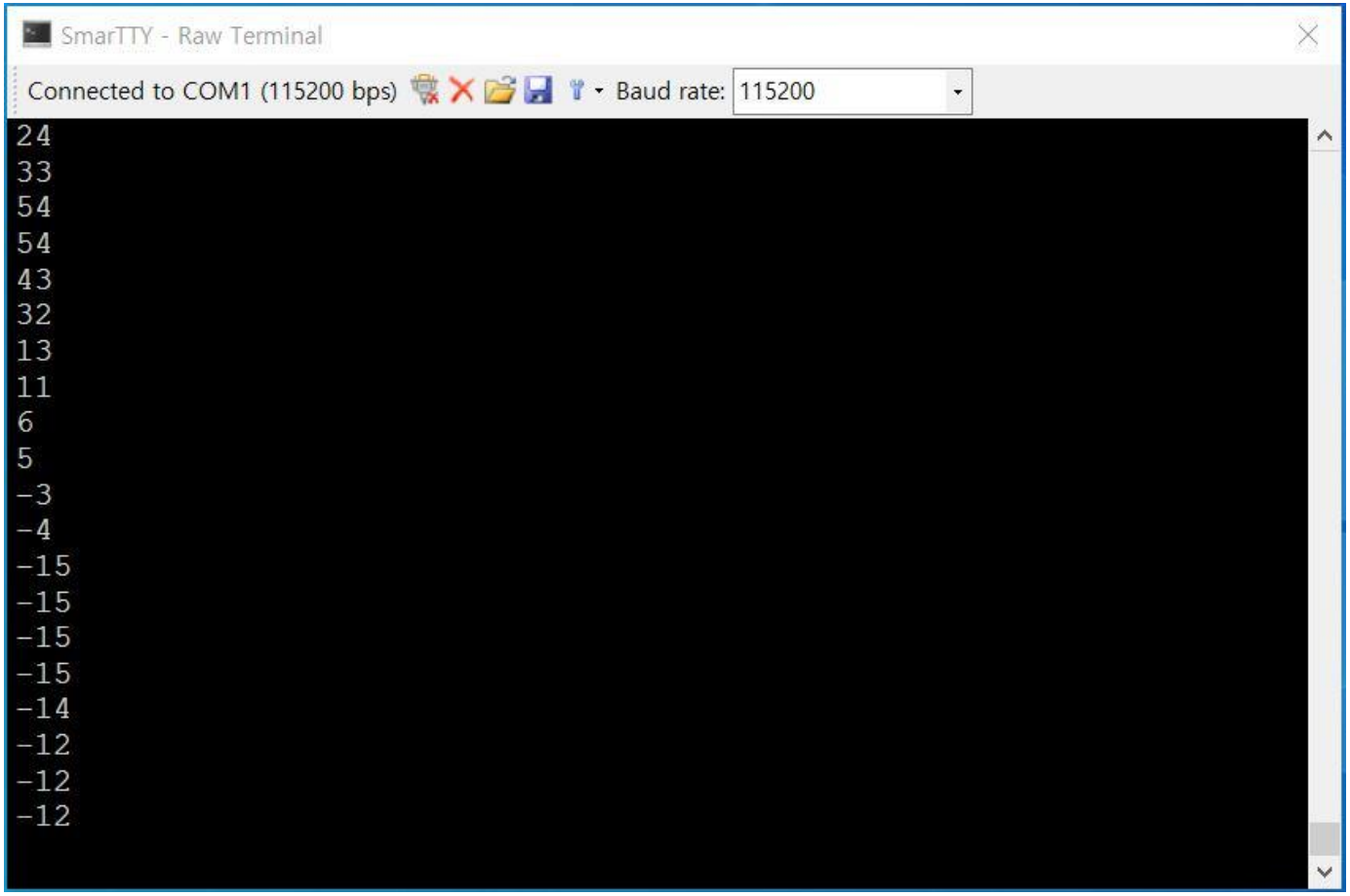
Remap PA15

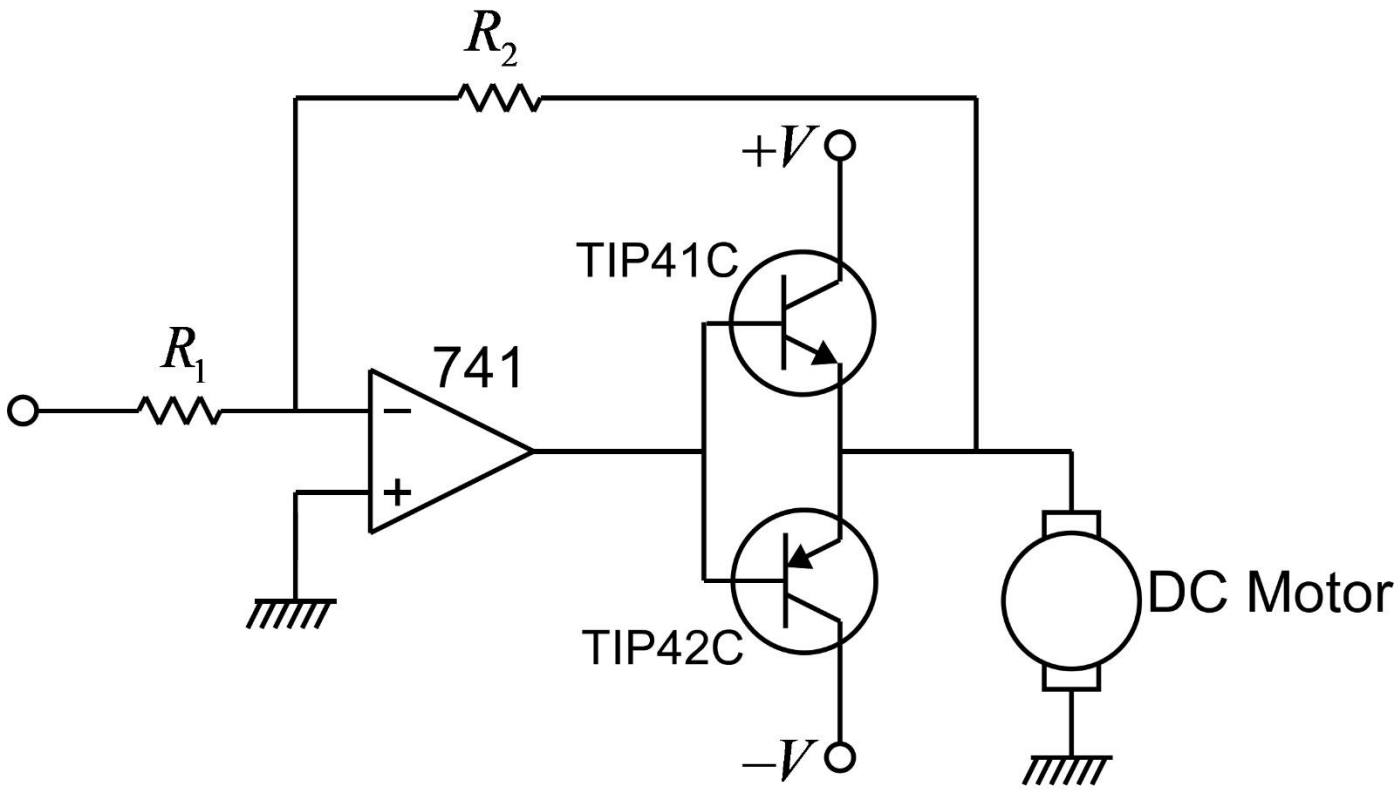

workspace - Device Configuration Tool - STM32CubeIDE
File Edit Navigate Search Project Run Window Help
Project Explorer
ADDAconversion
Encoder
Includes
Core
Drivers
Middlewares
USB_HOST
Encoder.ioc
STM32F429ZITX_FLASH.ld
STM32F429ZITX_RAM.ld
Hello
Simulation
Timer10
*Encoder.ioc
Pinout & Configuration
Clock Configuration
Project Manager
Additional Software
Pinout
TIM2 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Disable
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Encoder Mode
Configuration
Reset Configuration
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
User Constants
Search Signals
Show only Modified Pins
Pin ... Signal on Pin GPIO o... GPIO ... GPIO P... Maxim... User La... Modified
PA15 TIM2_CH1 n/a Alterna... No pull-... Low
PB3 TIM2_CH2 n/a Alterna... No pull-... Low
Pinout view
PA15
Reset_State
ADC1_EXTI15
ADC2_EXTI15
ADC3_EXTI15
I2S3_WS
SPI1_NSS
SPI3_NSS
SYS_JTDI
TIM2_CH1
TIM2_ETR
GPIO_Input
GPIO_Output
GPIO_Analog
EVENTOUT
GPIO_EXTI15
SDCLK


Pinout & Configuration
Clock Configuration
Additional Software
Pino
Categories
A->Z
System Core
DMA
GPIO
IWDG
NVIC
RCC
SYS
WWDG
Analog
Timers
RTC
TIM1
TIM2
TIM3
TIM4
TIM5
TIM6
TIM7
TIM8
TIM9
TIM10
TIM11
TIM12
TIM13
TIM14
TIM2 Mode and Configuration
Mode
Slave Mode Disable
Trigger Source Disable
Clock Source Disable
Channel1 Disable
Channel2 Disable
Channel3 Disable
Channel4 Disable
Combined Channels Encoder Mode
Configuration
Reset Configuration
User Constants
NVIC Settings
DMA Settings
GPIO Settings
Parameter Settings
Configure the below parameters :
Search (Crtl+F)
Prescaler (PSC - 16 bits value) 0
Counter Mode Up
Counter Period (AutoReload Register - 32 bits value) 0xFFFFFFFF
Internal Clock Division (CKD) No Division
auto-reload preload Disable
Trigger Output (TRGO) Parameters
Master/Slave Mode (MSM bit) Disable (Trigger input effect not delayed)
Trigger Event Selection Reset (UG bit from TIMx_EGR)
Encoder
Encoder Mode Encoder Mode TI1 and TI2
___ Parameters for Channel 1 ___

```
/* USER CODE BEGIN Includes */
#include "stdio.h"
/* USER CODE END Includes */

/* USER CODE BEGIN 0 */
#ifdef __GNUC__
#define PUTCHAR_PROTOTYPE int __io_putchar(int ch)
#else
#define PUTCHAR_PROTOTYPE int fputc(int ch, FILE *f)
#endif /* __GNUC__ */
PUTCHAR_PROTOTYPE
{
HAL_UART_Transmit(&huart1, (uint8_t *)&ch, 1, 0xFFFF);
return ch;
}
/* USER CODE END 0 */
```

```
/* USER CODE BEGIN 2 */
  HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_1);
/* USER CODE END 2 */

/* USER CODE BEGIN WHILE */
  while (1)
  {
          printf("%ld\r\n",TIM2->CNT);
          HAL_Delay(100);
/* USER CODE END WHILE */
```


SmarTTY - Raw Terminal
Connected to COM1 (115200 bps)
Baud rate: 115200
24
33
54
54
43
32
13
11
6
5
-3
-4
-15
-15
-15
-15
-14
-12
-12
-12


R_2
$+V$
TIP41C
R_1
741
DC Motor
TIP42C
$-V$

- Assume $L_a = 0$

$$e_a = R_a i_a + K_b \frac{d\theta}{dt}, \tau = K_t i_a = J_a \frac{d^2\theta}{dt^2}$$

$$E_a(s) = R_a I_a(s) + K_b s\Theta(s), K_t I_a(s) = J_a s^2 \Theta(s)$$

$$\frac{\Theta(s)}{E_a(s)} = \frac{1/K_b}{s\left(\frac{R_a J_a}{K_t K_b} s + 1\right)} = \frac{1/K_b}{s\left(T_m s + 1\right)}, \quad T_m = \frac{R_a J_a}{K_t K_b}$$

Transfer Function

- Output: angle(position)

$$\frac{\Theta(s)}{E_a(s)} = \frac{1/K_b}{s\left(\frac{R_a J_a}{K_t K_b}s+1\right)} = \frac{1/K_b}{s(T_m s+1)}, \quad T_m = \frac{R_a J_a}{K_t K_b}$$

- Output: angular velocity(speed)

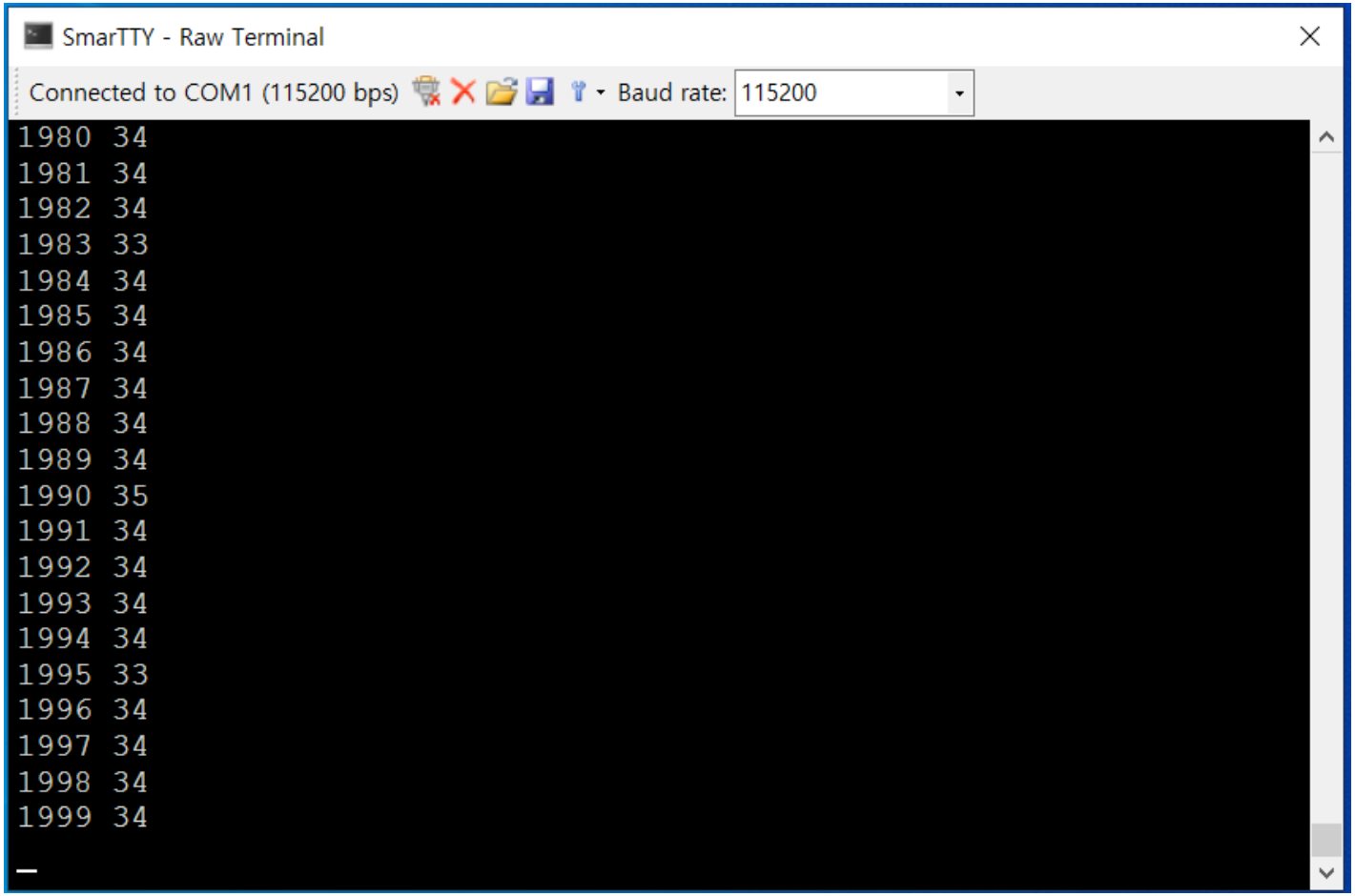
$$\frac{\Omega(s)}{E_a(s)} = \frac{s\Theta(s)}{E_a(s)} = \frac{1/K_b}{T_m s+1}$$

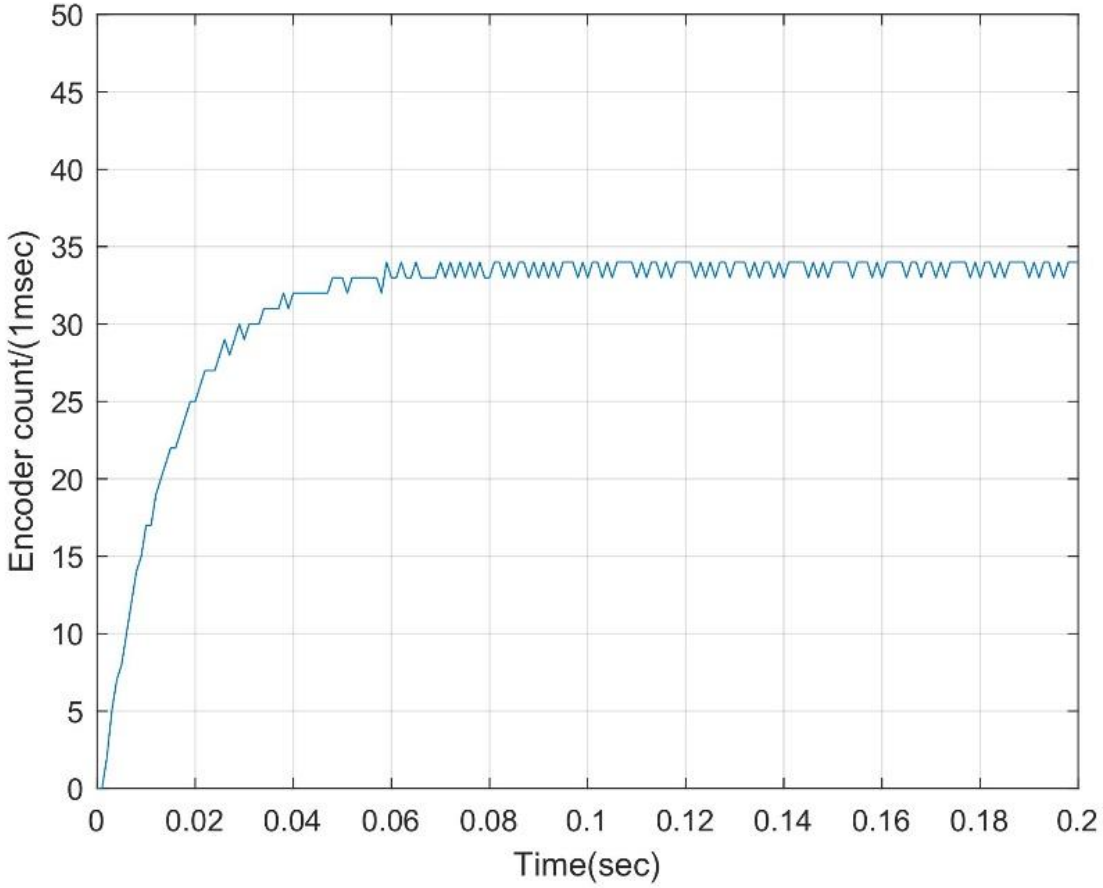
```
/* USER CODE BEGIN PV */
#define CAPTURE_START           1
#define CAPTURE_FINISHED 2
int data_flag=0;
int data_counter=0,sf=1000;
int data[4000];
/* USER CODE END PV */

/* USER CODE BEGIN 2 */
  HAL_TIM_Base_Start_IT(&htim10);
  HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_1);
  HAL_DAC_Start(&hdac, DAC_CHANNEL_2);
/* USER CODE END 2 */
```

```
/* Infinite loop */
/* USER CODE BEGIN WHILE */
while (1)
{
  if (data_flag == CAPTURE_FINISHED) {
    printf("%d %d\r\n",0,0);
        for (int i=1; i < 2*sf ;i++){
          printf("%d %d\r\n",i,data[i]-data[i-1]);
        }
        HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_RESET);
        data_flag = 0;
  }
  /* USER CODE END WHILE */
```

```
/* USER CODE BEGIN 4 */
void HAL_GPIO_EXTI_Callback(uint16_t GPIO_Pin)
{
  HAL_GPIO_WritePin(GPIOG, GPIO_PIN_14, GPIO_PIN_SET);
  data_flag = CAPTURE_START;
}
/* USER CODE END 4 */
```

```
/* USER CODE BEGIN Callback 0 */
  int x_encoder;
  if (htim->Instance == TIM10) {
    if (data_flag == CAPTURE_START) {
      HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R,
(uint32_t)(2048+1024));
      x_encoder = TIM2->CNT;
      data[data_counter++] = x_encoder;
      if (data_counter >= 2*sf) {
          data_flag = CAPTURE_FINISHED;
          data_counter = 0;
          HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R,
(uint32_t)(2048));
      }
    }
  }
/* USER CODE END Callback 0 */
```


SmarTTY - Raw Terminal
Connected to COM1 (115200 bps)
Baud rate: 115200
1980 34
1981 34
1982 34
1983 33
1984 34
1985 34
1986 34
1987 34
1988 34
1989 34
1990 35
1991 34
1992 34
1993 34
1994 34
1995 33
1996 34
1997 34
1998 34
1999 34


Encoder count/(1msec)
Time(sec)
0
5
10
15
20
25
30
35
40
45
50
0
0.02
0.04
0.06
0.08
0.1
0.12
0.14
0.16
0.18
0.2

$$\frac{34\times1000\times2\pi}{96\times4}=556(rad/\sec)$$

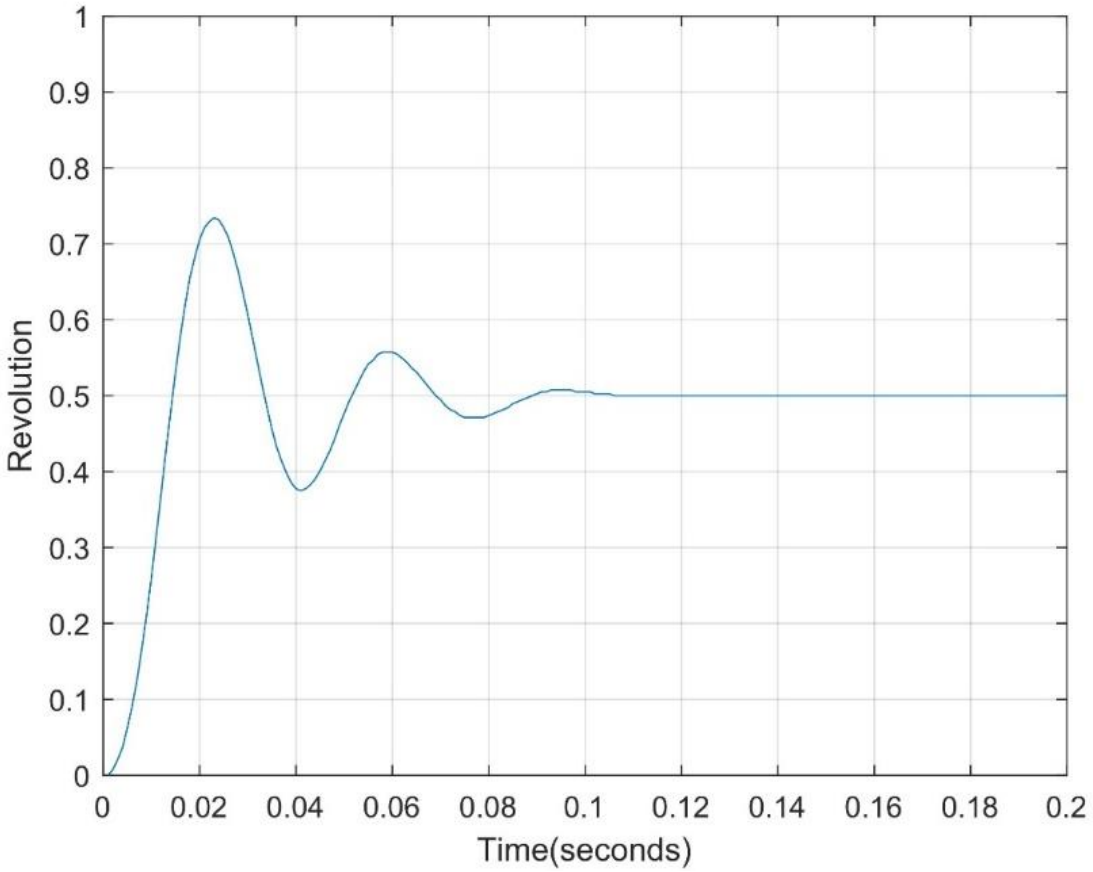
$$\frac{\Omega(s)}{E_a(s)}=\frac{s\Theta(s)}{E_a(s)}=\frac{1/K_b}{T_m s+1}=\frac{55.6}{0.015s+1}$$

```
/* USER CODE BEGIN PV */
float Kp,Kd,control;
int32_t y,oldy,ref,interrupt_counter,data_counter,sampling_frequency;
int16_t data[2000];
volatile uint8_t data_flag,data_done;
/* USER CODE END PV */

/* USER CODE BEGIN Init */
  sampling_frequency=1000;
  Kp=8.0; Kd=0.00;
/* USER CODE END Init */
```

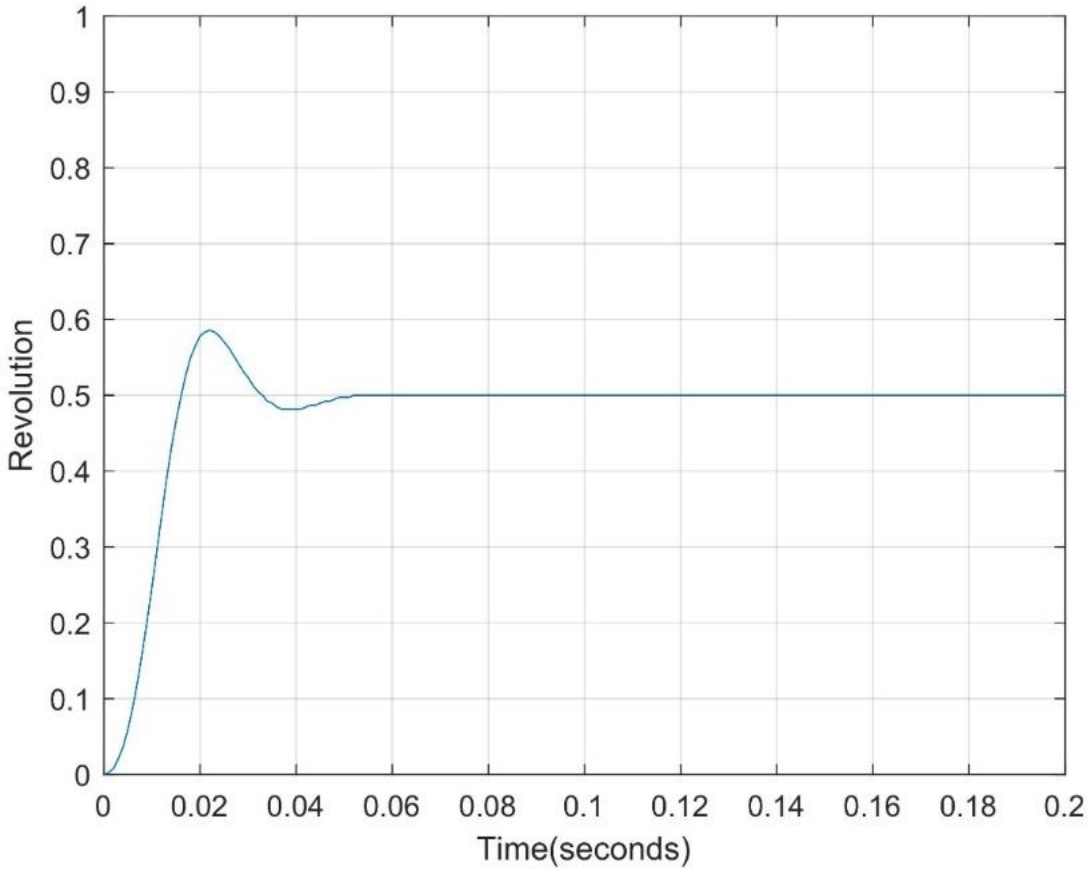
```
/* USER CODE BEGIN Callback 0 */
  int32_t da_value;
  if (htim->Instance == TIM10) {
    y = TIM2->CNT;
    interrupt_counter++;
    if (interrupt_counter >= sampling_frequency*2) {
      interrupt_counter=0;
      if (data_flag==1) {
        data_counter=0;
        data_flag=2;
        ref = 192;
      }
      else {
        ref = 0;
      }
    }
    if (interrupt_counter >= sampling_frequency*1) {
      ref=0;
    }
```

```
if (data_flag==2) {
      if (data_counter<=sampling_frequency*2) {
        data[data_counter++]=(int16_t)y;
      }
      else {
        data_done=1;
      }
    }
    control = Kp*(float)(ref-y)-(float)sampling_frequency*Kd*(float)(y-
oldy);
    oldy=y;
    if (control > 2047) control = 2047;
    if (control < -2048) control = -2048;
    da_value = control + 2048;
    HAL_DAC_SetValue(&hdac, DAC_CHANNEL_2, DAC_ALIGN_12B_R,
(uint32_t)(da_value));
  }
  /* USER CODE END Callback 0 */
```

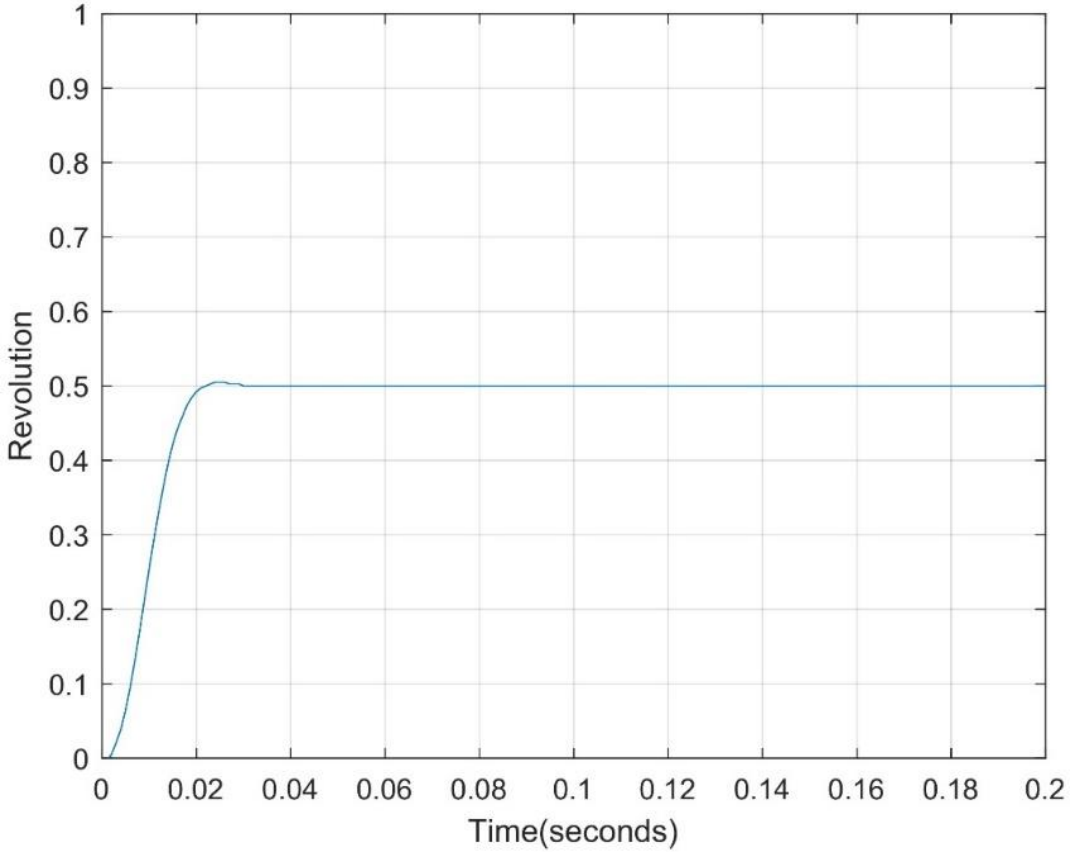
Step response for Kp=8.0 and Kd=0.0



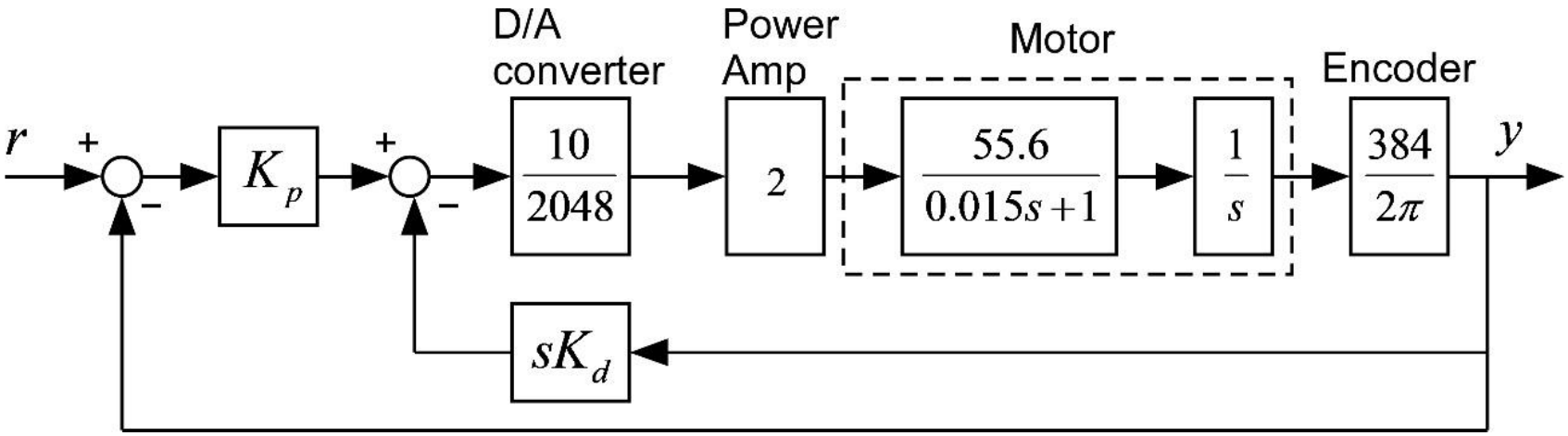
Step response for Kp=8.0 and Kd=0.02



Step response for Kp=8.0 and Kd=0.04

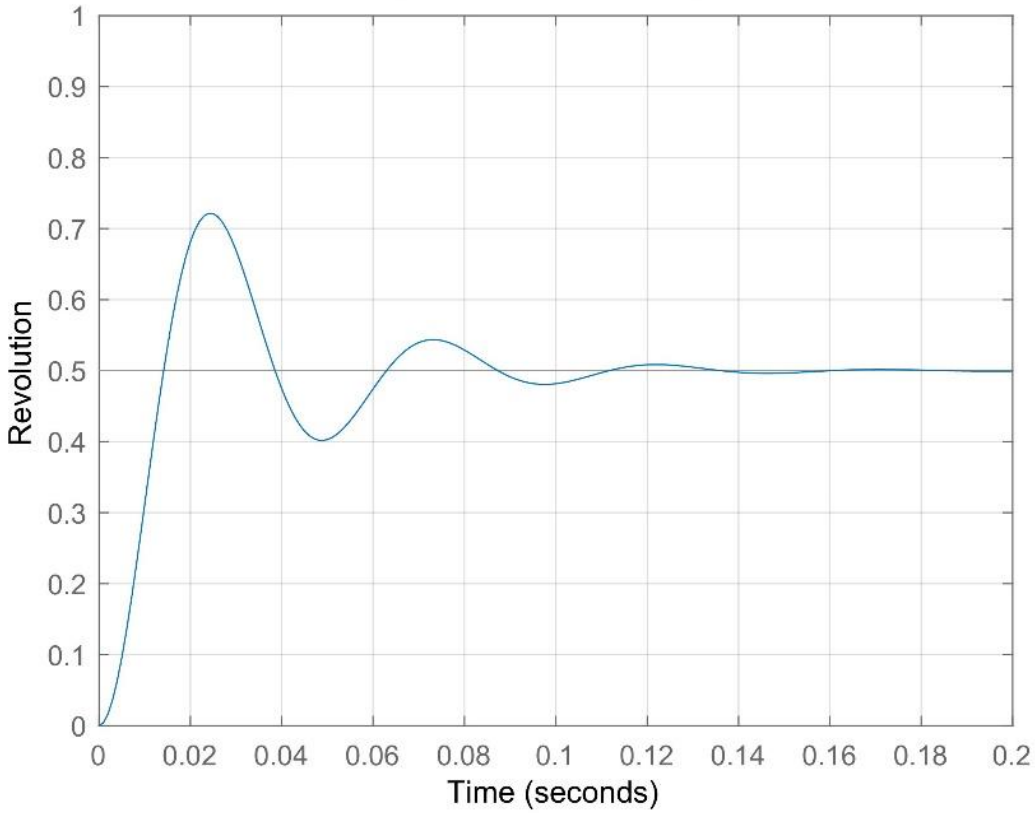


Block diagram

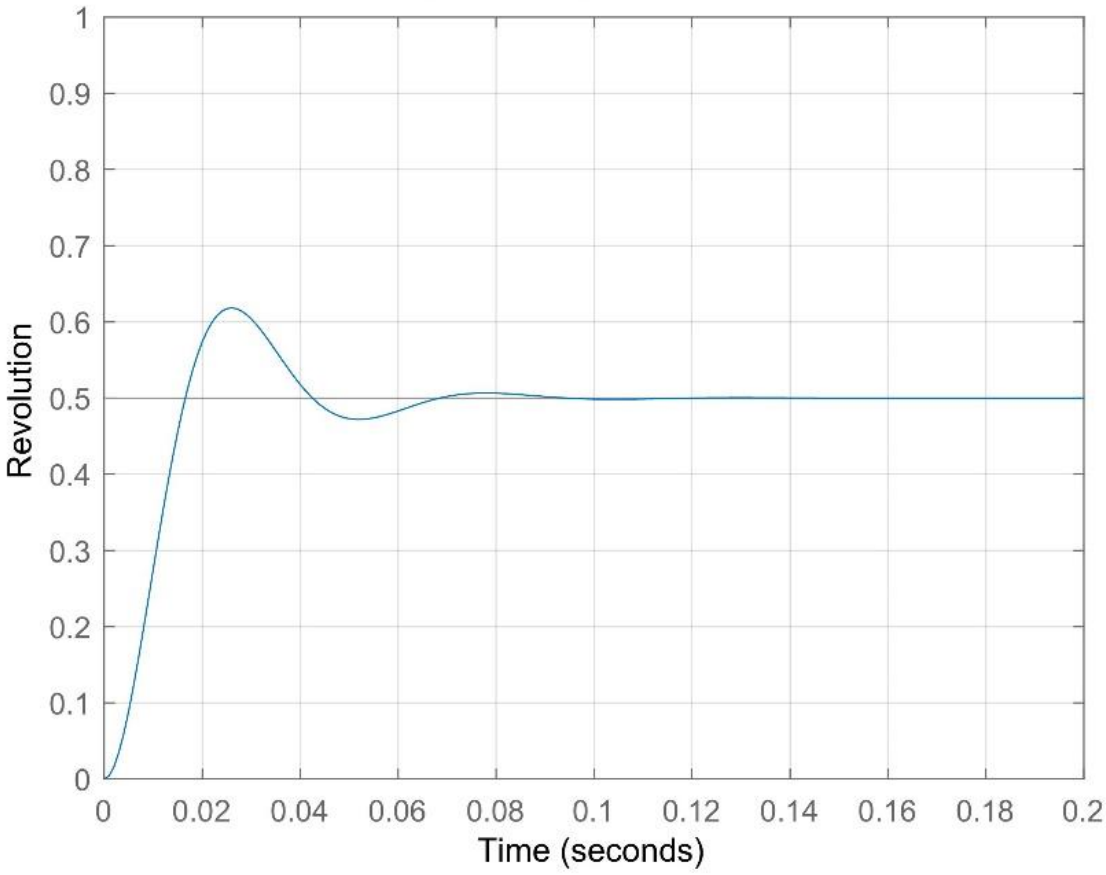


```
Kp=8;Kd=0.0;Ki=0;
D=Kp+tf([Ki],[1 0])
G=tf([2*55.6*384/(2*3.14*204.8)],[0.015 1 0])
G1=feedback(G, tf([Kd 0],[1]))
Gt=feedback(D*G1,1)
t=0:0.001:0.2;
R=0.5*ones(size(t));
figure(2)
lsim(Gt,R,t)
pole(Gt)
axis([0 .2 0 1])
xlabel('Time');
ylabel('Revolution');
grid on
```

Step response for Kp=8.0 and Kd=0.0



Step response for Kp=8.0 and Kd=0.02



Step response for Kp=8.0 and Kd=0.04