PID Controller

By: Yujie Zang 30-31/10/2024

1 Motor

1.1 Introduction

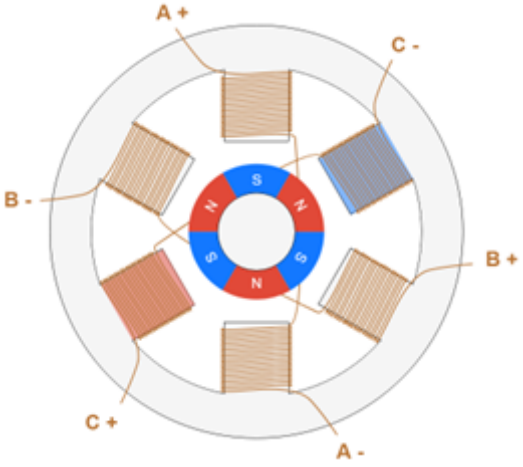
Motors are electric devices that **convert electrical energy to mechanical motion** in the form of a rotor rotating around a stationary axis. These versatile devices are a driving force in a wide range of applications. It is vital to understand the differences between different motors, as each motor impacts not only the end application, but motor driver selection.

1.2 Stepper Motor(Open-loop Control)

A stepper motor is a motor that **converts an electrical pulse signal** into a corresponding angular or linear displacement. As its name implies, stepper motors operate **in discrete steps**, where each step is a precise angle of rotation (typically about 1.8°). Stepper motors rotate a set number of times based on how many **electrical pulses** they have received. Because each rotation follows the exact angle of rotation, stepper motors are highly controllable.

Components:

1. **Rotor:** The rotor (connected to the shaft) is the rotating component in a stepper motor. The rotor has **teeth or magnetic poles** that produce motion when they interact with the stator.
2. **Stator:** The stator is the stationary segment of the motor. The stator has coils of wire that can produce magnetic fields; these coils are organized into groups called phases.
3. **Winding phases:** Stepper motors can be either **bipolar or unipolar**, with bipolar stepper motors having two winding phases, and unipolar stepper motors having four winding phases. Each phase is associated with one winding on the stator.
4. **Pulses and control:** A sequence of **electrical pulses must be sent to the winding phases** to rotate the stepper motor, with the order and timing of these pulses being used to determine the direction and distance of each step.



Stepper motors are highly precise because they move in discrete steps. In addition, they were designed to maximize their holding torque, which requires them to maintain a maximum current and makes them ideal for position-holding tasks, such as robotics and camera gimbals.

Open-Loop Control:

Here we need to use the output comparison function of one channel of a timer for the pulse output.

1. **Speed:** Let's first look at how to change the speed of the stepper motor, the speed of the stepper motor depends on the ARR value of the timer, the larger the ARR value, the faster the speed, and vice versa.
2. **Angle:** Angle control is achieved by controlling the number of pulses, how to control the number of pulses, that is, record the number of timer update interrupt, an interrupt is equivalent to generate a pulse, a pulse control stepper motor to turn a step Angle.
3. **Steering:** stepper motor steering is achieved by operating the IO port of the MCU high and low level, direction pin set high level, forward rotation; Low, reverse.

```
// Define pins
const int stepPin = 3;    // Pin to send step pulses
const int dirPin = 4;     // Pin to set direction

// Variables for control
volatile int pulseCount = 0; // Count of pulses
int stepsPerRevolution = 200; // Number of steps for a full revolution (depends
on motor specs)
int speed = 1000;            // Timer ARR value for speed control (lower value =
higher speed)

// Function to set up the timer
void setupTimer() {
    // Set timer to generate interrupts at desired speed
    Timer1.initialize(speed); // Set timer period (in microseconds)
    Timer1.attachInterrupt(timerISR); // Attach the interrupt service routine
}

// Interrupt service routine for timer
void timerISR() {
    digitalWrite(stepPin, HIGH); // Generate a pulse
    delayMicroseconds(1);         // Pulse duration
    digitalWrite(stepPin, LOW);   // End pulse
    pulseCount++;                 // Increment pulse count
}
```

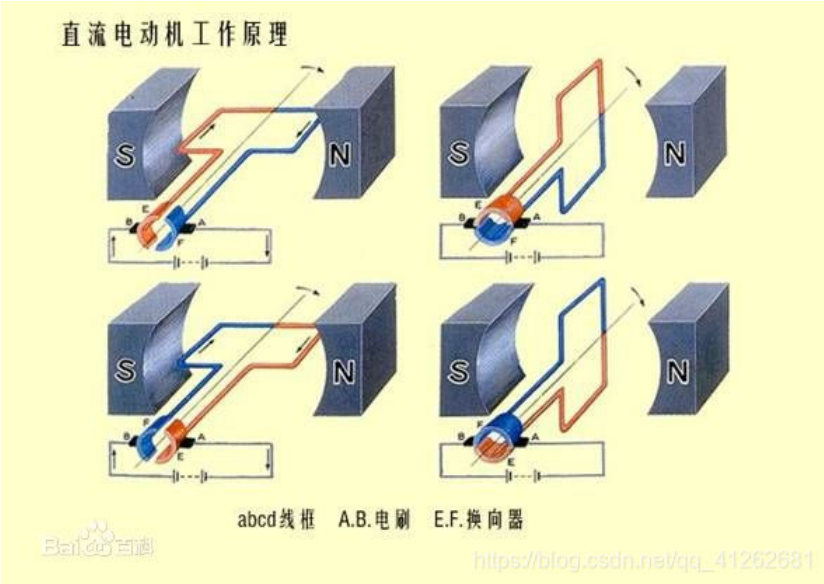
```
// Function to set motor direction
void setDirection(bool forward) {
    digitalWrite(dirPin, forward ? HIGH : LOW); // Set direction
}

// Function to move the motor a specified number of degrees
void moveMotor(int degrees) {
    int stepsToMove = map(degrees, 0, 360, 0, stepsPerRevolution); // Calculate
steps
    pulseCount = 0; // Reset pulse count

    // Move motor
    while (pulseCount < stepsToMove) {
        // Wait until we've sent the desired number of pulses
    }
}

void setup() {
    pinMode(stepPin, OUTPUT);
    pinMode(dirPin, OUTPUT);
    setDirection(true); // Set initial direction (forward)
    setupTimer(); // Set up the timer
}

void loop() {
    // Example usage: Move motor 90 degrees
    moveMotor(90);
    delay(1000); // Wait for a second before next movement
    setDirection(false); // Change direction
    moveMotor(90); // Move motor 90 degrees in reverse
    delay(1000);
}
```

Application:

Stepper motors are often used in applications that can function effectively with open-loop control, such as **3D printers, small CNC machines, and camera platforms.** They can be controlled directly by **sending pulse signals without the need for feedback**. Their **inherent ability to move in defined steps** makes them popular in positioning systems, like in **medical devices and scanning equipment**. Due to their simpler control systems and generally **lower cost**, stepper motors are favored in **budget-sensitive** applications that still require reasonable precision, such as **hobbyist robotics and educational projection**.

1.3 DC Motors: Brushed DC and Brushless DC Motors

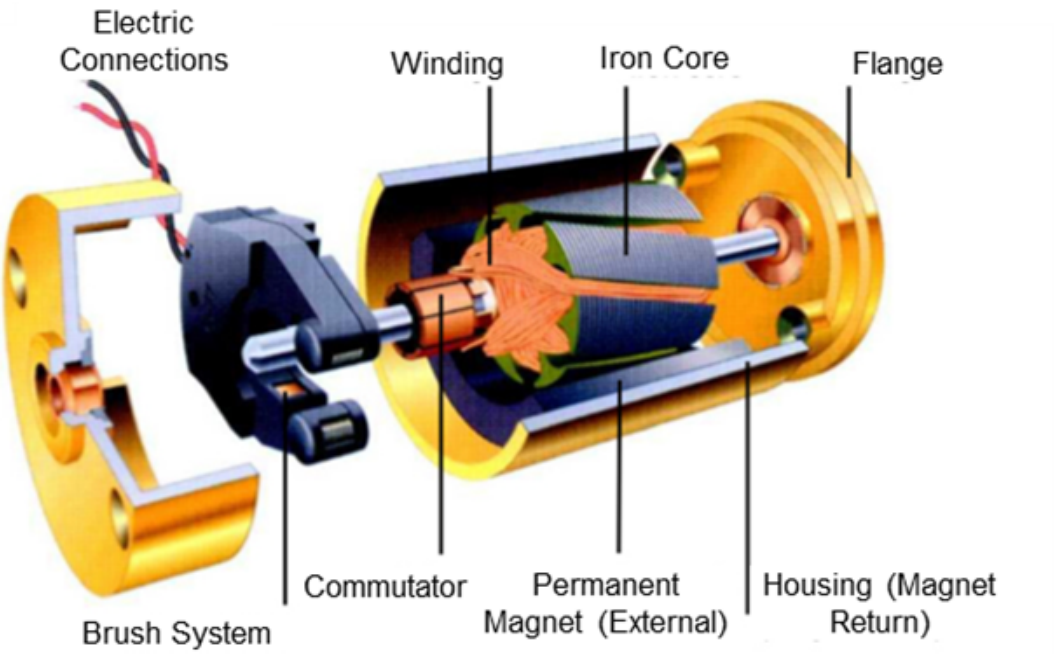
These motors generate a magnetic field between the rotating and stationary components; this magnetic field moves the rotor, which then rotates the motor. There are two main types of DC motors: brushed DC motors, and brushless DC (BLDC) motors.



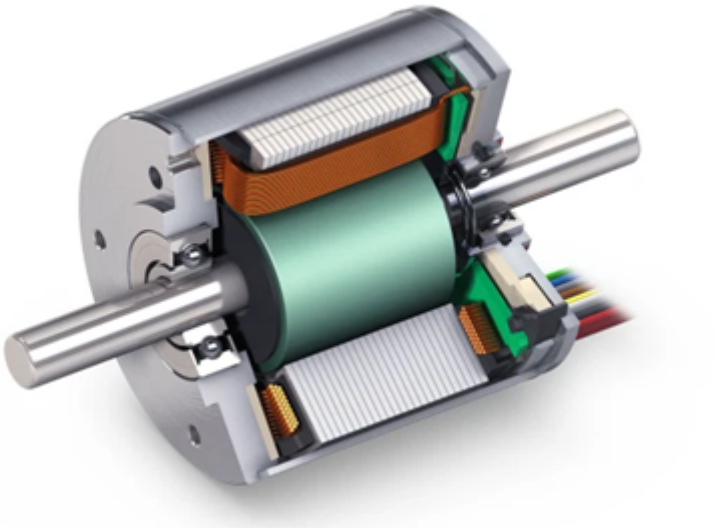
Component:

1. **Rotor:** The rotor (or armature) is a coil of wire wound around a core. Like a stepper motor's rotor, it is the rotating component of a brushed DC motor, and it is connected to the stator.
2. **Stator:** The stator is stationary and consists of one (or more) **permanent magnets or electromagnets**. **The stator produces a magnetic field that interacts with the rotor's magnetic field**, which creates torque while rotating the rotor.
3. **Commutator:** The commutator is a ring mounted on the rotor's shaft. The commutator is electrically connected to the windings on the rotor. The commutator can reverse the direction of current flow in the rotor's windings, which enables the motor to move.
4. **Brushes:** Brushes are stationary blocks made out of carbon or graphite that brush against the commutator. They conduct an electrical current that allows the motor to operate.

Brushed:



Brushless: Makes Magnet move BLDC motors do not have brushes, which typically makes them more reliable and longer lasting due to less wear and tear. BLDC motors have a stator with multiple coils, and permanent magnets or electromagnets on the rotor. They use electronic commutation and utilize a controller to control the current in the stator windings.



1.4 Comparing Stepper Motor Drivers and DC Motors

1.4.1 Operation/Controllability

Stepper motors can operate within an open-loop system, which means that the motor's precise position is determined by the exact number of steps or pulses sent to the motor. Because they operate in discrete, easily quantified steps, stepper motors do not need position control. However, stepper motors do require an external device, such as a microcontroller (MCU), to adjust the motor's speed and direction.

Brushed DC motors are powered by a DC power supply that is connected to the rotor via carbon brushes. Simple brushed DC motors can be controlled via an open-loop system, but more advanced motors may require feedback mechanisms. Typically, these motors do not require external controllers, and they can be easily adjusted. For example, adjusting the motor's voltage changes its speed.

BLDC motors must operate in closed-loop systems, which provide high precision but require additional control circuitry for smooth operation.

1.4.2 Efficiency and Noise

Stepper motors tend to be less efficient because they lose energy through heat dissipation; in addition, stepper motors operate at their maximum current at all times, which means they demand high amounts of energy. DC motors are more efficient, with brushless DC motors being the most efficient because they do not lose as much energy through friction from the brushes.

In terms of noise generation, stepper motors produce the most noise because of their discrete steps, which results in a whirring or ratcheting sound when the motor is rotating at a continuous speed. Brushed DC motors are less noisy, but when the brushes brush over the commutator, this still generates noise; as expected, BLDC motors produce the least amount of noise.

1.4.3 Summary

Table 1: Stepper Motors vs. Brushed DC Motors

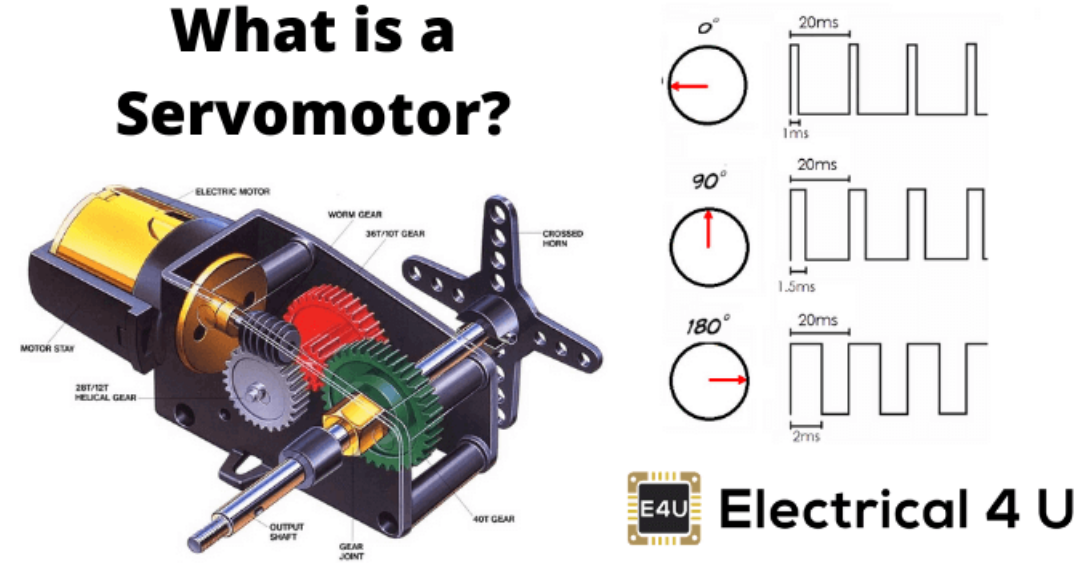
Motor	Advantages	Disadvantages	Applications
Stepper Motor	• High accuracy • High precision • Easy to control • Long lifespan (10,000 hours)	• Less efficient • Requires external control (microcontroller) • Noisy	• 3D printers • Telescope • Disk drives • Robotics
Brushed DC Motor	• Moderate efficiency • Faster response time • Can detect overload conditions	• Shorter lifespan; require maintenance to ensure reliability • Complex control	• Electric tools/appliances • Automotive (e.g. windshield wipers) • Toys • Fans
BLDC Motor	• High efficiency • Requires little maintenance • Quiet • Very long lifespan (10,000+ hours)	• Complex control • Susceptible to extreme temperatures	• Electric vehicles • Household appliances • Medical devices (e.g. infusion pumps, imaging)

1.5 Servo Motor(Close-loop)

Servo Motor Definition: A servo motor is defined as an electric motor that provides precise control of angular or linear position, speed, and torque using a feedback loop system.

Control Systems: The servo motor utilizes advanced control systems like PID and fuzzy logic to adjust movement according to input and feedback signals for optimal performance.

Feedback Mechanism: Effective use of sensors such as potentiometers and encoders helps in precise monitoring and adjustments of motor positions, speeds, or torques.



A servo motor consists of three main components:

- **A motor:** This can be either a DC motor or an AC motor depending on the power source and the application requirements. The motor provides the mechanical power to rotate or move the output shaft.
- **A sensor:** This can be either a potentiometer, an encoder, a resolver, or another device that measures the position, speed, or torque of the output shaft and sends feedback signals to the controller.
- **A controller:** This can be either an analog or a digital circuit that compares the feedback signals from the sensor with the desired setpoint signals from an external source (such as a computer or a joystick) and generates control signals to adjust the motor's voltage or current accordingly.

viewing a servo motor as a traditional motor combined with a sensor and a controller is an accurate and helpful perspective.

Generally, there are two types of control signals that can be used to control a servo motor: analog and digital.

- **Analog control signals** are continuous voltage or current signals that vary proportionally to the desired setpoint. They are typically used for simple or low-cost servo systems that do not require high accuracy or resolution. For example, a potentiometer can be used to generate an analog control signal for a hobby servo motor.
- **Digital control signals** are discrete pulses or bits that represent the desired setpoint in a coded form. They are typically used for complex or high-performance servo systems that require high accuracy, resolution, or communication. For example, a pulse-width modulation (PWM) signal can be used to generate a digital control signal for a brushless DC servo motor.

Controller:

The controller can also implement various control algorithms to optimize the performance of the servo motor. Some of the common control algorithms are:

- **Proportional-integral-derivative (PID) control:** This is a feedback-based control algorithm that adjusts the control signal based on the proportional, integral, and derivative terms of the error signal. It is widely used for servo systems that require a fast and accurate response.
- **Fuzzy logic control:** This is a rule-based control algorithm that adjusts the control signal based on fuzzy sets and linguistic variables. It is useful for servo systems that deal with

uncertainty or nonlinearities.

- **Adaptive control:** This is a self-tuning control algorithm that adjusts the control parameters based on the changing conditions of the servo system. It is beneficial for servo systems that face disturbances or variations.

Steering Motor and PWM:

Internal Potentiometer: Internally, both servos typically use a potentiometer to provide feedback on the position of the servo horn. This potentiometer measures the angle of the output shaft and allows the servo to determine its current position relative to the command received. However, this feedback mechanism is not as precise as an encoder. The accuracy is limited by the resolution of the potentiometer.

Pulse Width Modulation (PWM):Both MG996 and SG90S servos are controlled via PWM signals. The duration of the pulse determines the desired position of the servo horn (the output arm). For example, a typical PWM signal will have a pulse width ranging from approximately 1 ms to 2 ms, which corresponds to the servo's range of motion (usually 0° to 180°).

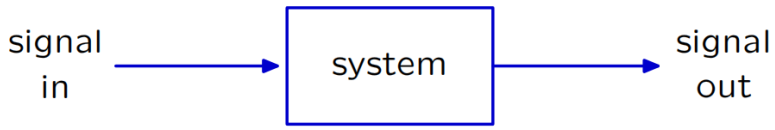
Application:

servo motors have a wide range of applications in various fields and industries, such as robotics, CNC machinery, automated manufacturing, medical equipment, etc.

2 Systems, Signals and Feedback

2.1 Introduction to systems and signals

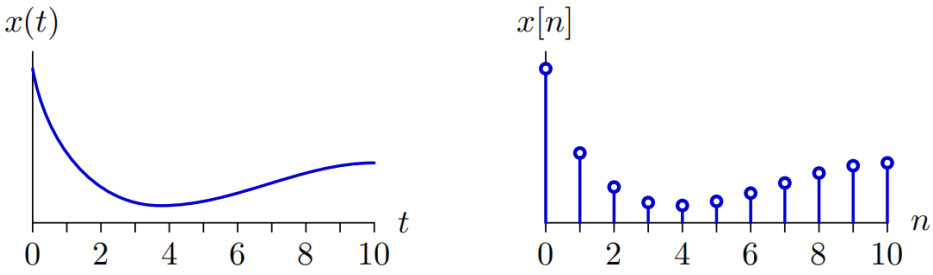
Describe a **system** (physical, mathematical, or computational) by the way it transforms an **input signal** into an **output signal**. The Signals and Systems approach has broad application: electrical, mechanical, optical, acoustic, biological, financial, ...



Continuous-Time Signals and Discrete-Time Signals:

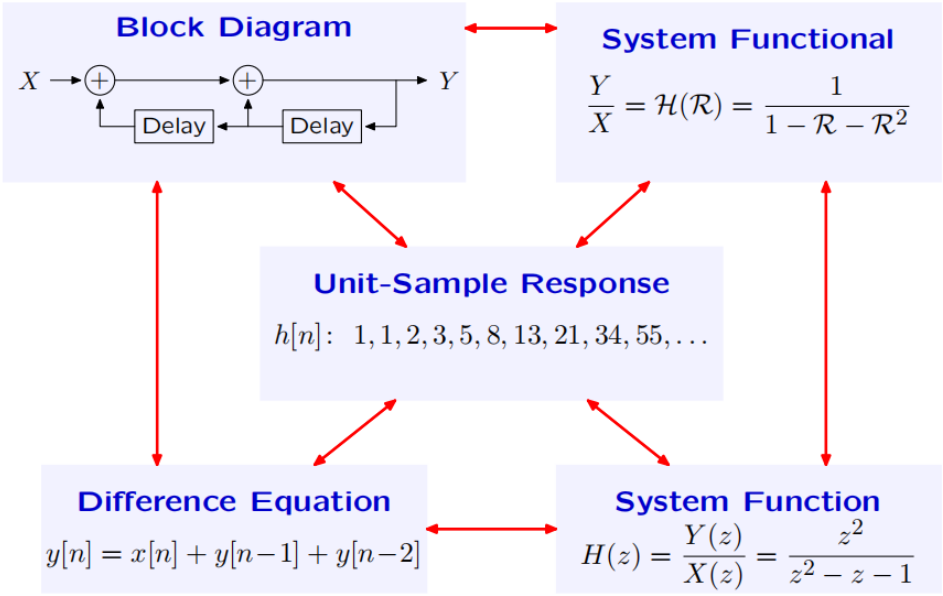
- Signals from **physical systems** often functions of **continuous** time.(mass and spring or leaky tank, for example)

- Signals from **computation systems** often functions of **discrete** time.(state machines: given the current input and current state, what is the next output and next state.)



Discrete-Time Representations and Z-Transform:

Choose one and let's work the most simply!



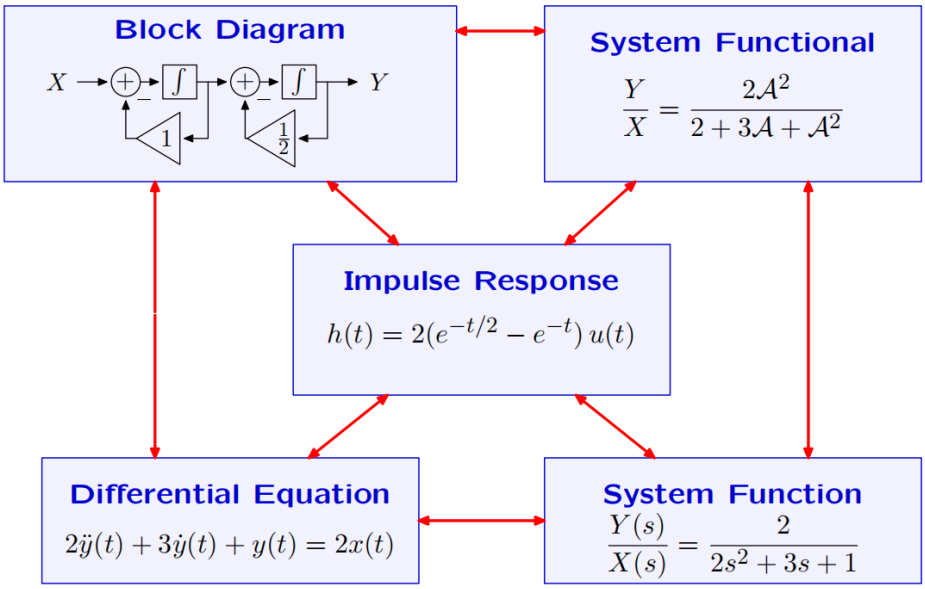
Z-Transform: It transforms a discrete-time signal, typically represented as a **sequence of numbers**, into a **complex frequency domain representation**.

$$Z\{x[n]\} = X(z) = \sum_{n=-\infty}^{\infty} x[n] z^{-n}$$

z is a complex variable, often expressed as $z = re^{j\omega}$, where r is the radius and ω is the angular frequency.

Continuous-Time Representations and Laplace-Transform:

Choose one and let's work the most simply!



Laplace-Transform: It transforms a time-domain signal f(t) into a complex frequency domain representation F(s).

$$\mathcal{L}\{f(t)\} = F(s) = \int_0^\infty f(t)e^{-st}dt$$

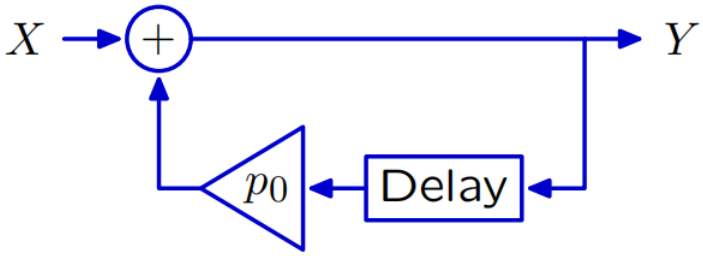
$$\text{s is a complex variable, typically expressed as } \; s = \sigma + j\,\omega$$

2.2Feedback

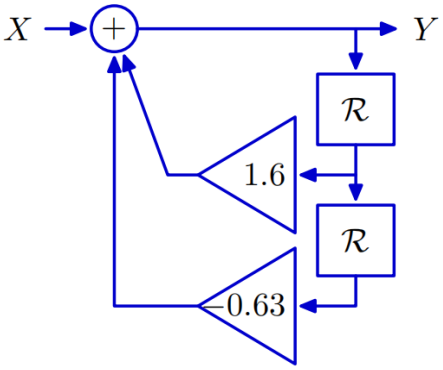
Feedback in control systems refers to the process of using the output of a system to influence its input, aiming to achieve desired performance. Feedback can be classified into two main types: **positive feedback** and **negative feedback**.

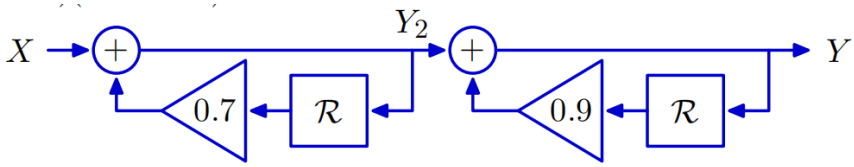
Benefits of Feedback:

- **Stability**: Helps maintain system stability by correcting deviations from desired performance.
- **Accuracy**: Improves the precision of system outputs by compensating for errors.



Second-Order Feedback:





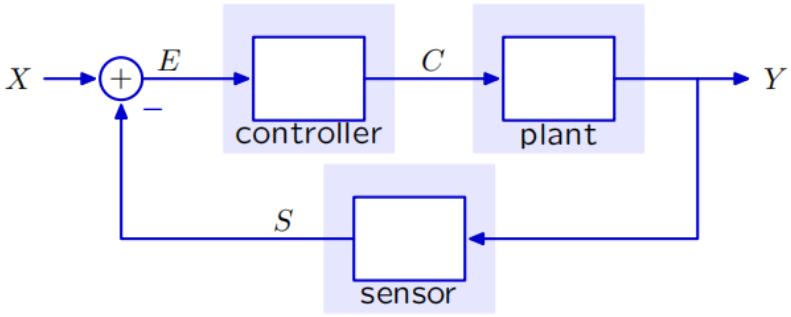
$$(1-0.7\mathcal{R})\,Y_2 = X \qquad (1-0.9\mathcal{R})\,Y = Y_2$$

Factor the operator expression to break the system into two simpler systems (divide and conquer)

Application of CT Feedback and Control:

Using feedback to enhance performance.

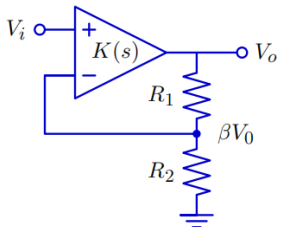
Feedback: simple, elegant, and robust framework for control.



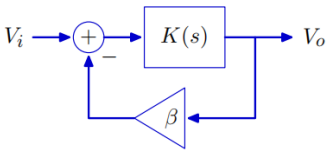
2.2.1 OP-AMPS

Problems:

1. The gain of an op-amp depends on frequency. And Low-gain at high frequencies limits applications. Unacceptable frequency response for an audio amplifier.
2. An ideal op-amp has fast time response.

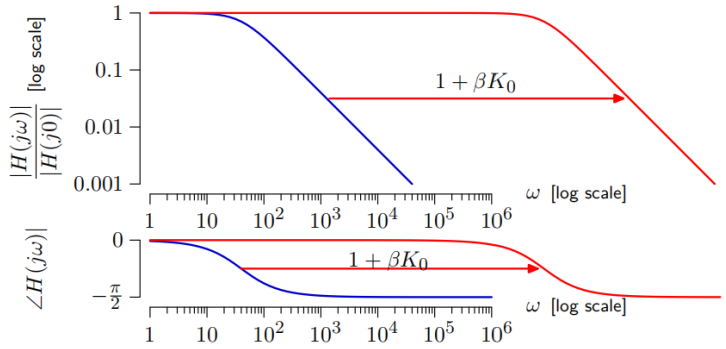


$$V_- = \beta V_o = \left(\frac{R_2}{R_1+R_2}\right) V_o$$



$$\frac{V_o}{V_i} = \frac{K(s)}{1+\beta K(s)}$$

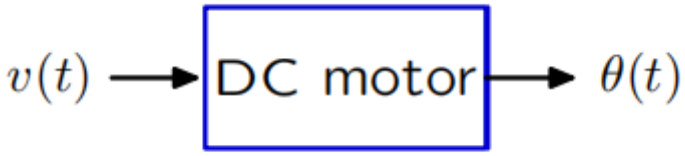
Feedback extends frequency response by a factor of $1 + \beta K_0$ ($K_0 = 2 \times 10^5$).



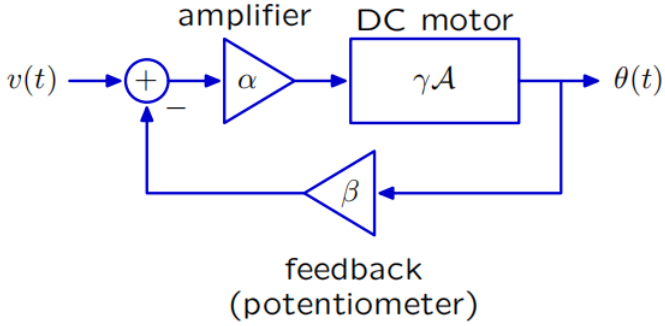
Improvement: wide bandwidth and high speed

2.2.2 DC Motor Controller

WANT: wish to build a robot arm (actually its elbow). The input should be voltage *v*(*t*), and the output should be the elbow angle *θ*(*t*).



Use proportional feedback to control the angle of the motor's shaft.

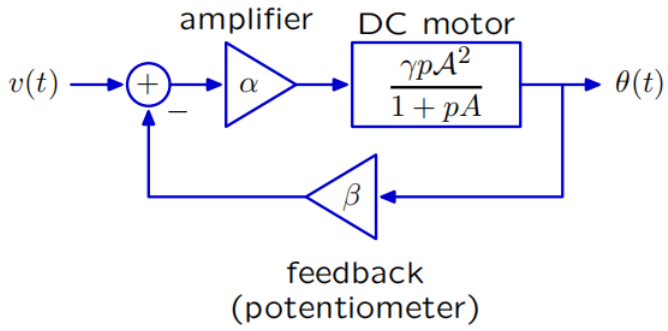


$$\frac{\Theta}{V} = \frac{\alpha\gamma\mathcal{A}}{1 + \alpha\beta\gamma\mathcal{A}} = \frac{\alpha\gamma\frac{1}{s}}{1 + \alpha\beta\gamma\frac{1}{s}} = \frac{\alpha\gamma}{s + \alpha\beta\gamma}$$

However, in real-world applications, factors like inertia, friction, and non-linearities can significantly affect the performance of the system.

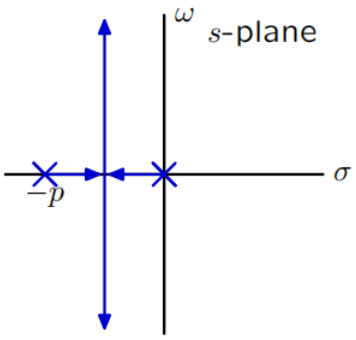
-> SO SECOND-ORDER CONTROL: For second-order model, increasing *α* causes the poles at 0 and −p to approach each other, collide at *s* = −p/2, then split into two poles with imaginary parts.

Analyze second-order model.



$$\frac{\Theta}{V} = \frac{\frac{\alpha\gamma p\mathcal{A}^2}{1+pA}}{1+\frac{\alpha\beta\gamma p\mathcal{A}^2}{1+pA}} = \frac{\alpha\gamma p\mathcal{A}^2}{1+pA+\alpha\beta\gamma p\mathcal{A}^2} = \frac{\alpha\gamma p}{s^2+ps+\alpha\beta\gamma p}$$

$$s = -\frac{p}{2} \pm \sqrt{\left(\frac{p}{2}\right)^2 - \alpha\beta\gamma p}$$

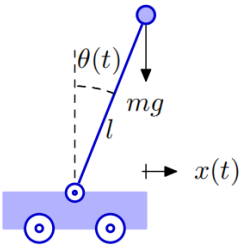


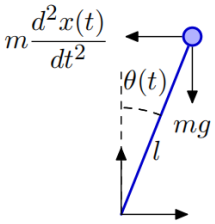
Control Strategies:

1. **PID Control**: A PID controller can be designed to handle second-order systems by **tuning the proportional, integral, and derivative gains**. This can help achieve desired response characteristics, such as settling time and overshoot.
2. **Lead and Lag Compensation**: These methods can be used to **adjust the phase margin and stability of the system**.
3. **State-Space Control**: This approach can model the dynamics of second-order systems more comprehensively, allowing for the design of **state feedback and observer-based control systems**.

2.2.3 Inverted Pendulum

Where are the poles of this system?





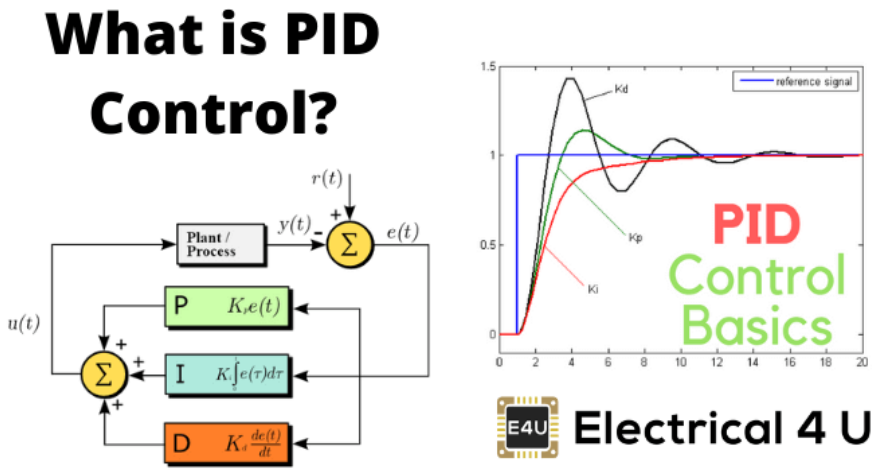
$$ml^2\frac{d^2\theta(t)}{dt^2} = mgl\sin\theta(t) - m\frac{d^2x(t)}{dt^2}l\cos\theta(t)$$

$$ml^2\frac{d^2\theta(t)}{dt^2} - mgl\theta(t) = -ml\frac{d^2x(t)}{dt^2}$$

$$H(s) = \frac{\Theta}{X} = \frac{-mls^2}{ml^2s^2 - mgl} = \frac{-s^2/l}{s^2 - g/l} \qquad \text{poles at } s = \pm\sqrt{\frac{g}{l}}$$

3 PID Controller

3.1 Introduction



PID control, representing **proportional-integral-derivative control**, is a feedback mechanism in control system, often referred to as three-term control. By adjusting three parameters—the proportional, integral, and derivative values of a process variable's deviation from its set point—specific control actions are effectively tailored.

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

Laplace-Transform:

Fig 26

The Functions of Parameters:

- **K_p**: The proportional gain determines the reaction to the current error. It produces an output that is directly proportional to the error value.
- **K_i**: The integral gain accumulates past errors over time, integrating the error signal to eliminate steady-state error.
- **K_d (just like Dampen)**: The derivative gain predicts future errors based on the rate of change of the error. It provides a damping effect by reacting to the speed of error changes.

Situations and Tuning:

- **Overshooting:** Lowering the proportional gain can help decrease the system's responsiveness, thus reducing overshoot.
- **Over-Damped:** If the derivative gain is too high, it can make the system overly sensitive to noise, potentially causing unnecessary adjustments.

3.2 Code

PID.h:

```
#ifndef PID_H
#define PID_H
class PID
{
public:
    PID(float kp, float ki, float kd, float i_max, float out_max);

    float calc(float ref, float fdb);

private:
    float kp_, ki_, kd_;
    float i_max_, out_max_;
    float output_;
    float ref_, fdb_;
    float err_, err_sum_, last_err_;
    float pout_, iout_, dout_;
};

#endif //PID_H
```

PID.cpp:

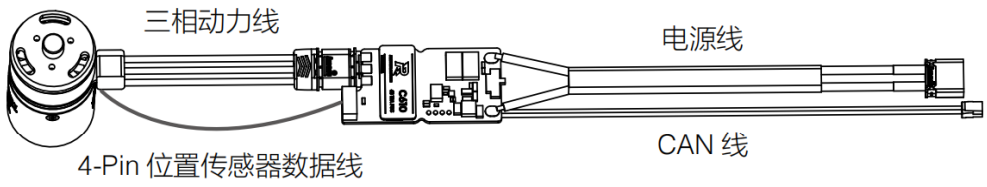
```
PID::PID(float kp, float ki, float kd, float i_max, float out_max)
    : kp_(kp), ki_(ki), kd_(kd), i_max_(i_max), out_max_(out_max),
      output_(0), ref_(0), fdb_(0), err_(0), err_sum_(0), last_err_(0),
      pout_(0), iout_(0), dout_(0) {}

float PID::calc(float ref, float fdb) {
    // update the err
    ref_ = ref;
    fdb_ = fdb;
    err_ = ref_ - fdb_;

    // calculate the pout_
    pout_ = kp_ * err_;

    // calculate the iout_
    err_sum_ += err_;
    if (err_sum_ > i_max_) err_sum_ = i_max_;
    if (err_sum_ < -i_max_) err_sum_ = -i_max_;
    iout_ = ki_ * err_sum_;

    // calculate the dout_
    dout_ = kd_ * (err_ - last_err_);
```

```
    // output the total
    output_ = pout_ + iout_ + dout_;
    if (output_ > out_max_) output_ = out_max_;
    if (output_ < -out_max_) output_ = -out_max_;

    // memorize the err
    last_err_ = err_;

    return output_;
}
```

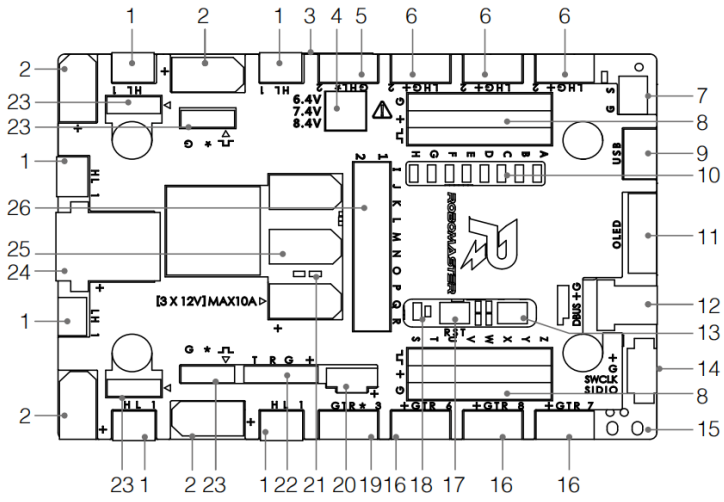
4 Practice

4.1 Equipment

Brushless DC gear motor M2006 with C610 Electronic Speed Controller:

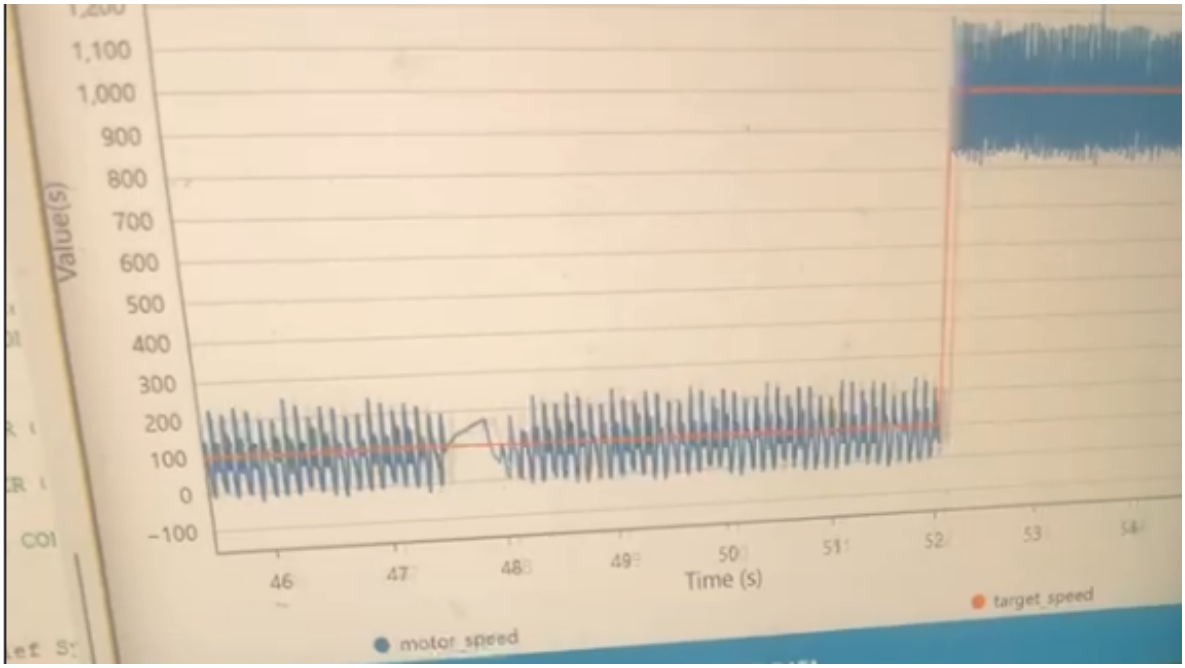


DJI A Board with STM32F427IIF6 Main Controller Chip:



4.2 One-Loop Speed Control

Videos is in the file of materials



4.3 Cascading Position and Speed Control

Videos is in the file of materials

4.4 Code

```
extern CAN_RxHeaderTypeDef RxHeader;
extern CAN_TxHeaderTypeDef TxHeader;
uint8_t RxData[8];
uint8_t TxData[8] = {0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00, 0x00};
uint32_t TxMailbox = CAN_TX_MAILBOX0;

// receive motor data
float real_position = 0;
float real_speed = 0;

// Speed limits
const float max_speed = 500.0;
const float min_speed = -500.0;


// PID control parameters for tuning
float kp_position = 0.7;
float ki_position = 0.1;
float kd_position = 0.3;
PID position_pid(kp_position, ki_position, kd_position, 20.0, 180);

float kp_speed = 0.6;
float ki_speed = 0.2;
float kd_speed = 0.2;
PID speed_pid(kp_speed, ki_speed, kd_speed, 100.0, max_speed);

float target_position = 0.0;
float target_speed = 100.0;
int16_t control_output;

float linearMapping(int in, int in_min, int in_max, float out_min, float
out_max) {
    return (out_max - out_min) / (in_max - in_min) * (in - in_min) + out_min;
}


void HAL_CAN_RxFifo0MsgPendingCallback(CAN_HandleTypeDef *hcan) {
    if (hcan->Instance == hcan1.Instance) {
        // Check if the message was successfully received
        if (HAL_CAN_GetRxMessage(hcan, CAN_RX_FIFO0, &RxHeader, RxData) ==
HAL_OK) {
            real_position = linearMapping(RxData[0] << 8 | RxData[1], 0, 8191,
-180, 180);
            real_speed = (int16_t)(RxData[2] << 8 | RxData[3]);
        }
    }
}

void HAL_TIM_PeriodElapsedCallback(TIM_HandleTypeDef *htim) {
    if (htim->Instance == htim6.Instance) {
        // PID position control
        target_speed = position_pid.calc(target_position, real_position);

        // Constrain the speed
        if (target_speed > max_speed) target_speed = max_speed;
```

```
        if (target_speed < min_speed) target_speed = min_speed;

        //  PID speed control
        control_output = static_cast<int16_t>(speed_pid.calc(target_speed,
real_speed));
        int16_t current_output = static_cast<int16_t>
(linearMapping(control_output, min_speed, max_speed, -10000, 10000));

        // Transmit the data
        TxData[0] = (uint8_t)(current_output >> 8);
        TxData[1] = (uint8_t)(current_output & 0xFF);
        HAL_CAN_AddTxMessage(&hcan1, &TxHeader, TxData, &TxMailbox);
    }
}
```

5 Appendix

5.1 Reference

1.An Introduction to Stepper Motors, DC Motors, and Motor Drivers | Article | MPS (monolithicpower.com)

2.一文掌握步进电机控制_步进电机的控制-CSDN博客

3.Servo Motor: Definition, Working Principle, and Applications | Electrical4U

4.直流电机的原理及驱动_直流电机驱动原理-CSDN博客

5.13. Continuous-Time (CT) Feedback and Control, Part 2 (youtube.com)

6.PID Controller Explained • PID Explained

7.PID Controllers in Control Systems | Electrical4U

5.2 Thanks

1.ROBOMASTER DJI

2.MIT OPEN COURSE

3.Supporters and Helpers