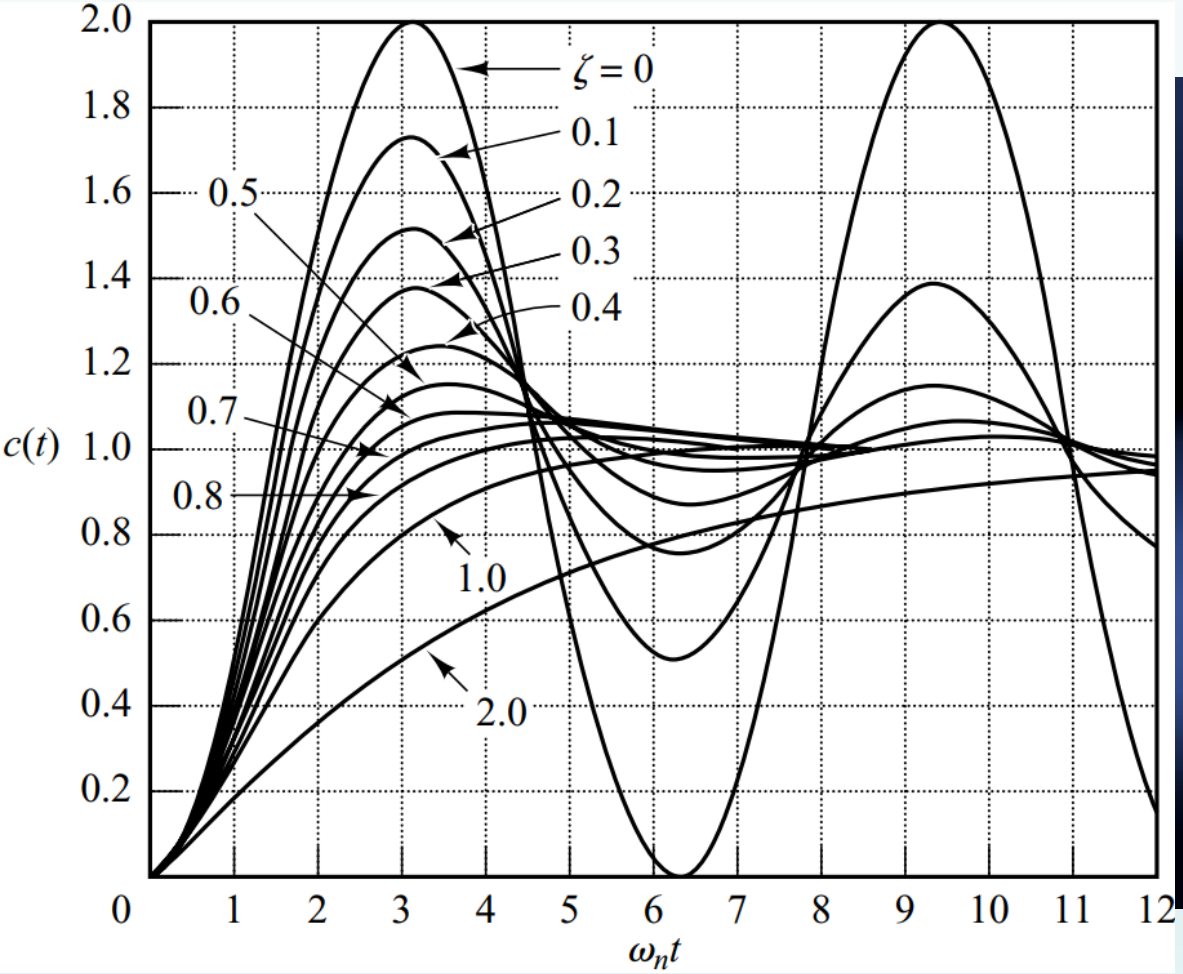



# PMR 3100 – Introdução à Engenharia Mecatrônica

## Módulo 04 – Meu Primeiro Robô

## Aula 22 – Controle PID de distância

*Prof. Dr. Rafael Traldi Moura*

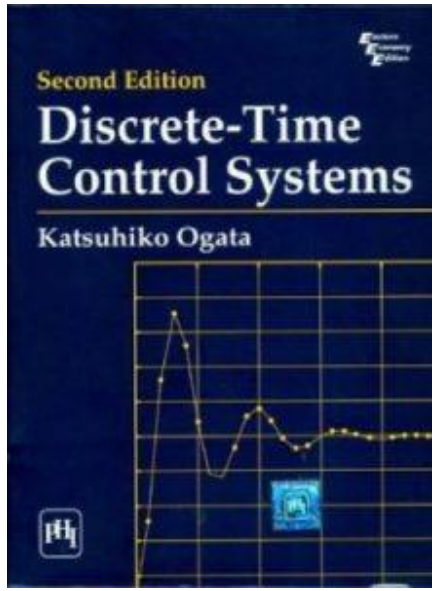
# Controle

Vocês terão um caminho longo, estudando controle em algumas disciplinas:

- PMR3302 - Sistemas Dinâmicos I para Mecatrônica;
- PMR3306 - Sistemas Dinâmicos II para Mecatrônica;
- PMR3305 - Sistemas a Eventos Discretos;
- PMR3404 - Controle I;
- PMR3409 - Controle II;

# Controle – bibliografia básica e **universal**

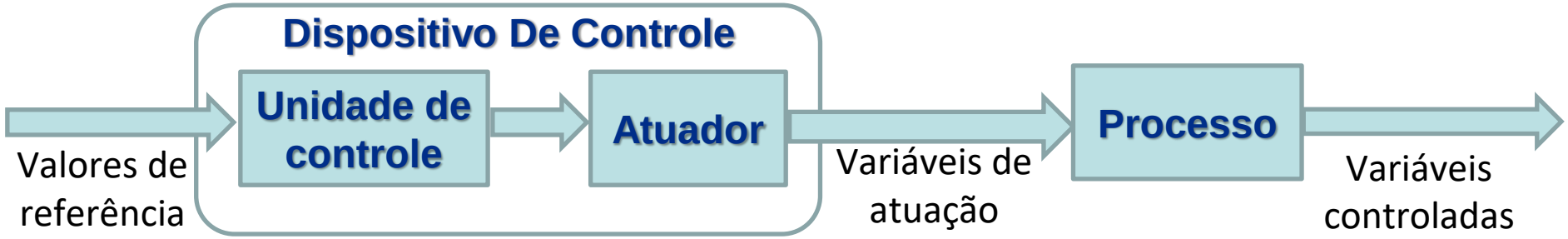

- Engenharia de Controle Moderno, Katsuhiko Ogata, Pearson Universidades; 5ª edição, 2010;



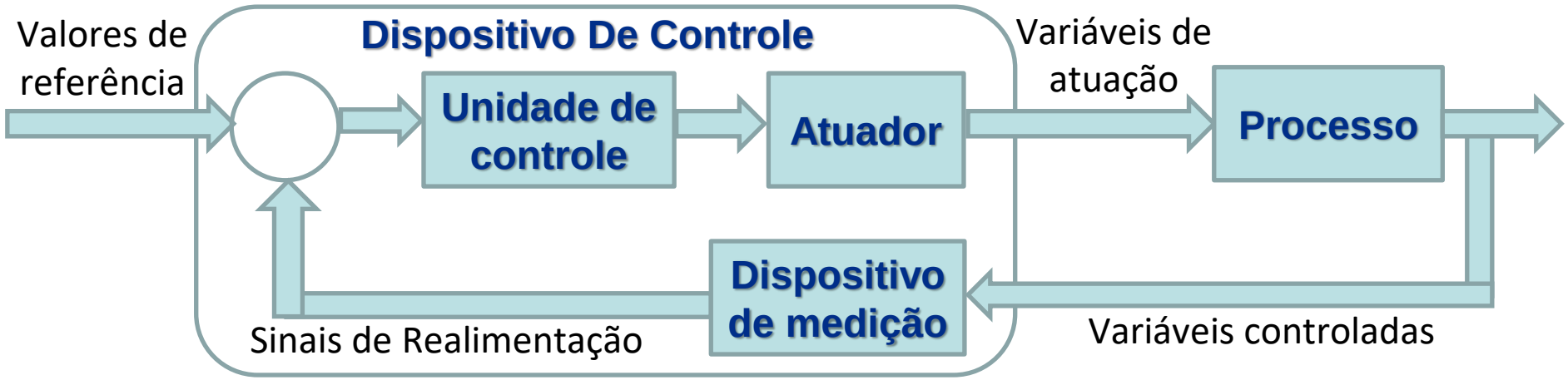

- Discrete-Time Control Systems, Katsuhiko Ogata, Pearson Universidades; 2ª edição, 1994;

# Controle – malha aberta e fechada

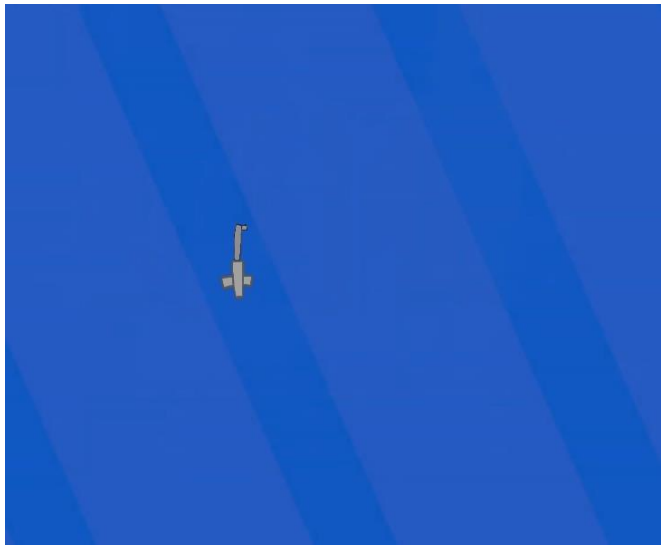
- Sistema mecatrônico com controle de malha aberta:

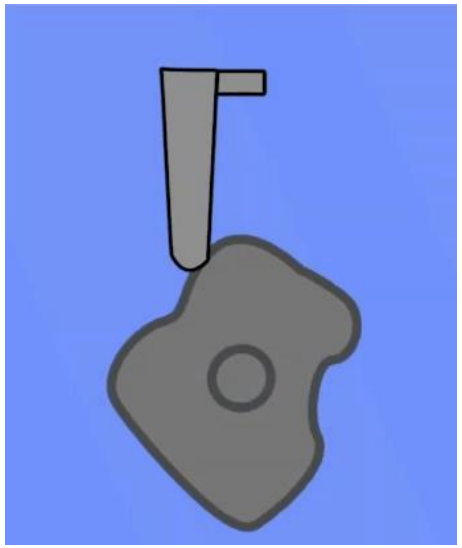


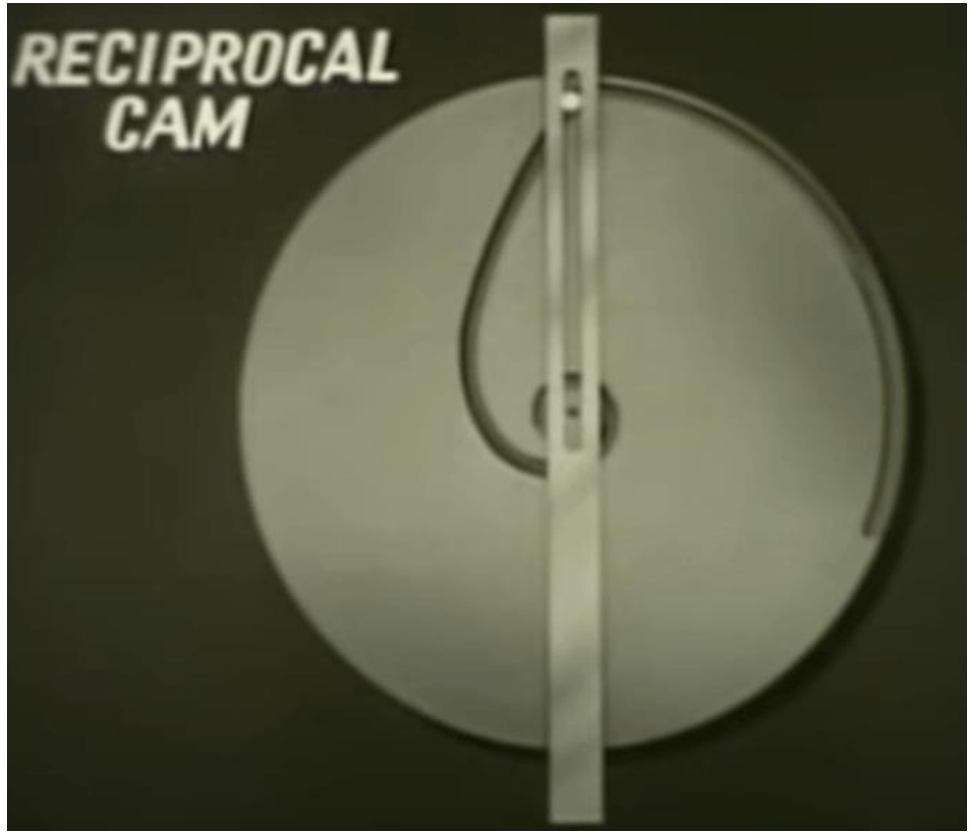

- Sistema mecatrônico com controle de malha fechada:

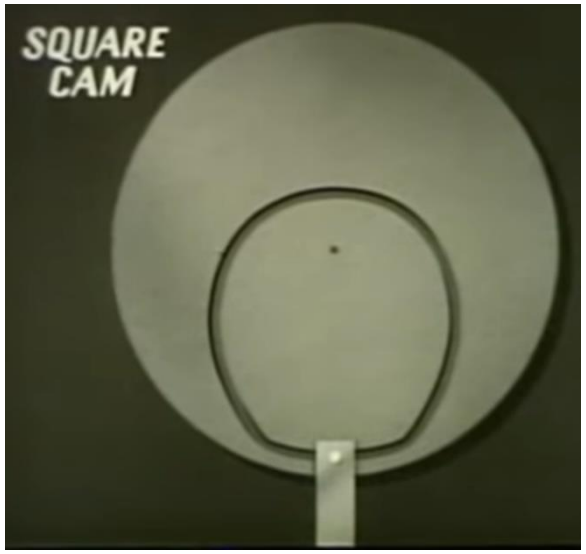


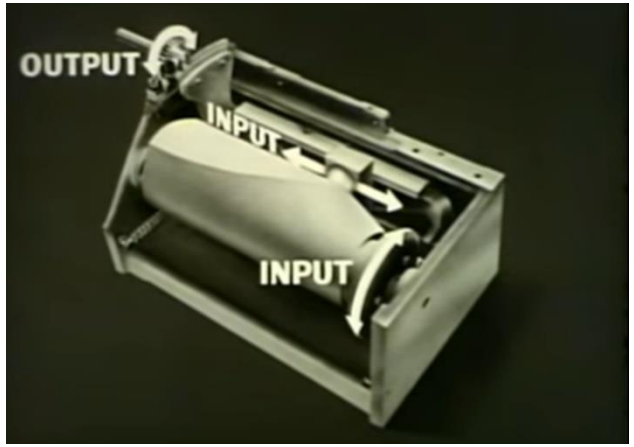
# Malha aberta – Came Seguidor







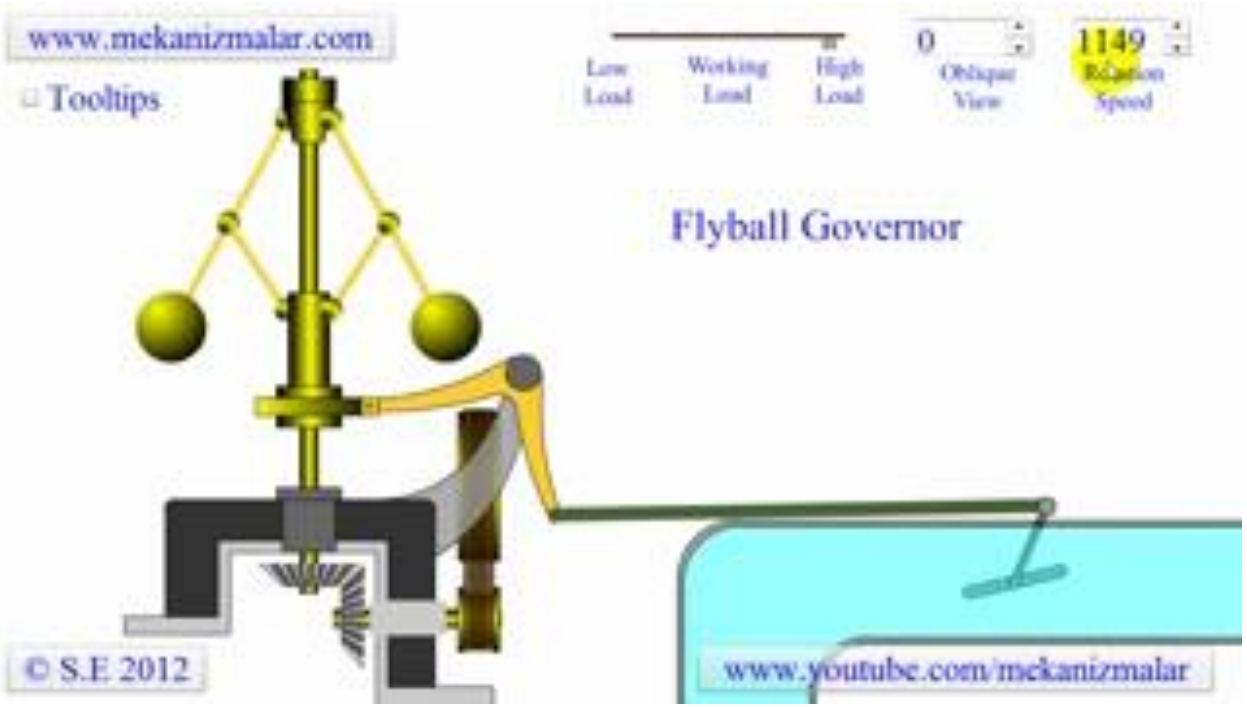

- VIDEO: COMPUTADOR MECÂNICO

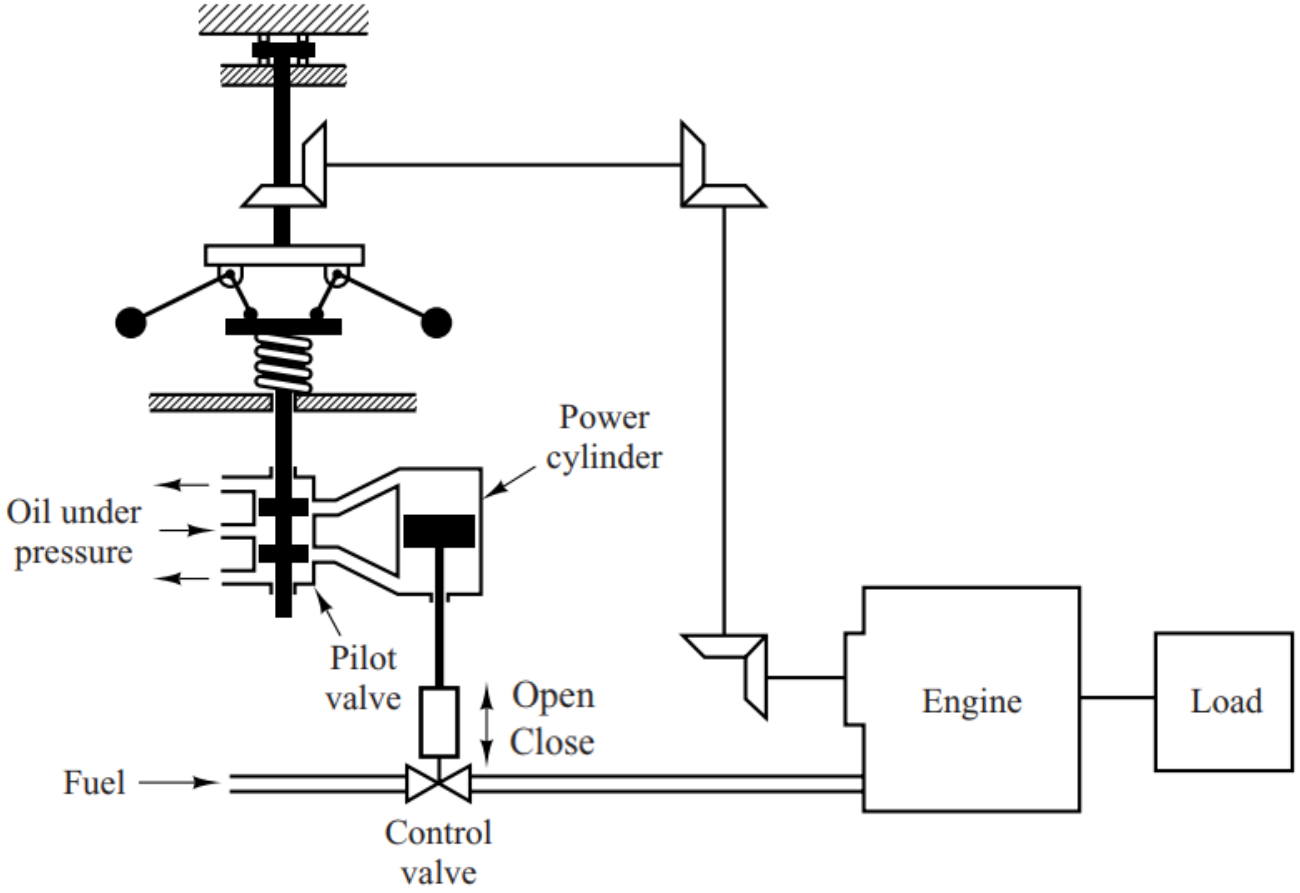
# Malha aberta – Exemplos

**Leonardo da Vinci**

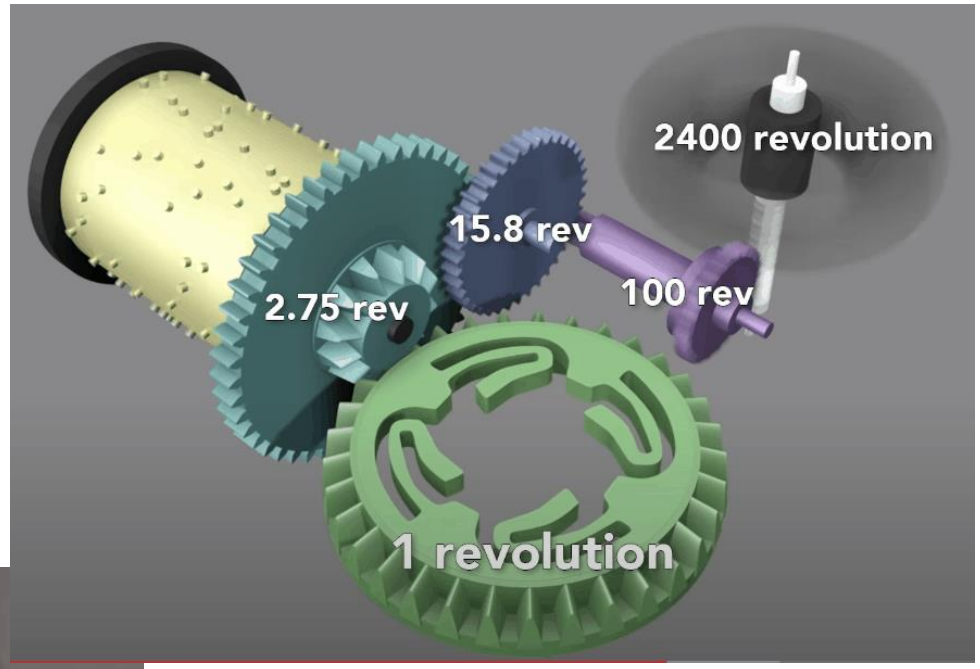
**Automato little writer (Jaquet Droz, Sec. 18)**

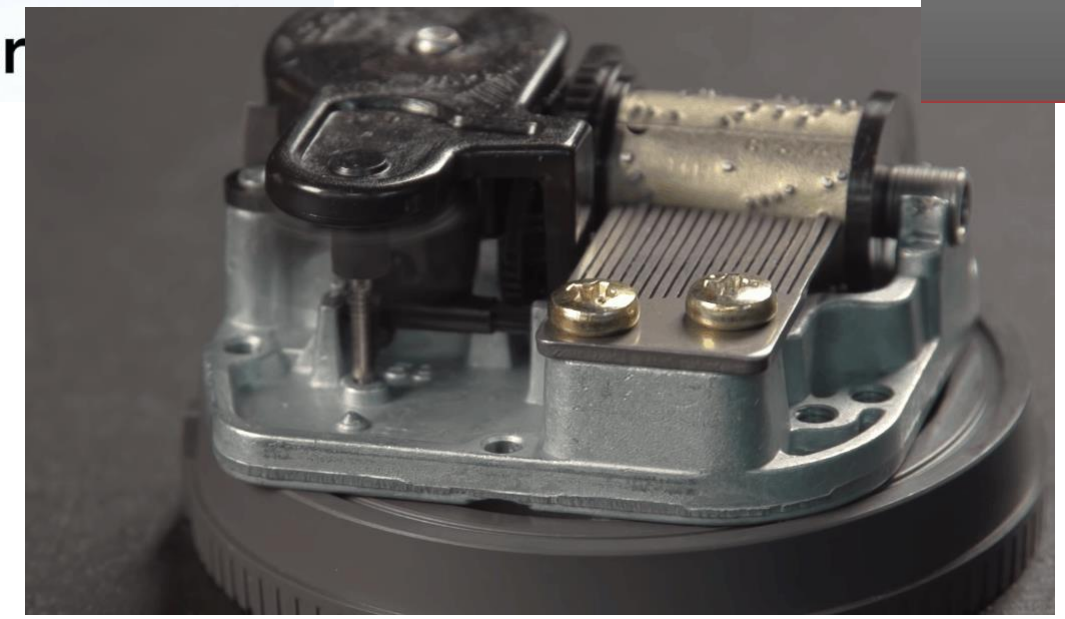
# Malha fechada – Regulador Watt de velocidade



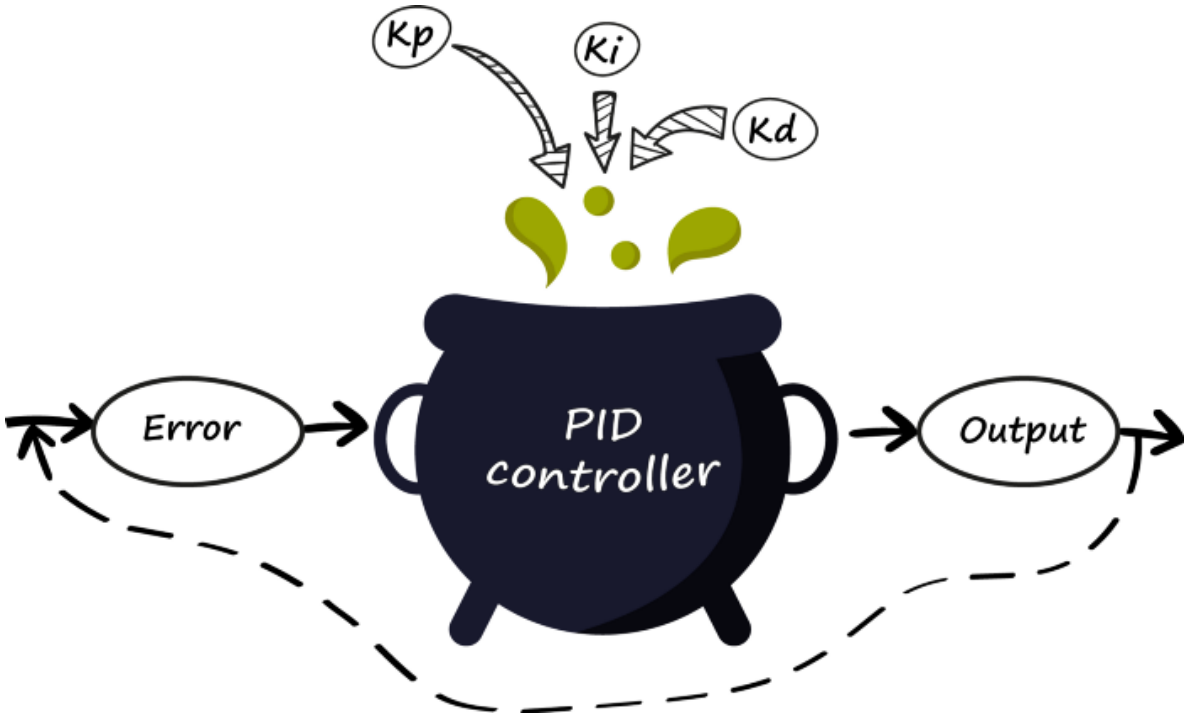
# Malha fechada – Regulador de velocidade

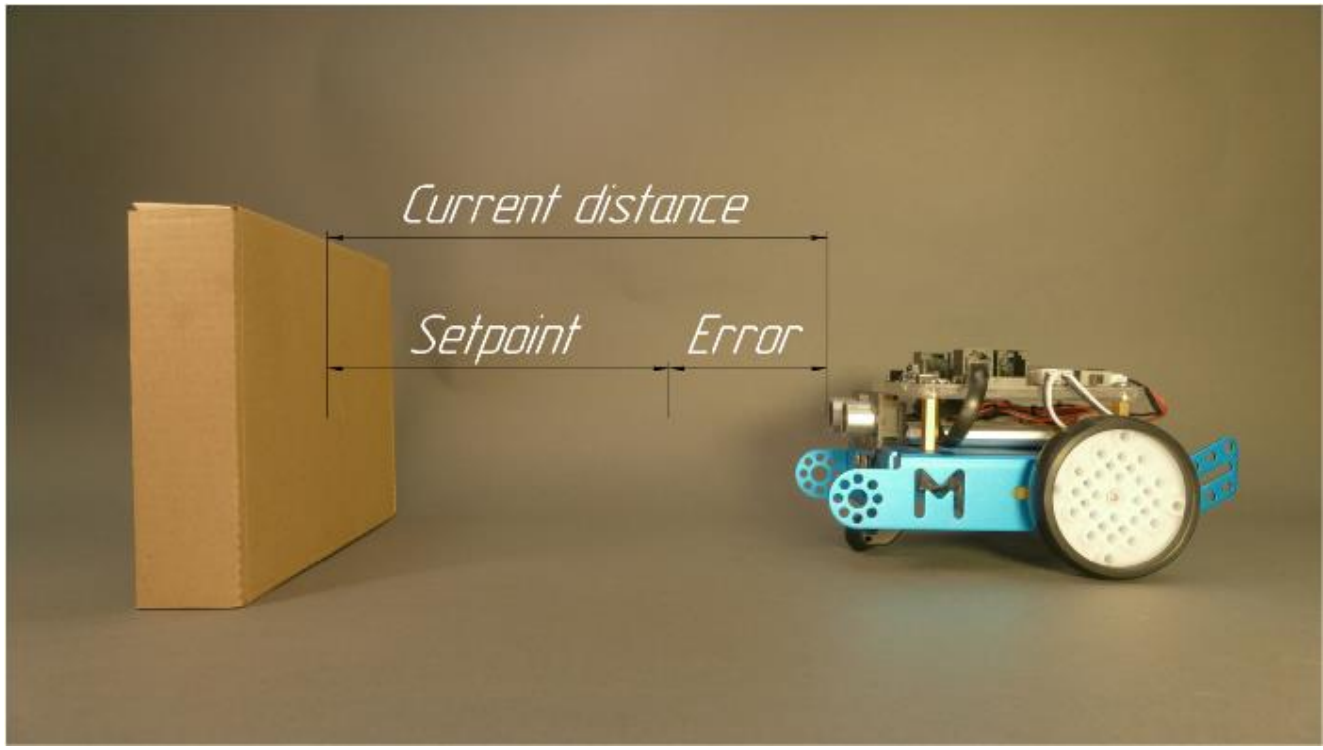



VIDEO:
Regulador de Velocidade

# Controle Proporcional Integrativo Derivativo




$$erro = x - setpoint$$

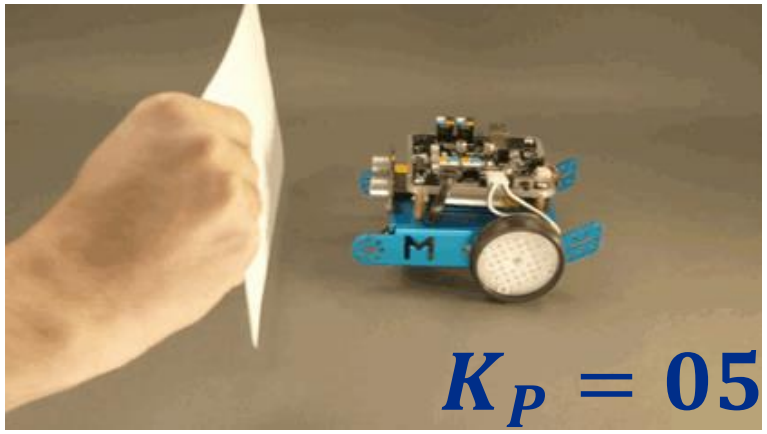
$$erro_i = \sum (x - setpoint)$$

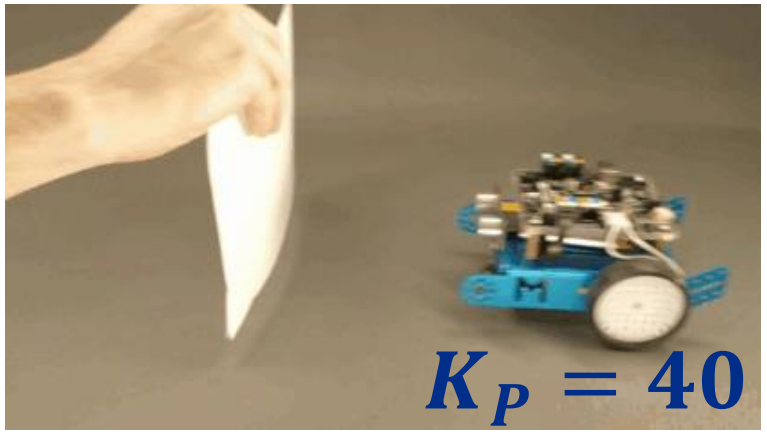
$$erro_d = \frac{\Delta erro}{\Delta t}$$

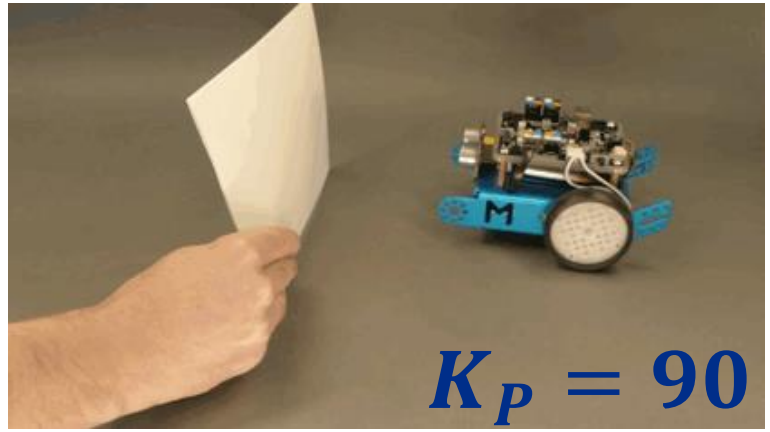
Fonte: XODlang

# Controle Proporcional

$$ação = K_P \cdot erro$$




Fonte: XODlang

# Carrinho PID de distância – códido parte 1

- A primeira parte do código contém:
  - define
  - declarações de variáveis globais

```
#define bitMotor1A 2
#define bitMotor1B 3
#define bitMotor2A 1
#define bitMotor2B 4
#define bitMotor3A 5
#define bitMotor3B 7
#define bitMotor4A 0
#define bitMotor4B 6

#define pinSH_CP  4   //Pino Clock  DIR_CLK
#define pinST_CP  12  //Pino Latch
DIR_LATCH
#define pinDS     8   //Pino Data   DIR_SER
#define pinEnable 7   //Pino Enable DIR_EN
#define pinMotor1PWM 11
#define pinMotor2PWM 3
//#define pinMotor3PWM 5
//#define pinMotor4PWM 6
```

# Carrinho PID de distância – códido parte 2

- A primeira parte do código contém:
  - define
  - declarações de variáveis globais

```
#define trigPin 10
#define echoPin 9


void ci74HC595Write(byte pino, bool estado);
void inicializa_Motor_Shield();
float le_sensor_ultrassonico();

int valorMotor_PWM;
float distancia;
float erro,integral,der,erro_anterior,PID;
float Kp,Ki,Kd;
```

# Carrinho PID de distância – códido parte 3

- A segunda parte do código contém: a função `setup()` com:
  1. Inicialização do shield controlador de motores;
  2. Inicialização e configuração do sensor ultrassônico;
  3. inicialização da comunicação serial em 9600kbps;
  4. Atribuição da configuração inicial para os motores;
  5. Inicialização das variáveis do PID;

```
void setup() {
   //inicializa e configura pinos usados
//shield de controle do motor
   inicializa_Motor_Shield();
   //inicializa e configura pinos usados no
//sensor ultrassonico
   pinMode(trigPin, OUTPUT);
   pinMode(echoPin, INPUT);
   //inicializa e configura comunicação serial
   Serial.begin(9600);
   //inicializa as variaveis do sensor
//ultrassonico
   distancia = le_sensor_ultrassonico();
   //inicializa as variaveis dos motores
   valorMotor_PWM = 0;
   ci74HC595Write(bitMotor1A, HIGH);
   ci74HC595Write(bitMotor1B, LOW);
   ci74HC595Write(bitMotor2A, HIGH);
   ci74HC595Write(bitMotor2B, LOW);
```

# Carrinho PID de distância – códido parte 4

- A segunda parte do código contém: a função `setup()` com:
  1. Inicialização do shield controlador de motores;
  2. Inicialização e configuração do sensor ultrassônico;
  3. inicialização da comunicação serial em 9600kbps;
  4. Atribuição da configuração inicial para os motores;
  5. Inicialização das variáveis do PID;

```
//inicializa as variaveis do PID
    erro = 0;
    integral = 0;
    der = 0;
    erro_anterior = 0;
    Kp = 30.0;
    Kd = 0.0;
    Ki = 0.0;
    Serial.println("Esperando 1 segundos");
    delay(1000);
    Serial.println("Setup completo");
}
```

# Carrinho PID de distância – códido parte 5

- Na terceira parte do código começa a função loop.
- É implementada a leitura dos sensores ultrassônico de distância;

```
void loop() {
  // LÊ OS SENSORES --------------------
//Lê o senor de distancia ultrassonico
  distancia = le_sensor_ultrassonico();
  Serial.print(" Dist=");
  Serial.print(distancia);
  Serial.print("cm ");
// fim de LÊ OS SENSORES ---------------
```

# Carrinho PID de distância – códido parte 6

- Na quarta parte do código continua a função loop.
- Nesta parte é implementado o controle;

```
// CALCULA O CONTROLE  ------------------------
  // NESTE controle queremos para a 5 cm do anteparo
  erro = distancia - 5.0;
  integral = integral + erro;
  der = erro - erro_anterior;
  erro_anterior = erro;
  //calcula o valor do PID e a velocidade dos motores
  PID = Kp*erro + Ki*integral + Kd*der;
  valorMotor_PWM =(int)(PID);
// fim de CALCULA O CONTROLE  ------------------------
```

# Carrinho PID de distância – códido parte 7

- Na quinta parte do código termina a função loop.
- Nesta parte temos a verificação se os atuadores estão saturados;
- E atualizamos os valores dos atuadores.

```
// ATUALIZA VALOR DOS ATUADORES  --------------------
  //verifica saturacao dos motores
  if(valorMotor_PWM > 0){
    ci74HC595Write(bitMotor1A, HIGH);
    ci74HC595Write(bitMotor1B, LOW);
    ci74HC595Write(bitMotor2A, HIGH);
    ci74HC595Write(bitMotor2B, LOW);
    Serial.print("  FRENTE,  PWM= ");
  }
  if(valorMotor_PWM < 0){
    ci74HC595Write(bitMotor1A, LOW);
    ci74HC595Write(bitMotor1B, HIGH);
    ci74HC595Write(bitMotor2A, LOW);
    ci74HC595Write(bitMotor2B, HIGH);
    valorMotor_PWM =-valorMotor_PWM;
    Serial.print("  RE,  PWM= ");
  }
  if(valorMotor_PWM > 255){valorMotor_PWM =255;}
  //atualiza a velocidade dos motores
  analogWrite(pinMotor1PWM, valorMotor_PWM);
  analogWrite(pinMotor2PWM, valorMotor_PWM);

  Serial.println(valorMotor_PWM);
// fim de ATUALIZA VALOR DOS ATUADORES  --------------------
  delay(10);
```

# Carrinho PID de distância – códido parte 8

- Na sexta parte do código contém a função `le_sensor_ultrassonico` que implementa a sequência necessária para emitir a onda, que bate no objeto e volta, medir o tempo necessário para a ida e volta da onda, além de transformar em distância a partir da velocidade do som no ar.

```
float le_sensor_ultrassonico() {
  long duracao;
//Limpa o pino de Trigger
  digitalWrite(trigPin, LOW);
  delayMicroseconds(2);
//Coloca o pino de Trigger em HIGH por
//10 microseg
  digitalWrite(trigPin, HIGH);
  delayMicroseconds(10);
  digitalWrite(trigPin, LOW);
//Le o pino de echo, retornando o tempo
//em microseg da onda sonora ir e vir
  duracao = pulseIn(echoPin, HIGH);
//Calcula a distancia
  return ((float)duracao)*0.0174;
}
```

# Carrinho PID de distância – códido parte 9

- Ainda na sexta parte do código temos a função para inicializar os motores;

```
void inicializa_Motor_Shield(){
    pinMode(pinSH_CP, OUTPUT);
    pinMode(pinST_CP, OUTPUT);
    pinMode(pinEnable, OUTPUT);
    pinMode(pinDS, OUTPUT);

    pinMode(pinMotor1PWM, OUTPUT);
    pinMode(pinMotor2PWM, OUTPUT);
//    pinMode(pinMotor3PWM, OUTPUT);
//    pinMode(pinMotor4PWM, OUTPUT);
    digitalWrite(pinEnable, LOW);
}
```

# Carrinho PID de distância – códido parte 10

- A última função na sexta parte do código faz a comunicação com o shield controlador de motor para dizer a dizer a direção de rotação do motor;

```
void ci74HC595Write(byte pino, bool estado) {
  static byte ciBuffer;
  bitWrite(ciBuffer, pino , estado);

  digitalWrite(pinST_CP, LOW); //Inicia a transmissão
  digitalWrite(pinDS, LOW);    //Apaga tudo
                               //para preparar transmissão
  digitalWrite(pinSH_CP, LOW);
  for (int nB = 7; nB >= 0; nB--) {
      digitalWrite(pinSH_CP, LOW);  //Baixa o Clock
      digitalWrite(pinDS,  bitRead(ciBuffer, nB) );
//Escreve o BIT
      digitalWrite(pinSH_CP, HIGH); //Eleva o Clock
      digitalWrite(pinDS, LOW);     //Baixa o Data
  }
  digitalWrite(pinST_CP, HIGH);  //Finaliza a Transmissão
}
```

# Exercício para entrega

**Implemente o controle no seu carrinho e:**

1. Explique a função da variável `Kp`. Como ela influência o sistema? Coloque valores diferentes de `Kp` e explique as mudanças ao mudar o valor da distância da parede.
2. O que acontece se forem colocados valores `0.001`, `0.01` e `0.1` em `Ki`, se for mantido o melhor valor de `Kp` encontrado em 1? Mude o código, permitindo que `integral = integral + erro` somente ocorra se o valor de valor do atuador estiver abaixo do mínimo necessário para vencer o atrito do motor. Como comportamento mudou?
3. O que acontece se forem colocados valores `0.001`, `0.01` e `0.1` em `Kd`, se forem mantidos os melhores valor de `Kp` e `Ki`, alem das alterações no código, do ítem 2?