Introdução à Teoria de Controle: uma visão simplificada para Robôs Lego NXT

MAC318 – Introdução à Programação de Robôs Móveis

Leliane Nunes de Barros

Valquiria Fenelon

Objetivo dessa aula

Como fazer com que um robô móvel seja capaz de realizar movimentos efetivos, seguros e previsíveis usando a teoria de controle?

- Introdução à Teoria de Controle
  - Tipos de sistemas de controle
  - Tipos de controladores

Definições e siglas

Variável de Processo (y): variável que é controlada no processo, como temperatura, pressão, umidade, **distância de um robô a uma parede**, etc.

Definições e siglas

Variável de Processo (y): variável que é controlada no processo, como temperatura, pressão, umidade, **distância de um robô a uma parede**, etc.

Valor desejado ou de referência (r): valor desejado para a variável que queremos controlar, variável **y**. Por exemplo: temperatura a 22 graus, **distância de 40 cm da parede**.

Definições e siglas

Variável de Processo (y): variável que é controlada no processo, como temperatura, pressão, umidade, **distância de um robô a uma parede**, etc.

Valor desejado ou de referência (r): valor desejado para a variável que queremos controlar, variável **y**. Por exemplo: temperatura a 22 graus, **distância de 40 cm da parede**.

Variável Manipulada (u) : variável sobre a qual o controlador atua para controlar o processo, como a posição de uma válvula, tensão aplicada a uma resistência de aquecimento, **potência aplicada a um motor do robô**, etc.

Definições e siglas

Erro ou Desvio (e): diferença entre r e y (negativa ou positiva)

$$e = r - y$$

Definições e siglas

Erro ou Desvio (e): diferença entre r e y (negativa ou positiva)

$$e = r - y$$

Ação de controle (u): pode ser reversa ou direta. Define genericamente a atuação aplicada à u na ocorrência de variações de y. Por exemplo, um comando de movimentação para frente (ou para trás) com uma determinada potência:

- **Ação Reversa:** Se y aumenta, u diminui. Tipicamente utilizada em controles de aquecimento.

- **Ação Direta:** Se y aumenta, u aumenta. Tipicamente utilizada em controles de refrigeração

Sistema de Controle

A técnica de controle PID consiste em calcular um valor de atuação (u) sobre o processo a partir das informações do valor desejado (r) e do valor atual da variável do processo (y). Isso é feito transformando u em um sinal adequado ao atuador utilizado (válvula, resistência ou motor), e deve garantir um controle estável e preciso.

Sistema de Controle: exemplos

[Imagem: Agência Brasil]

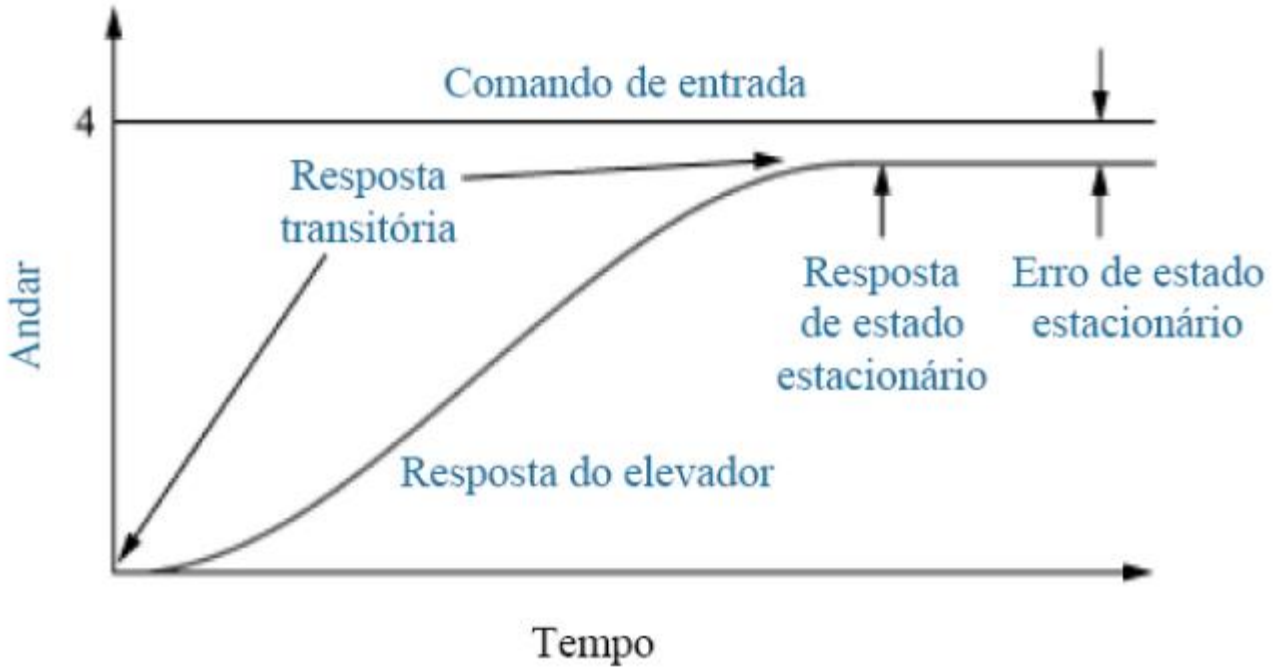
- Exemplos:
  - Controle de temperatura de fornos elétricos, geladeiras e ar-condicionado
  - Controle de antena apontando para direção comandada (compensação de perturbações, como por exemplo, vento ou chuva)
  - Controle de robôs industriais

Medidas de Desempenho

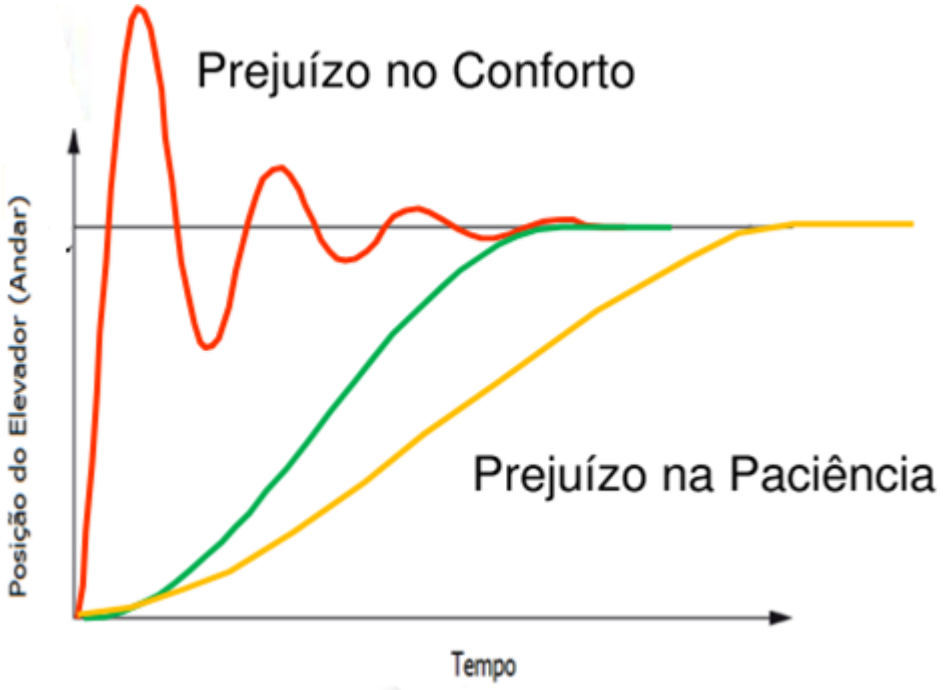
As principais medidas de desempenho de um sistema de controle são com relação ao valor da variável de processo y em termos de:

- Erro no regime estacionário (ou permanente)
- Resposta transitória

- Exemplo: movimentar um elevador para o andar y = 4



Desempenho na resposta transitória

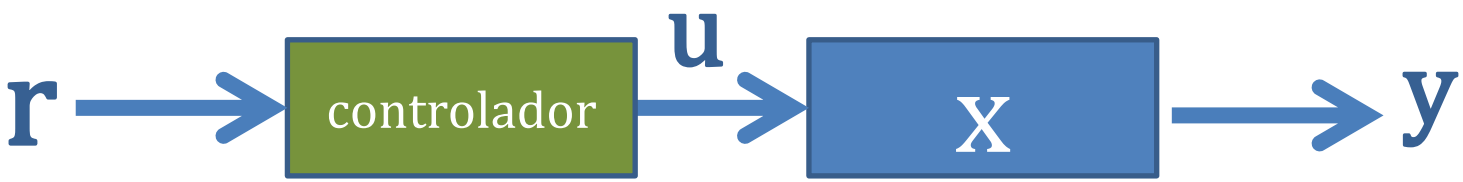


Controle sobre um Sistema Σ

- Estado do Sistema Σ (X): representação de Σ num determinado instante de tempo t, em termos de variáveis do processo, por exemplo, o estado X.
- Referência (r): como queremos que Σ se comporte.
- Saída (y): resposta do sistema Σ (novo valor para y)
- Entrada (u): atuação de controle dado como entrada para Σ (novo valor atribuido à variável manipulada u)
- Dinâmica (*f*(u,x)=y): descrição de como o estado X muda
- Realimentação: mapeia o valor da saída para entrada

Tipos de Controle

- Controle de Malha Aberta:



- Controle de Malha Fechada:

Atividades anteriores

- Tarefa 1: Robô chutador
  - Malha aberta ou fechada?
- Tarefa 2: Robô que percorre um quadrado
  - Malha aberta ou fechada?
- Tarefa 3: Robô que evita obstáculos
  - Malha aberta ou fechada?

Atividade de Controle

De uma distância de 80 cm, o robô deve se movimentar perpendicularmente a uma parede e parar ao atingir uma distância de 40 cm.

O robô móvel deve ser equipado com sensor ultrassônico na Porta 4.

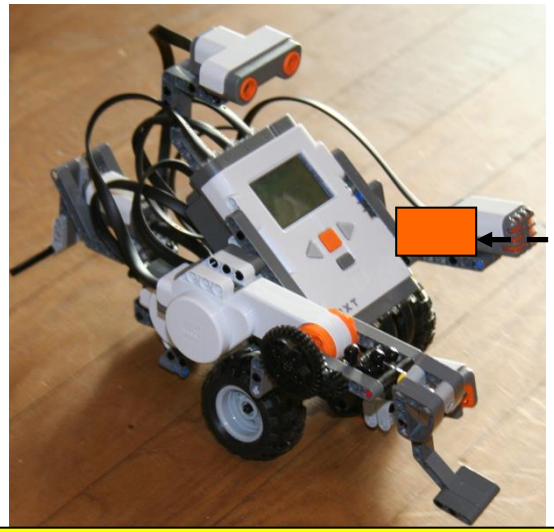
Objetivo: Analisar como o controle pode atuar neste sistema.

Atividade de Controle

De uma distância de 80 cm, o robô deve se movimentar perpendicularmente a uma parede e parar ao atingir uma distância de 40 cm.

O robô móvel deve ser equipado com sensor ultrassônico na Porta 4.

Objetivo: Analisar co
controle pode atuar
sistema.



Tentem programar o seu robô para realizar essa tarefa.

Exemplo: robô que se aproxima de uma parede

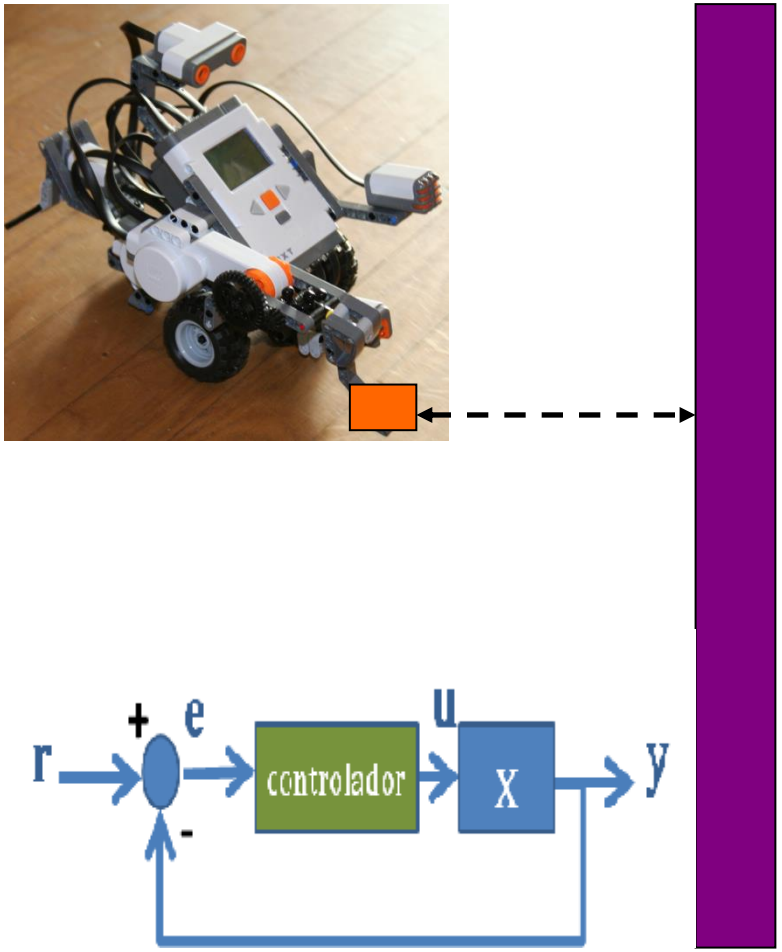
$r \rightarrow$ *distância desejada (40 cm)*

$e \rightarrow$ *diferença entre distância desejada e percebida*

$u \rightarrow$ *potência dos motores*

$x \rightarrow$ *estado do sistema no instante de tempo t (representado pela distância da parede)*

$y \rightarrow$ *distância real (percebida)*



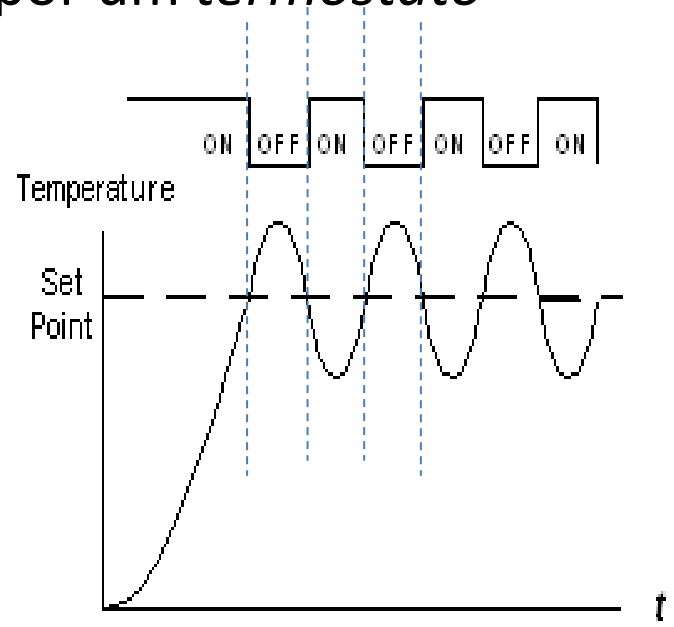
Tipos de Controladores

Tipos de Controladores (Malha Fechada)

1. Liga-desliga (on-off, bang-bang)
2. Proporcional (P)
3. Integral (I)
4. Derivativo (D)
5. Proporcional + Integral (PI)
6. Proporcional + Derivativo (PD)
7. Proporcional + Integral + Derivativo (PID)

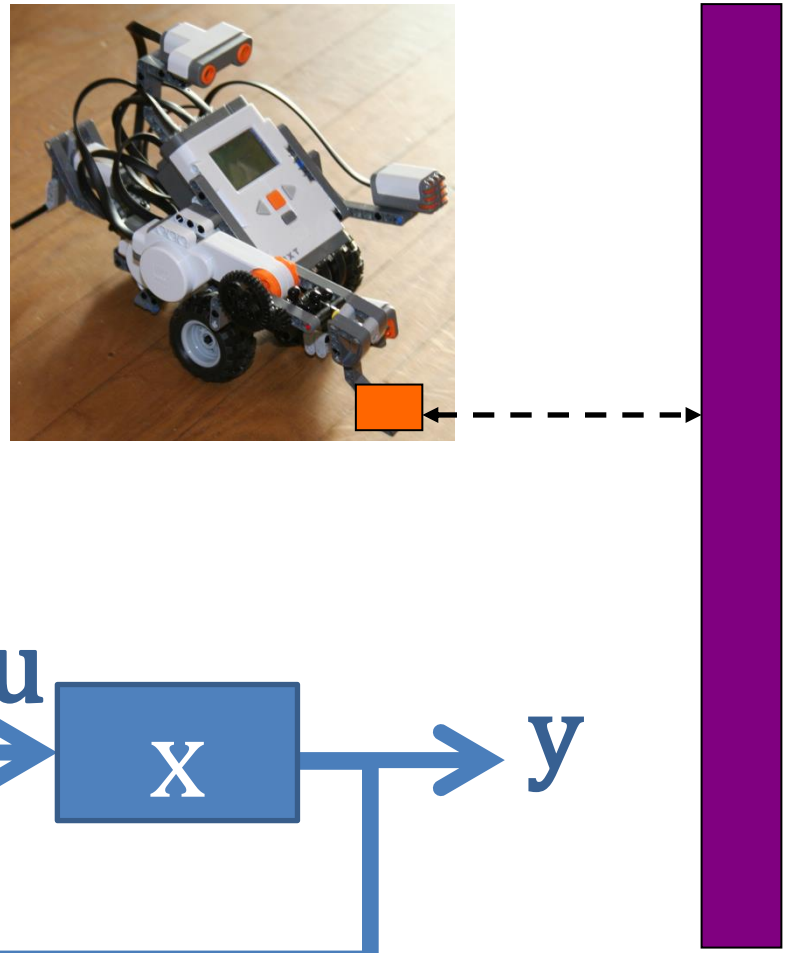
Controle liga-desliga (bang-bang)

- Compara sinal de entrada com realimentação
  - se saída for maior que a entrada, ***desliga*** o atuador;
  - se a saída for menor que a entrada, ***liga*** o atuador.
- Ex.: fornos elétricos e geladeiras:
  - Calefator ou compressor controlado por um *termostato* (controlador de temperatura).
- Vantagens: controle simples e de baixo custo
- Desvantagens: oscilação da saída, não garante precisão e pode desgastar o controlador e atuador(consumo de energia) .

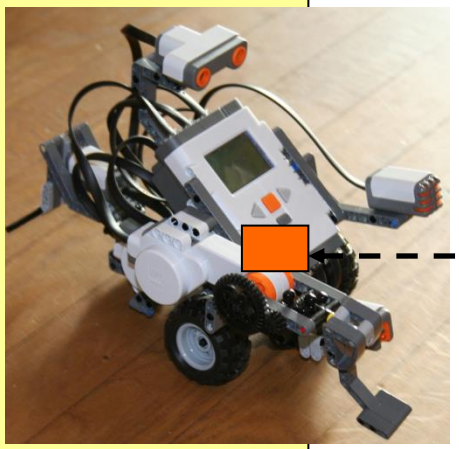


Controle liga-desliga (bang-bang)

$$u = \begin{cases} 100 & if \quad e < 0 \\ -100 & if \quad e > 0 \\ 0 & if \quad e = 0 \end{cases}$$



```
import lejos.nxt.*;
public class LigaDesliga {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
     int r = 40;
     int y, u;
     LCD.drawString("Controle Liga Desliga", 0, 0);
     while (true) {
        y = sonic.getDistance();
        u=300;
        Motor.B.setSpeed(u);
        Motor.C.setSpeed(u);
        if (y > r) {
          Motor.B.forward();
          Motor.C.forward();
        } else {
          Motor.B.backward();
          Motor.C.backward();
        }
      }
    }
}
```



Execute o programa para diferentes valores de u

Controlador Proporcional

- Limitação do controle bang-bang: quando o erro é pequeno tem uma reação exagerada. Pode não convergir para um regime estacionário

Controlador Proporcional

- Limitação do controle bang-bang: quando o erro é pequeno tem uma reação exagerada. Pode não convergir para um regime estacionário

- Idéia: Controlador proporcional! O sinal do controle (u) é variável e proporcional ao erro medido (e).

$$u = y - r$$

- Menor erro implica em menor sinal de controle, e vice-e-versa.

- Resultado: atuação mais suave e mais estável.

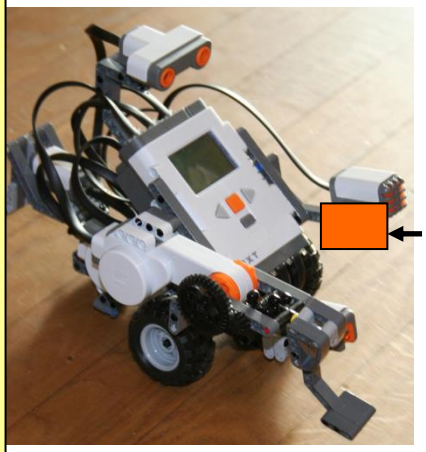
Regulator Thread

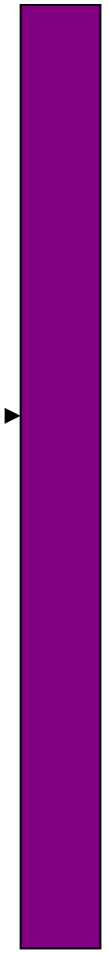
- O uso da classe regulatedMotor, dentre outras coisas, ativa uma *thread* para regulagem da velocidade dos motores de um robô diferencial (vide slides das aulas 2 e 3).
- A regulagem dos motores pode interferir com o sistema de controle que queremos desenvolver.
- Por isso, usaremos como variável de controle a potência do motor, ao invés da velocidade, através do método setPower() da classe NXTMotor

Controlador Proporcional Simplificado

```
import lejos.nxt.*;
public class PPowerInt {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    int r = 40;
    int y ;
    int u ;
    int e ;

    LCD.drawString("Controle Proporcional", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = (y - r);
      u = e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower(u);
      mB.setPower(u);
    }
  }
}
```





Rodem esse programa no robô

Controlador Proporcional Simplificado

```
import lejos.nxt.*;
public class PPowerInt {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    int r = 40;
    int y ;
    int u ;
    int e ;

    LCD.drawString("Controle Proporcional", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = (y - r);
      u = e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower(u);
      mB.setPower(u);
    }
  }
}
```

Obs: o objetivo pode não ser alcançado com valores de u muito pequenos

Controlador Proporcional

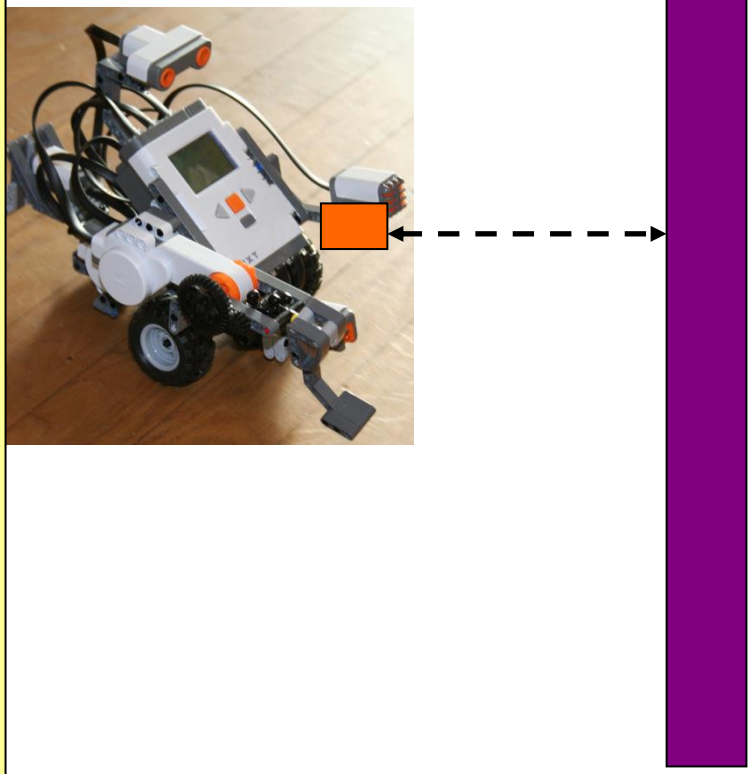
- Idéia: uso de uma constante para aumentar o ganho de potência

$$u = k_p e$$

- $\boldsymbol{k_p}$ (ganho proporcional) define a intensidade da ação proporcional e faz ir mais rápido ao ponto desejado
  - A potência mínima ainda é útil
  - A potência total pode ajudar o robô a ir mais rápido para o seu objetivo quando ele estiver longe.
- Converge para um erro estacionário.

Controlador Proporcional

```
import lejos.nxt.*;
public class PPowerInt {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    int r = 40;
    int y ;
    int u ;
    int e ;
    int kp = 25;
    LCD.drawString("Controle Proporcinal", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = (y - r);
      u = kp * e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower(u);
      mB.setPower(u);
    }
  }
```



Controlador Proporcional

Aumentando o ganho:

- Oscilações: um ganho alto pode causar uma potência alta mesmo estando próximo do objetivo.
- Diminui a estabilidade.
- Aumento da velocidade de convergência.
- Tem mais *overshoot* (arrancadas).

Controlador Proporcional

Aumentando o ganho:

- Oscilações: um ganho alto pode causar uma potência alta mesmo estando próximo do objetivo.
- Diminui a estabilidade.
- Aumento da velocidade de convergência.
- Tem mais *overshoot* (arrancadas).

Diminuindo o ganho:

- Menos oscilações.
- Aumenta a estabilidade.
- Diminui a velocidade de convergência.
- Tem menos *overshoot*.

Controlador Proporcional

Aumentando o ganho:

- Oscilações: um ganho alto pode causar uma potência alta mesmo estando próximo do objetivo.
- Diminui a estabilidade.
- Aumento da velocidade de convergência.
- Tem mais *overshoot* (arrancadas).

Diminuindo o ganho:

- Menos oscilações.
- Aumenta a estabilidade.
- Diminui a velocidade de convergência.
- Tem menos *overshoot*.

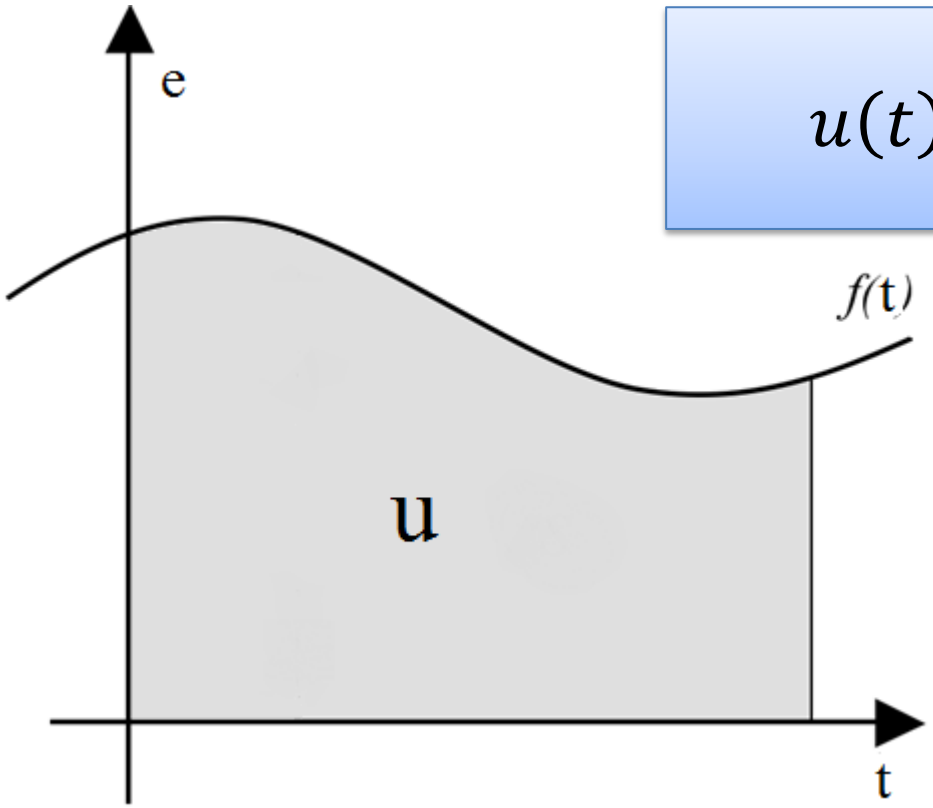
Erro no regime estacionário?
Com uma régua, medir a distância em que o robô parou.

Controlador Integral

Limitação do controle proporcional: *erro no regime estacionário* (estado não muda mesmo que a resposta não seja a desejada) :

- Olhando para o passado, o controle integral tem o efeito de eliminar o erro de um controle puramente proporcional.
- O controle integral consiste em uma resposta na saída do controlador (u) que é proporcional à amplitude e duração de ***todos os erros percebidos anteriormente***.

Calculando Integral



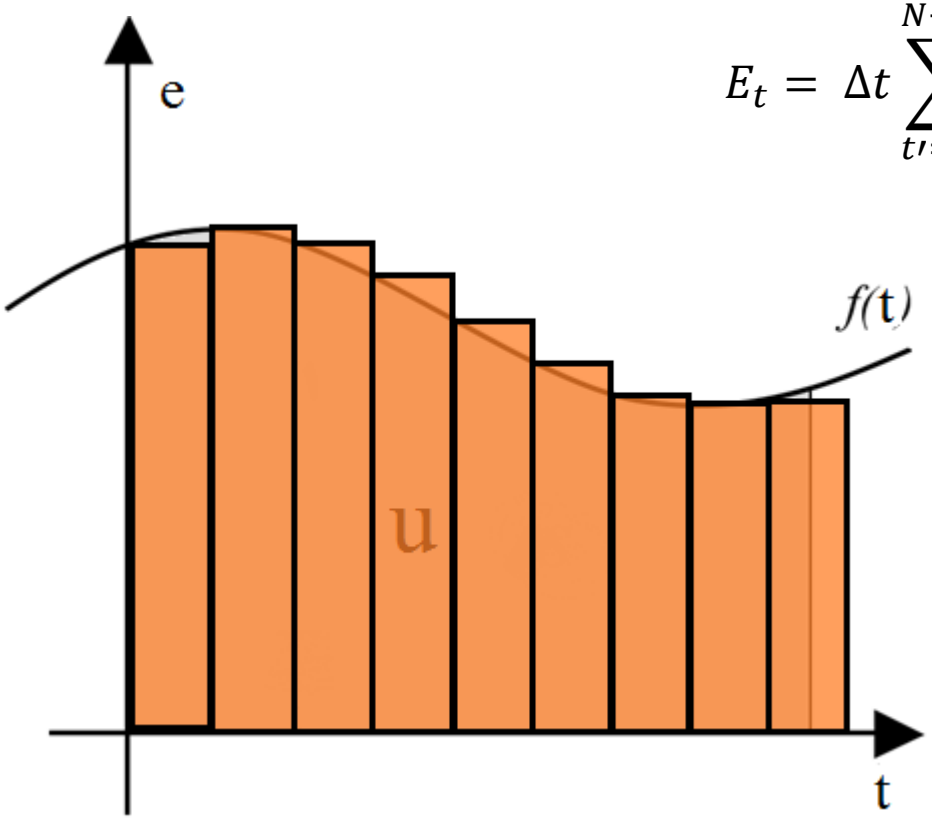
$$u(t) = k_I \int_0^t e(\tau) d(\tau)$$

k_I: ganho integral, define a intensidade da ação integral

Integral = soma de aproximações

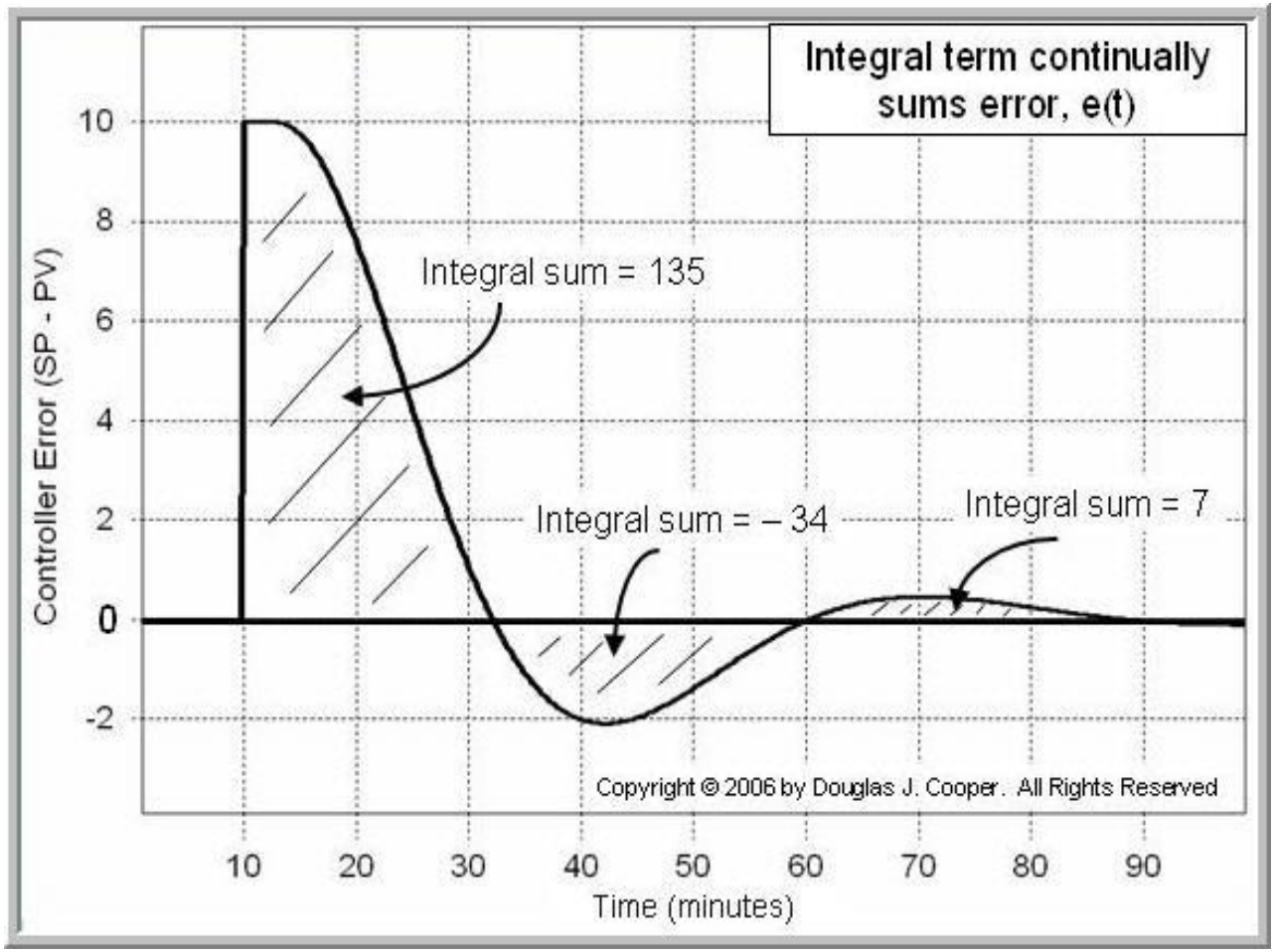
$$\int_0^t e(\tau)d\,\tau \approx \sum_{t=0}^{N} e(t)\Delta t = E$$

$$E_t = \Delta t \sum_{t\prime=0}^{N+1} e(t') = e(N+1) + E_{t-1}$$



$$E_t = E_{t-1} + e$$

Somatória das áreas

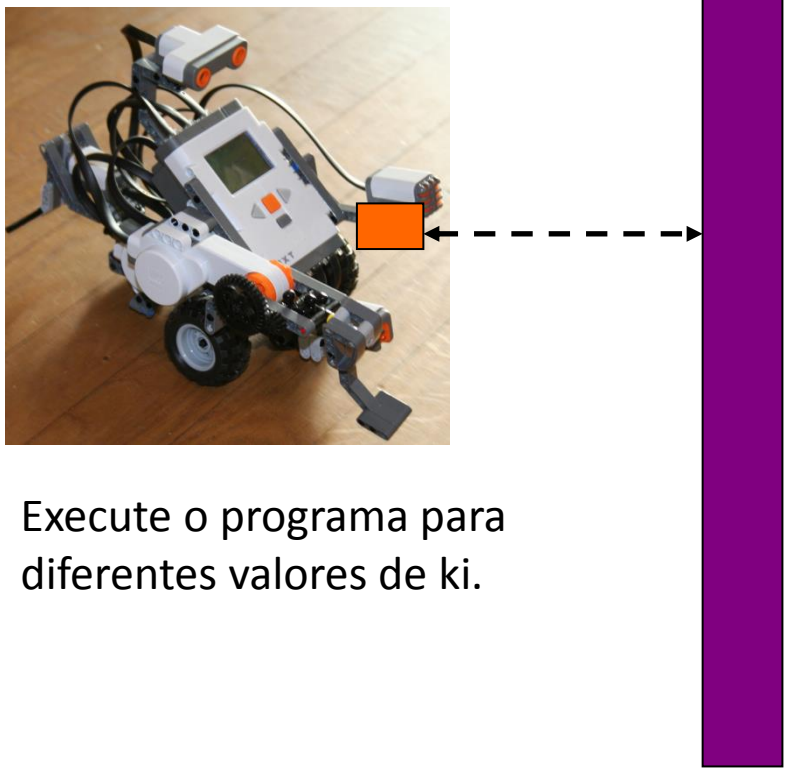


http://www.controlguru.com/wp/p69.html

Controlador Proporcional e Integral:

```
import lejos.nxt.*;

public class ControlePI {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    double r = 40;
    double y = 0;
    double u;
    double e = 0;
    double E = 0;
    double kp = 25;
    double ki = 0.01;
    LCD.drawString("Controle PI", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = y - r;
      E = E + e;
      u = ki * E + kp * e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower((int)u);
      mB.setPower((int)u);
    }
  }
}
```

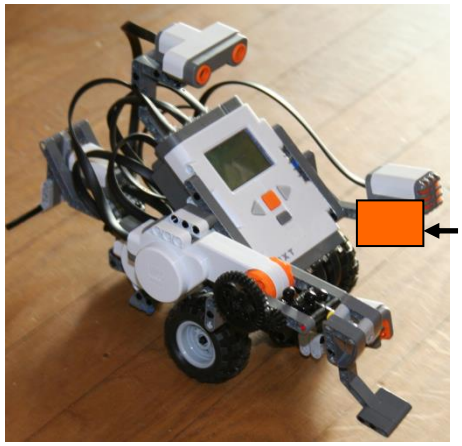


Execute o programa para diferentes valores de ki.

Controlador Proporcional e Integral:

```
import lejos.nxt.*;

public class ControlePI {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    double r = 40;
    double y = 0;
    double u;
    double e = 0;
    double E = 0;
    double kp = 25;
    double ki = 0.01;
    LCD.drawString("Controle PI", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = y - r;
      E = E + e;
      u = ki * E + kp * e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower((int)u);
      mB.setPower((int)u);
    }
  }
}
```



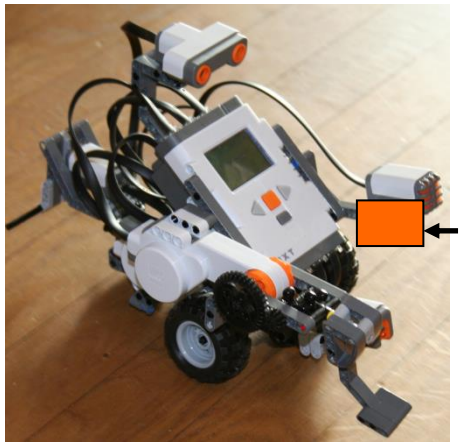
Execute o programa para

Execute o programa para diferentes valores de ki e com kp alto.

Controlador Proporcional e Integral:

```
import lejos.nxt.*;

public class ControlePI {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    double r = 60;
    double y = 0;
    double u;
    double e = 0;
    double E = 0;
    double kp = 20;
    double ki = 0.01;
    LCD.drawString("Controle PI", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = y - r;
      E = E + e;
      u = ki * E + kp * e;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower((int)u);
      mB.setPower((int)u);
    }
  }
}
```



Execute o programa para

Erro no regime estacionário?
Com uma régua, medir a distância em que o robô parou e comparar com a medida anterior.

Propriedades Proporcional e Integral

- A ação integral sobre a proporcional faz com que haja o acionamento do atuador após o período transitório, assim o controle é muito preciso, embora mais lento.

  - Integrador torna o sistema lento, o uso de um integrador pouco atuante, retarda em demasia a convergência

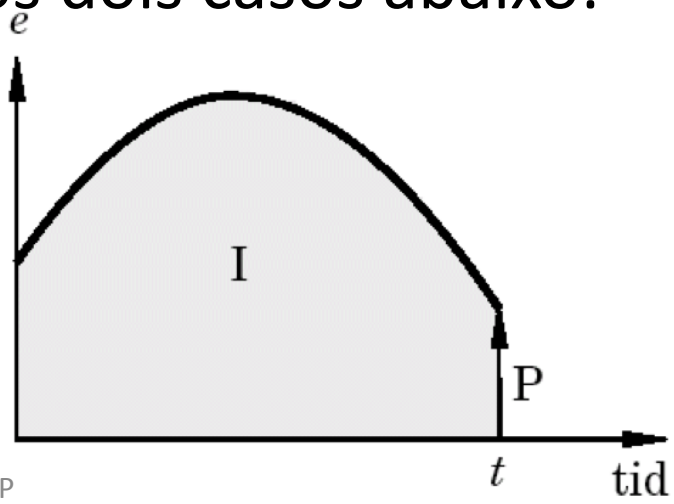
  - Com um termo integral excessivamente atuante pode levar o processo à instabilidade (mesmo com um erro pequeno ele continua atuando muito).

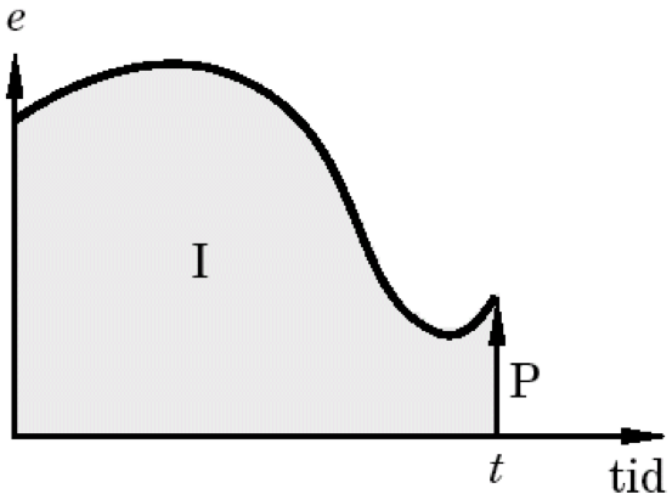
Controlador Derivativo

Limitações do controle PI: não possui nenhuma predição sobre o futuro do sistema (se o erro está crescendo ou decrescendo).

Controle derivativo corrige variações bruscas devido a perturbações ou no início da partida do processo.

- Exemplo: no controle PI, o mesmo sinal de controle é obtido nos dois casos abaixo:

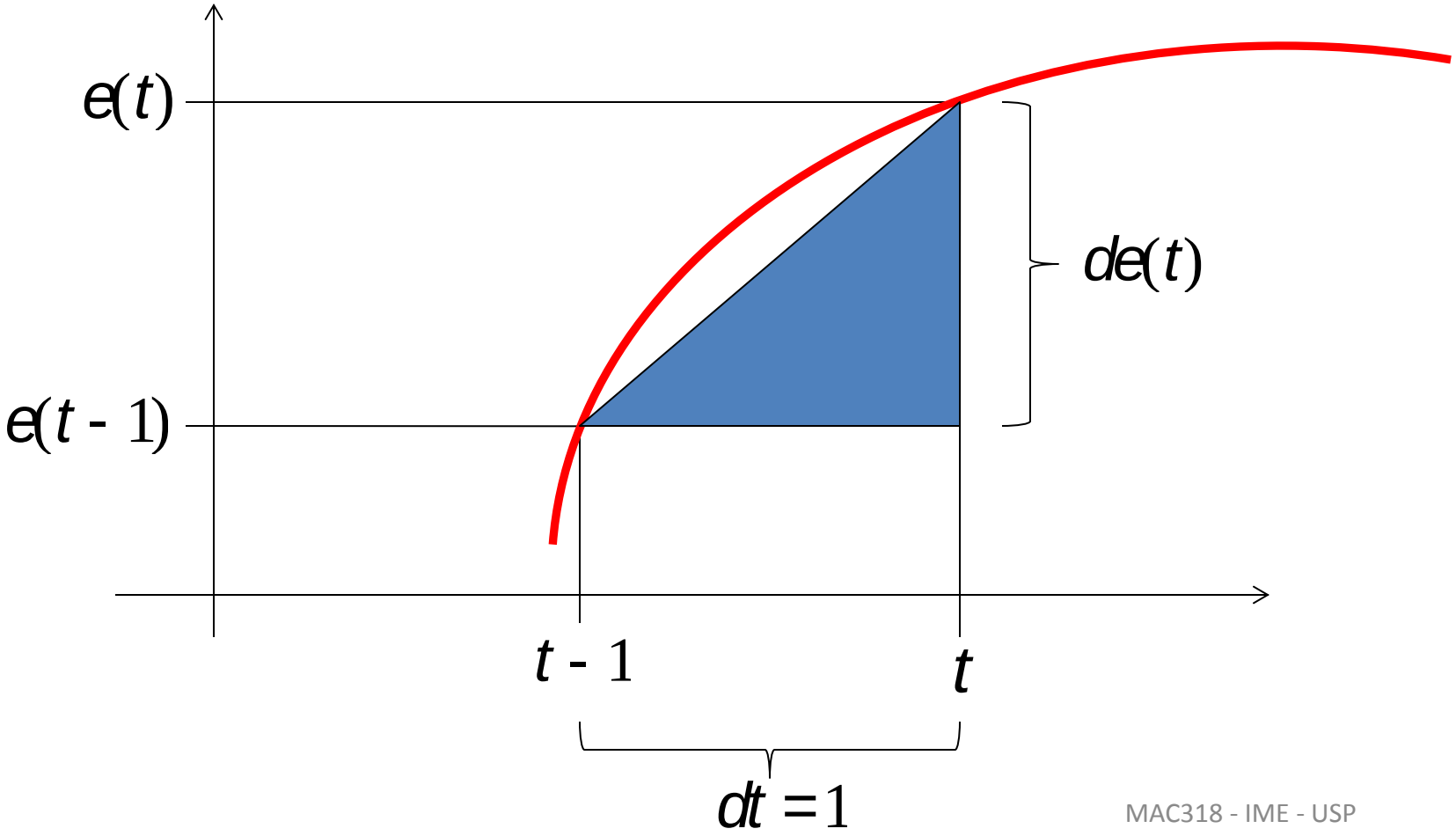




Controlador Derivativo

- Adiciona um elemento que realiza uma previsão do comportamento do robô no futuro próximo, tendo como base os valores atuais e do passado próximo: reage rapidamente às mudanças do erro.

Calculando a Derivada



Calculando a Derivada

- A derivada é a taxa de variação da função:

$$\frac{de(t)}{dt} = \frac{e_t - e_{t-1}}{dt}$$

- Velocidade de variação do erro (considerando um intervalo *dt* unitário.

$$vve = e_t - e_{t-1}$$

Propriedades Derivativas

- Ação derivativa consiste em uma resposta na saída do controlador ***u*** que é proporcional à velocidade de variação do erro.
- Durante perturbações e na partida do processo, a ação derivativa atua sempre no sentido de atenuar as variações do erro.
- Só atua quando há variação no erro (processo estável implica num efeito nulo)

Propriedades Derivativas

- Sua principal função é melhorar o desempenho do processo durante os transitórios.
  - A intensidade da ação derivativa é ajustada variando-se o intervalo de cálculo da diferença (ganho K_D).
  - O aumento de k_D aumenta a ação derivativa, reduzindo a velocidade de variação de y, consequentemente reduzindo o *overshoot (*deixa o processo mais lento).
  - Diminuindo o ganho k_D a velocidade de variação y aumenta e aumenta *overshoot* (acelera o processo).

Controlador Proporcional e Derivativo (PD)

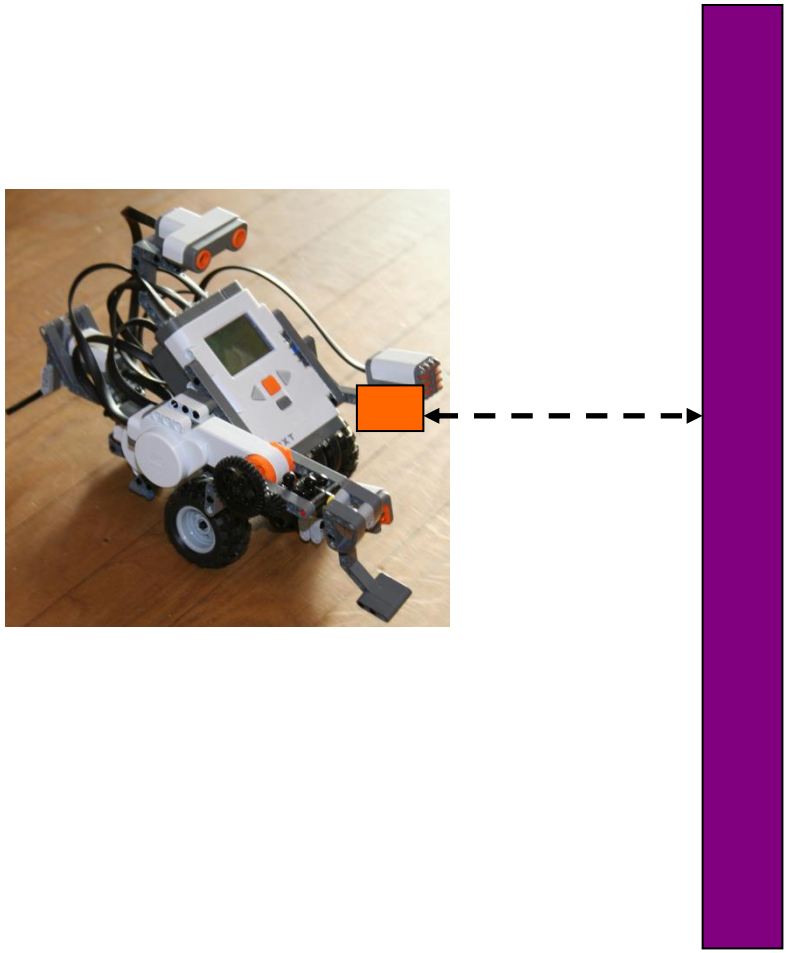
- Isoladamente o derivativo não é uma técnica de controle.
- Deve ser empregado associado a uma ação proporcional.

$$u(t) = k_P e(t) + k_D \frac{de(t)}{dt}$$

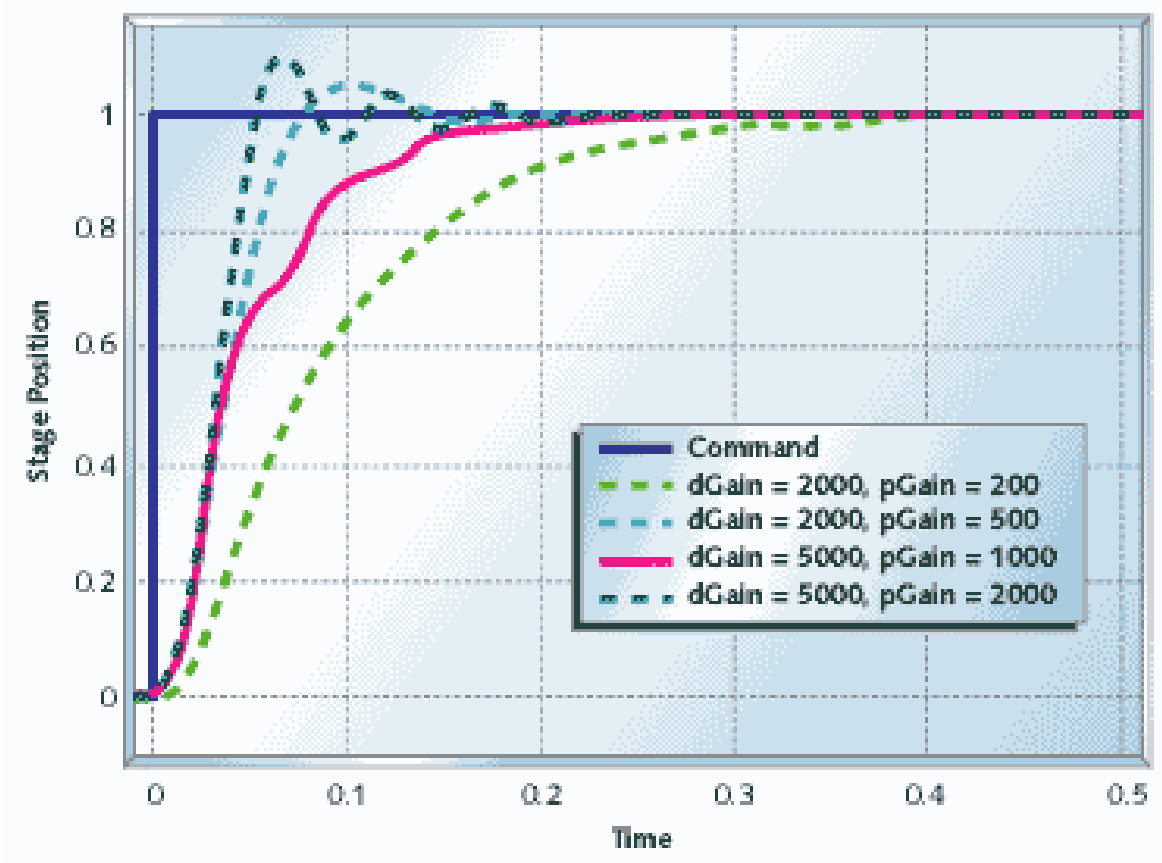
$$u = k_P e + k_D \dot{e}$$

Algoritmo Proporcional e Derivativo

```
import lejos.nxt.*;
public class ControlePI {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    double r = 40;
    double y = 0;
    double u;
    double e = 0;
    double eant =0;
    double kp = 25;
    double kd =  10;
    LCD.drawString("Controle Proporcinal 2", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = y - r;
      edif= e – eant;
      eant = e;
      u =  kp * e +  kd*edif ;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower((int)u);
      mB.setPower((int)u);
    }
  }
}
```



Propriedades PD



https://www.embedded.com/2000/0010/0010feat3.htm

Resumo: Controle PID

$$u(t) = K(e(t) + \frac{1}{T_I}\int^{t} e(\tau)d\tau + T_D\frac{de(t)}{dt})$$

$$= P + I + D$$

Controle PID

- Controle proporcional: um controlador que multiplica o erro por uma constante:

$$u_t = e_t \times k_P$$

- Controle integral: um controlador que considera a integral do erro no tempo (usa a história):

$$u_t = (e_0 + e_1 + \ldots + e_n) \times k_I$$

- Controle derivativo: um controlador que considera a diferencial do erro no tempo (previsão futura):

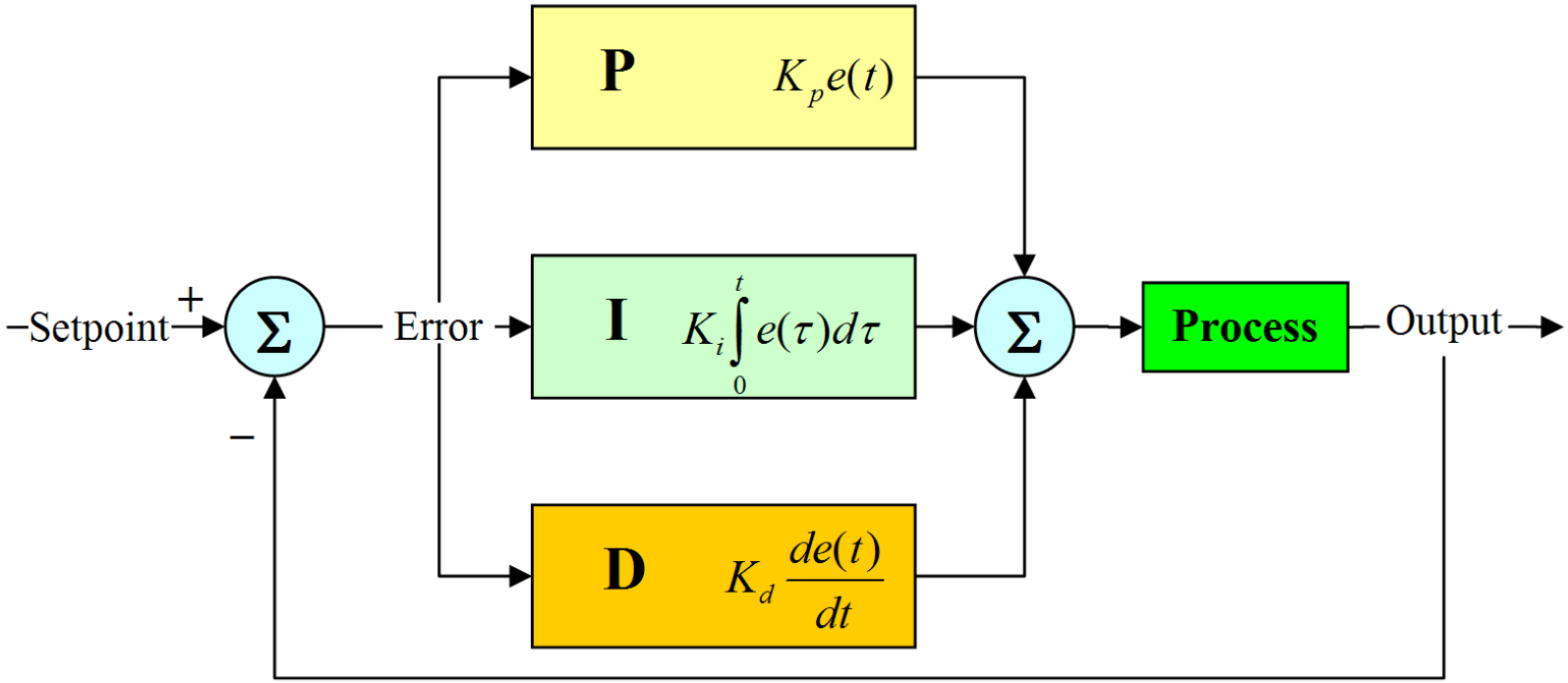
$$u_t = (e_t - e_{t-1}) \times k_D$$

- k_P , k_I e k_D são os ganhos das parcelas P, I e D do controle PID, também chamadas de ações do controle, que definem a intensidade de cada ação de controle.

Intuitivamente

- Proporcional:
  - Trata o erro ATUAL (medido em uma iteração).
- Integral:
  - Trata o erro acumulado: aplica um controle constante mesmo quando o erro é zero.
- Derivativo:
  - Antecipa e reage às taxas de mudança rápidas antes que o erro cresça muito.

Controle PID



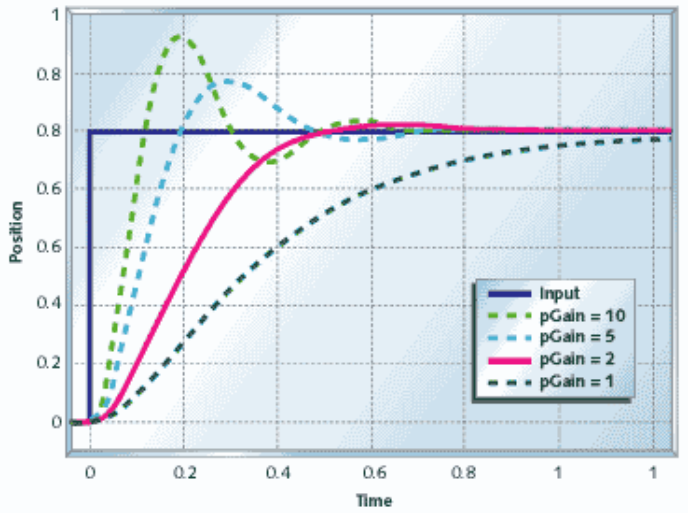
http://en.wikipedia.org/wiki/PID_controller

O Algoritmo PID

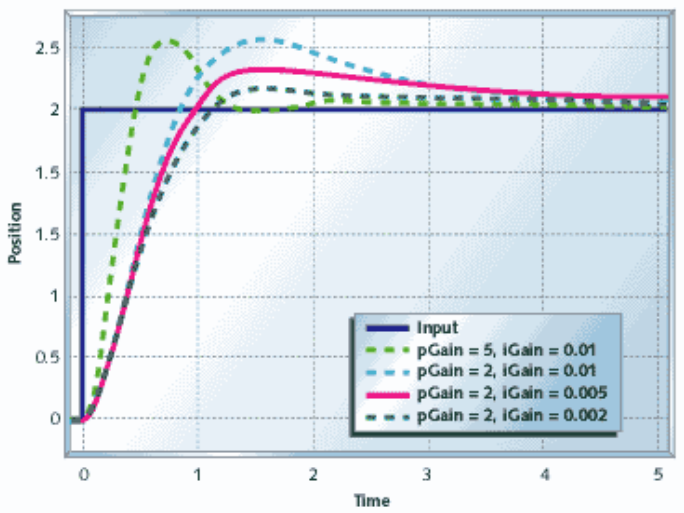
- Unindo as três técnicas conseguimos unir o controle básico da ação P com a eliminação do erro da ação I e com a redução de oscilações da ação D.
- Cria a dificuldade de ajustar a intensidade de cada um dos termos, processo chamado de *sintonia do PID*
- Um bom desempenho também depende da capacidade do sistema para atingir uma resposta independente do controlador utilizado

Comparação P, PI, PD, PID

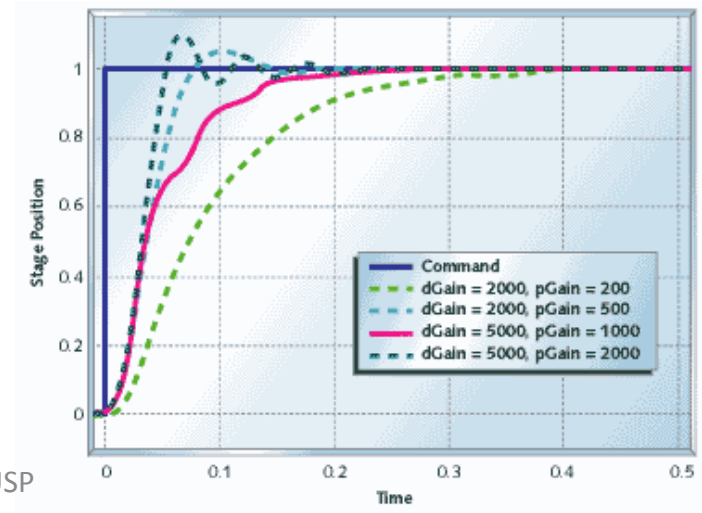
P



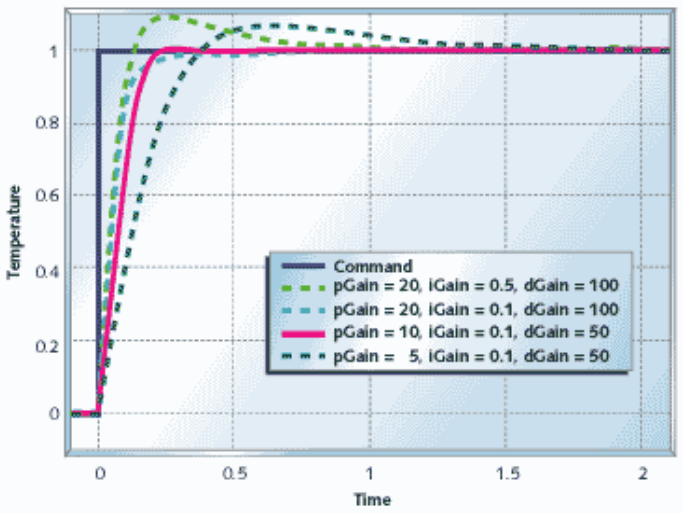
PI



PD

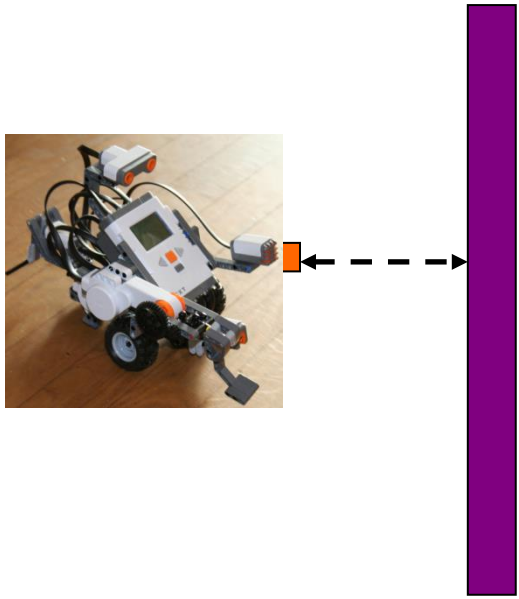


PID



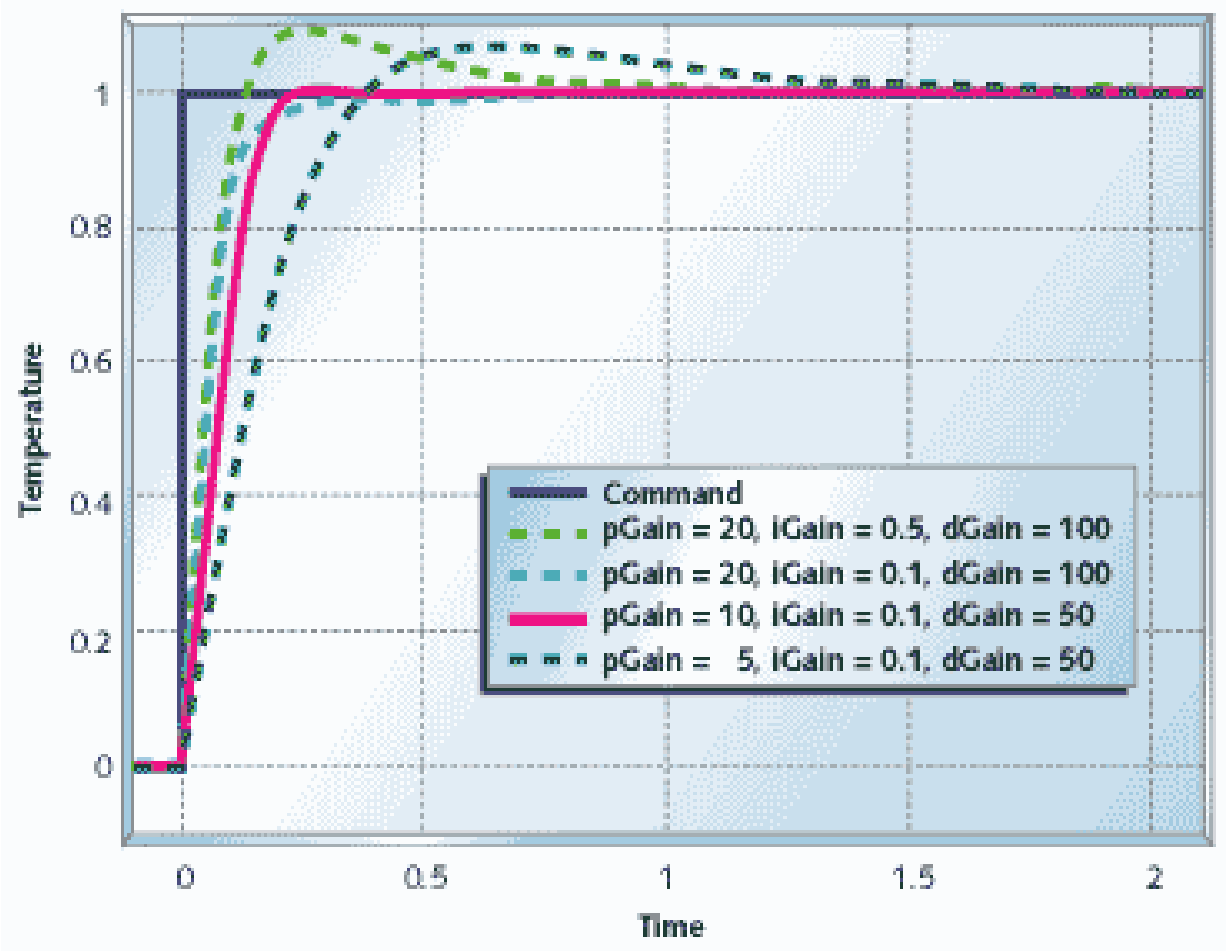
O Algoritmo PID

```
import lejos.nxt.*;
public class ControlePI {
  public static void main(String[] args) {
    UltrasonicSensor sonic = new UltrasonicSensor(SensorPort.S4);
    NXTMotor mC = new NXTMotor(MotorPort.C);
    NXTMotor mB = new NXTMotor(MotorPort.B);
    double r = 40;
    double y = 0;
    double u;
    double e = 0;
    double eant =0;
    double E = 0;
    double kp = 25;
    double ki = 0.01;
    double kd =  10;
    LCD.drawString("Controle Proporcinal 2", 0, 0);
    while (true) {
      y = sonic.getDistance();
      e = y - r;
      E = E + e;
      edif= e – eant;
      eant = e;
      u = ki * E + kp * e +  kd*edif ;
      if (u > 100) u = 100;
      if (u < -100) u = -100;
      mC.setPower((int)u);
      mB.setPower((int)u);
    }
  }
}
```



Execute o programa para diferentes valores de kp, ki e kd.

PID para controle de temperatura



https://www.embedded.com/2000/0010/0010feat3.htm

Resumindo - Ganhos

Ganho	Ao aumentar, o processo...	Ao diminuir, o processo
k_p	Torna-se mais rápido. Fica mais instável ou mais oscilante. Tem mais *overshoot*.	Torna-se mais lento. Geralmente se torna mais estável ou menos oscilante. Tem menos *overshoot*.
k_i	Torna-se mais rápido, atingindo rapidamente o *setpoint*. Fica mais instável ou mais oscilante. Tem mais *overshoot*.	Tonar-se mais lento, demorando para atingir o *setpoint*. Fica mais estável ou mais oscilante.
k_d	Torna-se mais lento. Tem menos *overshoot*.	Torna-se mais rápido. Tem mais *overshoot*.

Como melhorar o processo

Se o desempenho do processo...	Tente uma a uma as opções:
Está quase bom, mas o overshoot está um pouco alto.	Aumentar k_P em 20% Diminuir k_I em 20% Aumentar k_D em 50%
Está quase bom, mas não tem *overshoot* e demora para atingir o *setpoint.*	Diminuir k_P em 20% Aumentar k_I em 20% Diminuir k_D em 50%
Está bom, mas o sinal de controle (u) está sempre variando entre 0% e 100% ou está variando demais.	Aumentar k_P em 20% Diminuir k_D em 50%
Está ruim. Após a partida, o transitório dura vários períodos de oscilação, que reduz muito lentamente ou não reduz.	Aumentar k_P em 50%
Está ruim. Após a partida avança lentamente em direção ao *setpoint e (u)* já é menor que 100%.	Diminuir k_P em 50% Aumentar k_I em 50% Diminuir k_D em 70%

Referências

- NISE, Norman S. Engenharia de sistemas de controle. 3. ed. Rio de Janeiro: LTC, c2002. 695 p. ISBN 8521613016.
- Apresentação de aulas de:
  - Robótica da FEI (Reinaldo Bianchi): http://fei.edu.br/~rbianchi/robotica/
  - Control of Mobile Robots (Magnus) Egerstedt:https://www.coursera.org/course/conrob
  - Controle 1 (Leonardo Gonsioroski da Silva) http://www.prof-leonardo.com.br/controle1/Aula1-ControleI.pdf
- Artigo técnico: http://www.novusautomation.com/artigosnoticias/arquivos/ArtigoPIDBasicoNovus.pdf