Oficina Seguidor de Linha 2.0

O Guia Definitivo do Mochileiro Maker

PETEE UFMG

OFICINA DE SEGUIDOR DE LINHA, UNIVERSIDADE FEDERAL DE MINAS GERAIS

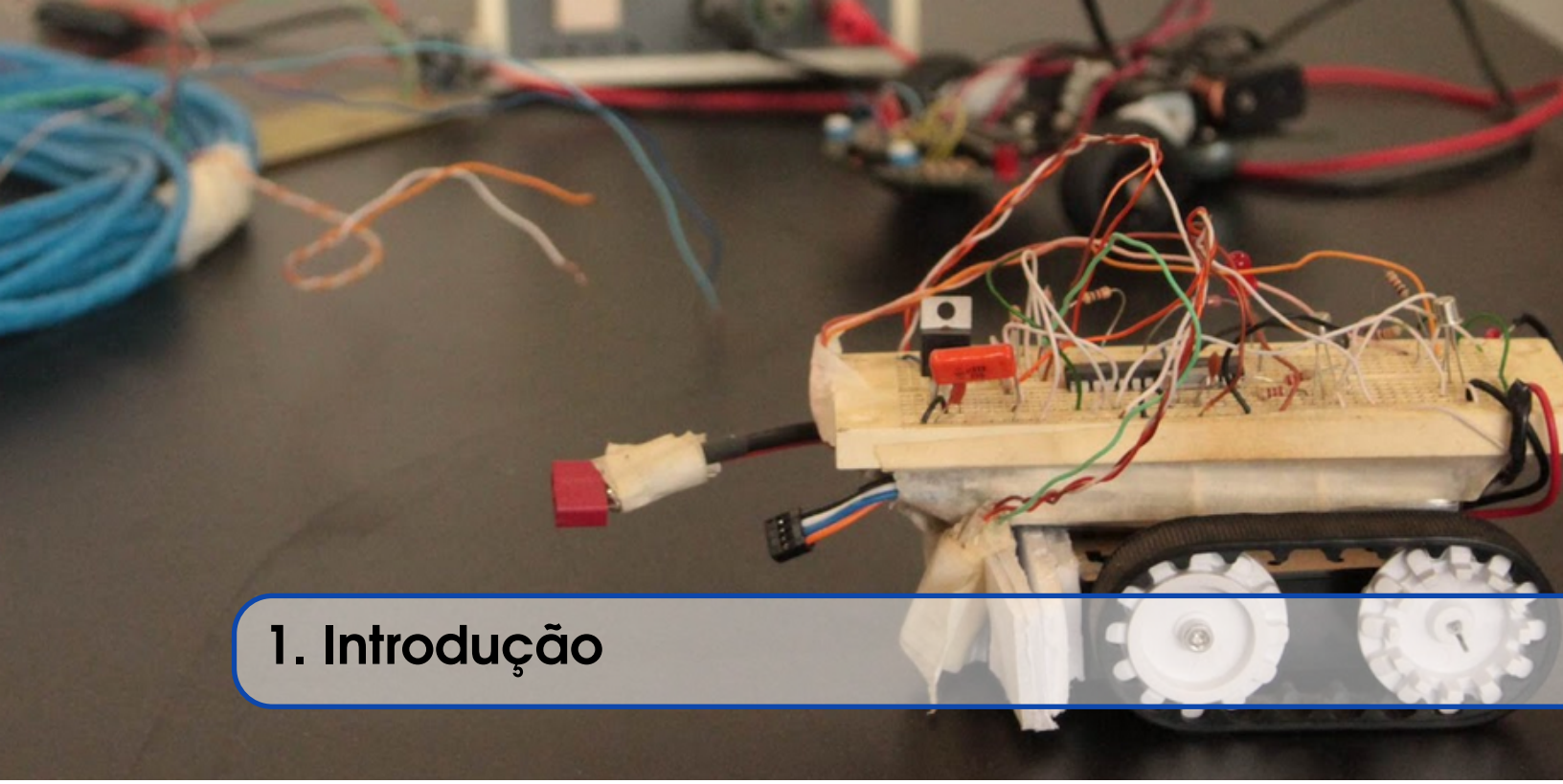
HTTP://PETEE.CPDEE.UFMG.BR/

Este texto foi escrito como complemento para a oficina de seguidor de linha, promovida pelo Programa de Educação Tutorial da Engenharia Elétrica (PETEE) da Universidade Federal de Minas Gerais.

1ª Ed - Mateus Simões, Junho 2017
2ª Ed - Felipe Lisboa, Junho 2018

Sumário

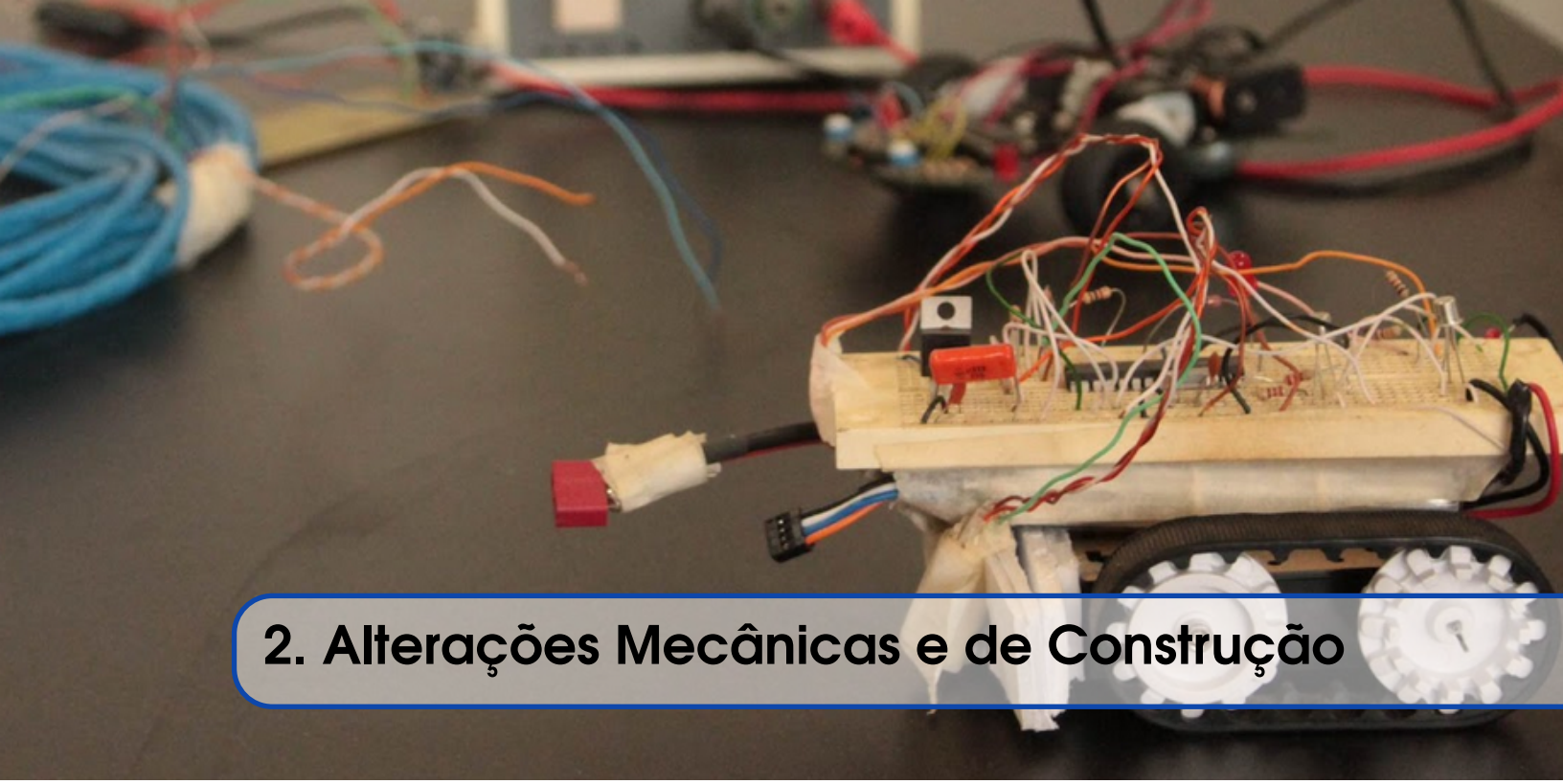
Oficina 2.0



1. Introdução

A Oficina de Seguidor de Linha 2.0 é uma extensão da Oficina de Seguidor de Linha 1.0 e tem como principal objetivo desenvolver melhorias em relação ao carrinho previamente desenvolvido. Serão abordadas melhorias relacionadas à aspectos mecânicos e de construção do carrinho assim como melhorias em relação aos algoritmos de controle.

Para tal, serão desenvolvidos alguns conceitos teóricos necessários para total compreensão das modificações. No entanto, os conceitos não serão abordados à fundo, uma vez que se referem a assuntos que apresentam um nível de complexidade maior. Através dos conceitos teóricos, uma série de dicas de montagem e implementações de algoritmos serão desenvolvidas e comentadas.



2. Alterações Mecânicas e de Construção

2.1 Chassi e a Otimização das Acelerações Angular e Linear

Para um sistema tal como um carrinho seguidor de linha, é desejável que um torque produzido pelos motores gere uma aceleração angular considerável. Em outras palavras, deseja-se que o carrinho responda rápido aos comandos de mudança de direção.

A relação de linearidade entre o torque produzido pelos motores e a aceleração angular gerada se dá pela 2ª Lei de Newton para o movimento de rotação:

$$\sum \tau = I \cdot \alpha$$

Segundo a Lei, o somatório dos torques aplicados é igual ao momento de inércia em relação à um eixo vezes a aceleração angular do carrinho.

Para atingir o resultado desejado, faz sentido que o momento de inércia seja baixo. Dessa forma, o torque gerado pelos motores produz uma aceleração angular maior. Uma analogia pode ser feita com o 2ª Lei de Newton para o movimento linear:

$$\sum F = m \cdot a$$

Pela Lei de Newton, uma força aplicada sobre um corpo de massa menor gera uma aceleração linear maior.

Para obter um carrinho mais rápido e responsivo, desejamos ambas acelerações angular e linear altas. A partir disso é possível concluir que **devemos reduzir a massa do carrinho e seu momento de inércia**. Mas como fazer isso?

A resposta para a primeira questão vem de maneira direta, basta construí-lo mais leve. Várias alternativas são possíveis para tal. Um bom projetista de seguidor de linha sempre deve fazer o possível para que seu carrinho fique leve, de forma a aumentar sua aceleração linear.

A fim de reduzir o momento de inércia, é válido relembrar sua definição:

$$I = \sum (m_i \cdot r_i^2)$$

A relação nos diz que o momento de inércia em relação à um dado eixo de rotação é dado pela soma da massa de cada ponto do carrinho ponderada pela distância ao quadrado em relação ao eixo de rotação. A Figura **??** ajuda a visualização da relação. O eixo de rotação, por aproximação, pode ser considerado o centro do eixo dos motores. Em outras palavras, se as maiores massas estiverem concentradas em pontos os quais a distância r em relação ao eixo de rotação é pequena, então o resultado do somatório será pequeno e consequentemente, o momento de inércia. Essa análise leva a conclusão de que **a maior parte da massa do carrinho deve estar concentrada próxima ao seu eixo de rotação**.

Com isso, foram definidos dois parâmetros que aumentam a eficiência mecânica do seguidor de linha. Um bom projeto mecânico é essencial para o sucesso na elaboração de um seguidor de alta eficiência.

Adicionalmente, é de extremo interesse que o chassi seja bastante baixo, devido à necessidade que a placa de sensores esteja próxima ao solo. Este tópico será melhor abordado nas seções seguintes.

2.2 Rodas e Motores

Em primeiro lugar, rodas e motores devem respeitar os critérios estabelecidos na seção anterior. Componentes pesados são empecilhos, e devem ser evitados.

Boas rodas são leves e possuem um coeficiente de atrito razoável. Um material leve com um bom coeficiente de atrito é a borracha, por exemplo. As rodas também não podem ser muito altas, afim de respeitar o critério de que o chassi deve permanecer baixo e rente ao solo.

Bons motores também são essenciais para o desenvolvimento de um seguidor de linha de alta eficiência. Motores de qualidade possuem uma característica linear e previsível de torque e velocidade. Em contrapartida, motores mais baratos geralmente usados para a construção do carrinho apresentam características muitas vezes não linear e imprevisível de torque e velocidade, principalmente em baixas rotações. Esse aspecto prejudica a implementação do algoritmo de controle, uma vez que os mesmos valores de PWM, e consequentemente de tensão aplicada nos motores podem gerar comportamentos totalmente inesperados.

O principal obstáculo no que tange os motores, no entanto, é financeiro. Motores de qualidade maior são consideravelmente mais caros. Os motores usualmente utilizados fornecem limitações

diversas, mas muitas vezes, são as opções mais financeiramente viáveis. Embora seja difícil refinar o algoritmo de controle, ainda é possível implementa-lo com motores de qualidade inferior.

2.3 Placa de Sensores

A qualidade de aquisição dos sinais é um fator fundamental para o sucesso do projeto. Projetistas com alguma experiência sabem que não se programa um carrinho cujos sensores não funcionam.

A primeira decisão a ser tomada é a escolha do sensor infravermelho utilizado. A Figura 2.1 mostra o sensor mais comummente utilizado em bons projetos de placa. Esse sensor já vem com o par emissor-receptor encapsulado, diminuindo consideravelmente as interferências exteriores no sistema.

Figura 2.1: Sensor Infravermelho Encapsulado

Outro ponto é importante é fazer com que os sensores estejam na menor distância possível um do outro. Como veremos adiante, sensores menos espaçados possibilitam uma frequência de amostragem de erro maior, tornando a resposta ao algoritmo de controle cada vez mais suave e próxima de um controle totalmente analógico. Em outras palavras, sensores menos espaçados tornam o movimento do carrinho mais suave.

Uma condição segura para o sucesso do algoritmo é e que o espaçamento entre sensores seja **no mínimo, menor que a espessura da linha a ser seguida**. Essa condição pode ser visualizada na Figura 2.2 e será esclarecida durante a discussão sobre o algoritmo PID.

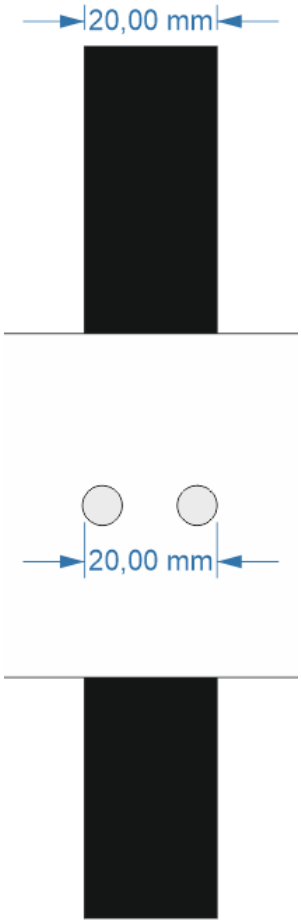


Figura 2.2: Espaçamento entre sensores

2.4 Circuitos Dedicados

Outra possível alternativa para otimizar o carrinho seguidor é a substituição do micro-controlador por um sistema totalmente dedicado e analógico. Ou essa solução tomada parcialmente, isto é, a substituição do Arduíno por controladores mais rápidos e computacionalmente menos complexos. Os seguidores de mais alta-performance utilizam circuitos dedicados e atingem altíssimas velocidades, uma vez que não gastam tempo com conversões analógico-digital e todo o tempo de processamento envolvido em um sistema digital.

A Figura 2.3 mostra um seguidor de linha simples sem o uso de microcontroladores, totalmente analógico. Sua resposta será certamente mais rápida, no entanto, quanto mais complexo o algoritmo adotado, mais complexo será o circuito.

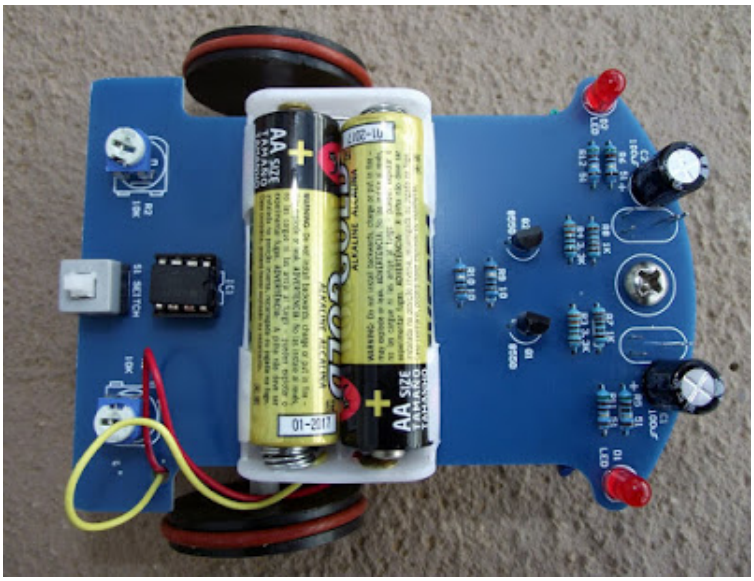
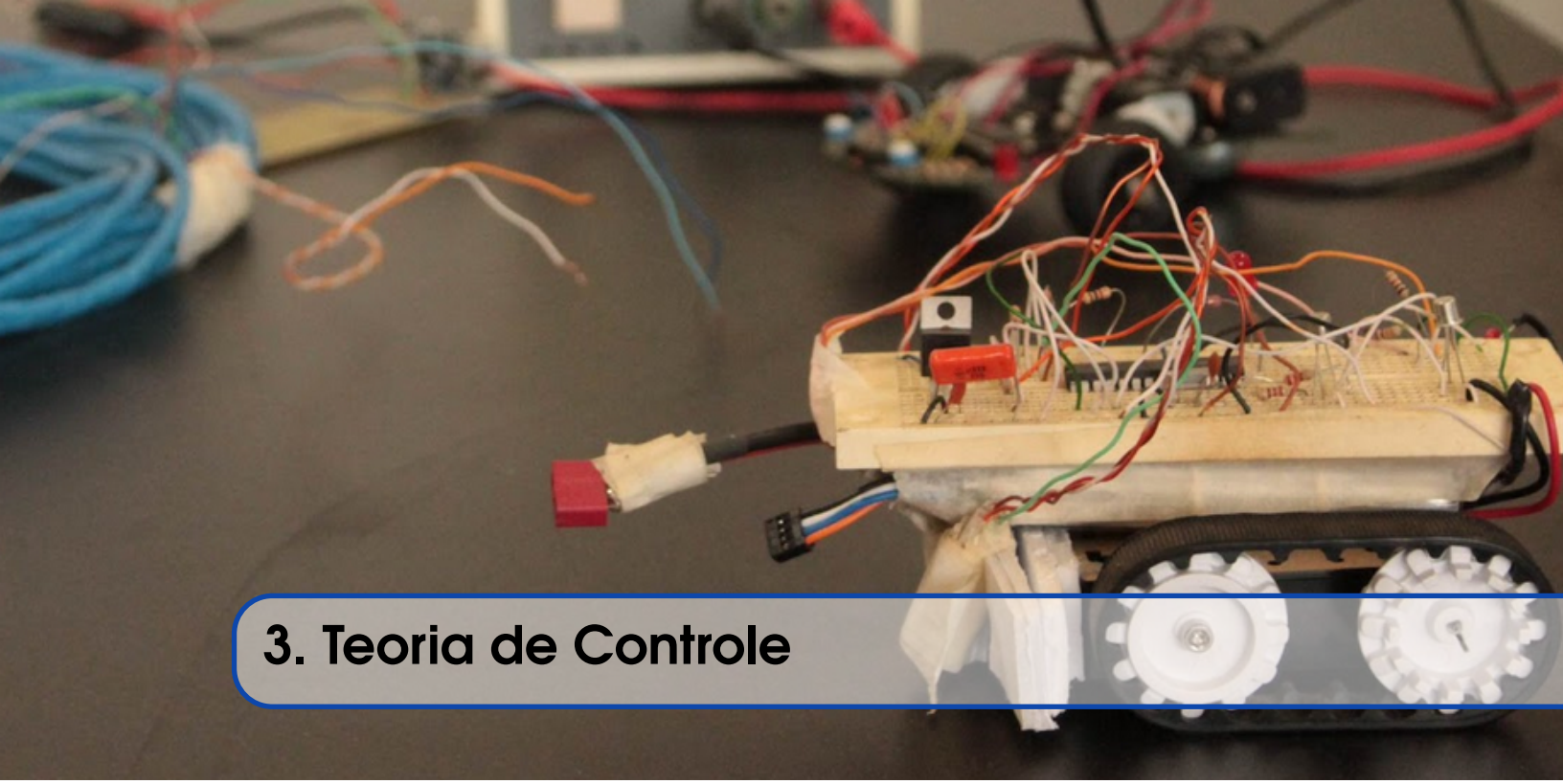


Figura 2.3: Robô Analógico



3. Teoria de Controle

Teoria de Controle trata-se de uma sub-área da Engenharia Elétrica que lida com o comportamento de sistemas dinâmicos. Um controlador é utilizado para influenciar no sistema de modo a manter as grandezas em questão em um dado valor de desejado (valor de referência)

3.1 Conceitos

Para a fixação dos conceitos, serão utilizados 3 exemplos:

- Um forno o qual deseja-se manter a temperatura constante em 100ºC;
- Um reservatório de água alimentado por uma bomba o qual deseja-se manter o nível constante;
- Um carrinho seguidor de linha.

A Figura 3.1 mostra uma malha de controle fechada. O conceito de malha será melhor explorado nas seções seguintes. Por enquanto, apenas observa-se as variáveis presentes na malha para a associação de conceitos.

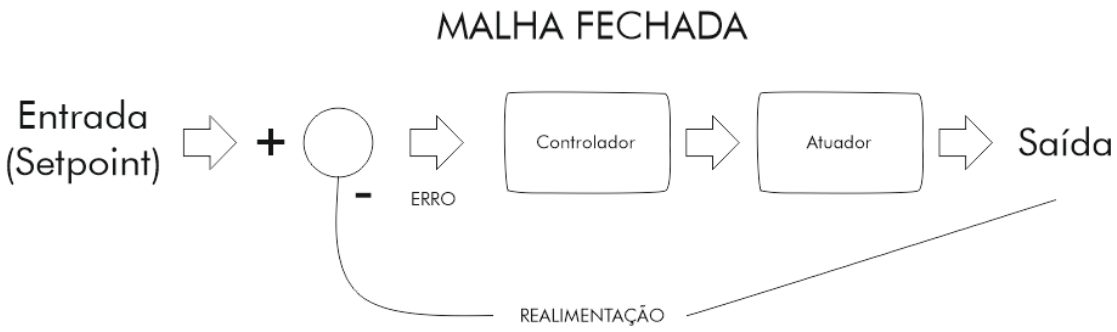


Figura 3.1: Malha de controle fechada

3.1.1 Variável controlada

Variável controlada é a grandeza sob a qual deseja-se agir para influenciar o estado do sistema. A Tabela 3.1 indica qual é a variável controlada no contexto de cada exemplo.

Tabela 3.1: Variável controlada

	Variável controlada
Forno	Temperatura do Forno
Reservatório	Nível da água
Seguidor de Linha	Posição do carrinho

3.1.2 Referência ou Setpoint

É o valor desejado para a variável de controle. No caso do forno, pode ser que se queira manter sua temperatura constante à 100ºC, por exemplo. A Tabela 3.2 mostra exemplos de possíveis *setpoints* para os exemplos aqui tratados.

Tabela 3.2: Setpoint

	Variável controlada	**Setpoint**
Forno	Temperatura do Forno	100ºC
Reservatório	Nível da água	Metade
Seguidor de Linha	Posição do carrinho	Centro da Linha

Nota-se que neste ponto ainda não é necessário definir o setpoint quantitativamente. Como o setpoint será alcançado depende de como o sistema em questão fará a aquisição de dados, ou, em outras palavras, o sensoriamento, ou realimentação.

3.1.3 Sensoriamento ou Realimentação

A **malha de realimentação** é a parte do sistema responsável por adquirir os dados sobre a variável controlada. Fisicamente, é representada pelos sensores envolvidos no processo. Neste ponto, torna-se necessário definir quantitativamente a variável que deseja-se adquirir, afim de especificar o sensor utilizado em um projeto. A Tabela 3.3 exemplifica possíveis sensores para os exemplos trabalhados.

Tabela 3.3: Sensoriamento

	Variável controlada	**Setpoint**	**Sensoriamento**
Forno	Temperatura do Forno	100ºC	Sensor de Temperatura
Reservatório	Nível da água	Metade	Sensor de Nível
Seguidor de Linha	Posição do carrinho	Centro da Linha	Par receptor / emissor infravermelhos

Em alguns sistemas, é necessário avaliar qual o efeito da inclusão de um sistema de sensoriamento no próprio sistema. Isto é, os sensores de alguma forma influenciam no comportamento do sistema ?

No carrinho seguidor de linha, por exemplo, a informação provinda dos sensores é um conjunto de valores de tensão sobre os receptores infravermelhos. Cada valor de tensão é traduzido em um valor de 0 a 1023 por um conversor Analógico-Digital presente no Arduíno. As questões a serem respondidas são: **como podemos traduzir esses valores em informação acerca da posição do carrinho** e **como podemos definir o setpoint em termos destes valores** ?

Para responder essas questões, é necessário definir um conceito de extrema importância, o **erro**.

3.1.4 Erro

Erro é o nome dado para a variável que representa a diferença entre o valor atual de saída e o setpoint desejado.

Em alguns sistemas, como o caso do forno, o conceito de erro torna-se evidente. Se queremos que o forno tenha uma temperatura constante de 100ºC, quando sua temperatura for 103ºC, por exemplo, o erro atual será de +3ºC. Caso a temperatura seja corrigida pelo controlador para 2ºC, o erro agora é de -2ºC.

No entanto, em uma variedade de outros sistemas, como o seguidor de linha, a concepção do erro não é trivial. Definir um modelo digital (ou um código) para especificar o erro é um dos maiores desafios de projeto.

A variável que deseja-se controlar é a posição do carrinho. Como é possível fazer isso através da informação provinda da régua de sensores? Sabe-se que para esta aplicação, a informação de cada sensor é de natureza lógica, isto é, cada sensor **pode ou não** estar detectando a linha. A posição ideal é aquela na qual apenas o sensor do meio (para uma régua com número ímpar de sensores) detecta a linha. Logo, uma possível solução é definir o erro como sendo um valor numérico que representa a diferença entre a posição ideal.

Por exemplo, se apenas o sensor do meio detecta a linha, definimos o erro como sendo 0. Caso o sensor mais a direita detecte a linha ao mesmo tempo que o sensor do meio, significa que o carrinho está desviando levemente de sua posição ideal, ou seja, existe erro no sistema, pode-se definir essa

situação como erro de valor 1. Agora, caso apenas o sensor a direita do sensor central detecte a linha, o erro é maior que a situação anterior, e pode-se atribuir valor 2 ao erro. E assim por diante.

O cálculo do erro será melhor discutido no capítulo seguinte, no qual discute-se detalhadamente o algoritmo de controle adotado para o carrinho seguidor de linha.

3.1.5 Controlador

O Controlador é a parte principal de qualquer sistema. Ele representa a "parte pensante"do sistema, o responsável por controlar de fato o sistema físico.

Existem dois tipos de controladores: analógicos e digitais.

Controladores analógicos são constituídos por circuitos que desempenham alguma função matemática desejada. Por exemplo: somar sinais, subtrair, amplificar, integrar e diferenciar. Com uma gama de funções matemáticas tem-se uma unidade lógico aritmética capaz de agir sob o sistema. No entanto, controladores analógicos estão cada vez mais perdendo espaço para controladores digitais, tornando-se obsoletos.

Controladores digitais, por definição, são algoritmos implementados por computadores. Esses computadores também podem ser de tipos variáveis. Sistemas dinâmicos e móveis, como é o caso da robótica, requerem controladores menores, práticos. Computadores desenvolvidos com esse intuito são chamados de **microcontroladores**, **como é o caso do arduíno**. Sistemas industriais geralmente utilizam controladores maiores, que implementam algoritmos específicos em linguagens de mais baixo nível.

A Figura 3.2 mostra um PLC, controlador digital utilizado em aplicações industriais. Nota-se a diferença de mobilidade e tamanho entre um PLC e um microcontrolador, como o Arduíno.

Figura 3.2: Controlador industrial

Deve-se definir qual algoritmo de controle será utilizado e como ele será implementado. **O algoritmo de controle** é diferente do **controlador**. No seguidor de linha, o algoritmo de controle usado é um **PID** (*proportional, integral and differential*) e este é implementado em forma de código no Arduíno.

Alguns elementos são importantes no que tange controladores digitais, como por exemplo taxa de amostragem, conversores A/D e D/A, memória, etc. No entanto, para o contexto do seguidor de linha, estes conceitos não são tão relevantes.

3.1.6 Atuadores

Atuadores são os mecanismos físicos que permitem que o controlador **atue** sobre o sistema. A Tabela 3.4 mostra alguns possíveis atuadores para os exemplos aqui discutidos.

Tabela 3.4: Atuadores

	Atuador
Forno	Aquecedor magnético
Reservatório	Bomba d'água
Seguidor de Linha	Motores

Assim como no sensoriamento, uma avaliação sobre o impacto dos atuadores no próprio sistema é necessária. Na verdade, o controlador costuma interferir significativamente nos sistemas controladores, logo, deve-se levar em conta sua presença na hora de definir o controlador, o algoritmo e seus parâmetros.

3.2 Resposta Temporal

A maioria dos sistemas físicos de interesse para Engenharia de Controle são regidos por equações diferenciais de segunda ordem, incluindo o seguidor de linha. Quando excitamos a entrada do sistema com um degrau, todos esses sistemas respondem de maneira similar.

A Figura 3.3 mostra um gráfico que representa a resposta temporal do regime transitório dos sistemas de 2ª ordem. Regime transitório é o período que procede a perturbação no sistema. A situação do seguidor de linha é por si própria dinâmica ao ponto de haver perturbação constante. Caso o carrinho não encontrasse curvas, ele entraria em um modo que chamamos de **estado estacionário**. O estado estacionário é um estado no qual não existem muitas perturbações, exigindo menos desempenho do algoritmo de controle. Logo, **devemos projetar o controlador para atender especificações de regime transitório**.

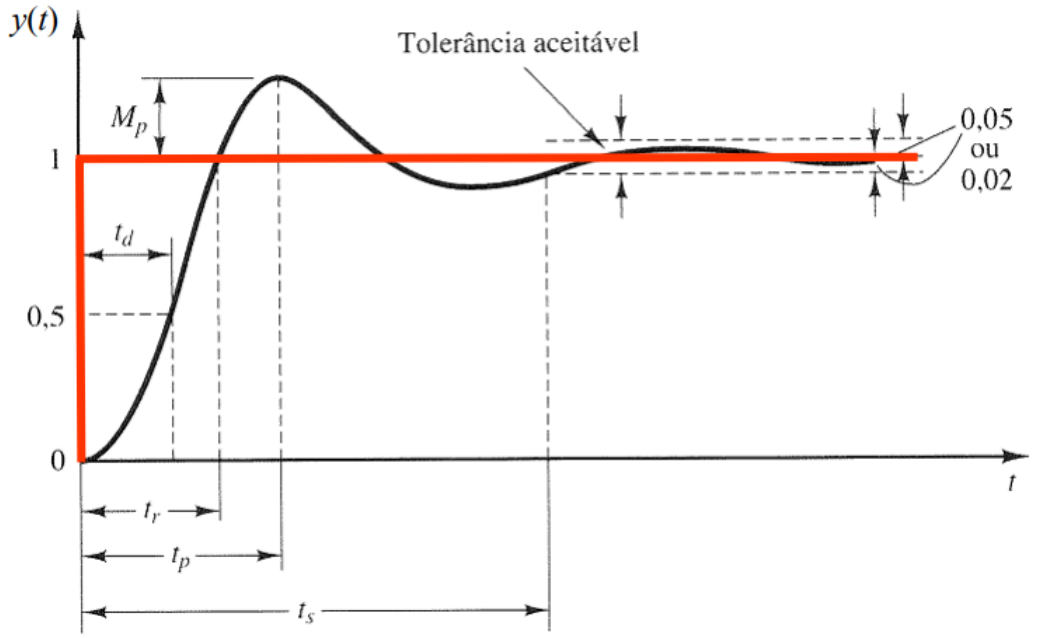


Figura 3.3: Resposta temporal do sistema de 2ª Ordem

Os parâmetros mais importantes desse gráfico são:

- **y(t)** é a variável controlada. No caso do seguidor de linha, y(t) é a posição desejada do carrinho;
- A linha vermelha horizontal é a **referência** ou **setpoint** desejada para o sistema, ou setpoint;
- **Mp** ou **pico** é o valor de posição mais distante da referência que o carrinho atinge;
- **tp** ou **tempo de pico** é o tempo que o carrinho demora para atingir esse pico após a perturbação;
- **ts** ou **tempo de acomodação** é o tempo que o sistema demora para se estabilizar após a perturbação;
- **Tolerância aceitável** ou **erro de estado estacionário** é a quantidade aceitável que o carrinho pode variar sem perturbações no sistema.

Um sistema bem controlado possui baixo valor de pico, erro de estado estacionário nulo ou pequeno, tempo de acomodação pequeno, e tempo de pico também baixo. O objetivo do controlador PID é agir sob todos esses parâmetros de forma a otimizar a reposta.

Neste ponto, **cabe uma importante observação**. Projetos de controladores reais devem respeitar critérios de projeto rigorosos e são feitos através de formulações matemáticas complexas. No entanto, esse não é o objetivo desta apostila. Aqui, a relação entre os parâmetros da resposta temporal e o controlador PID será discutida somente qualitativamente.

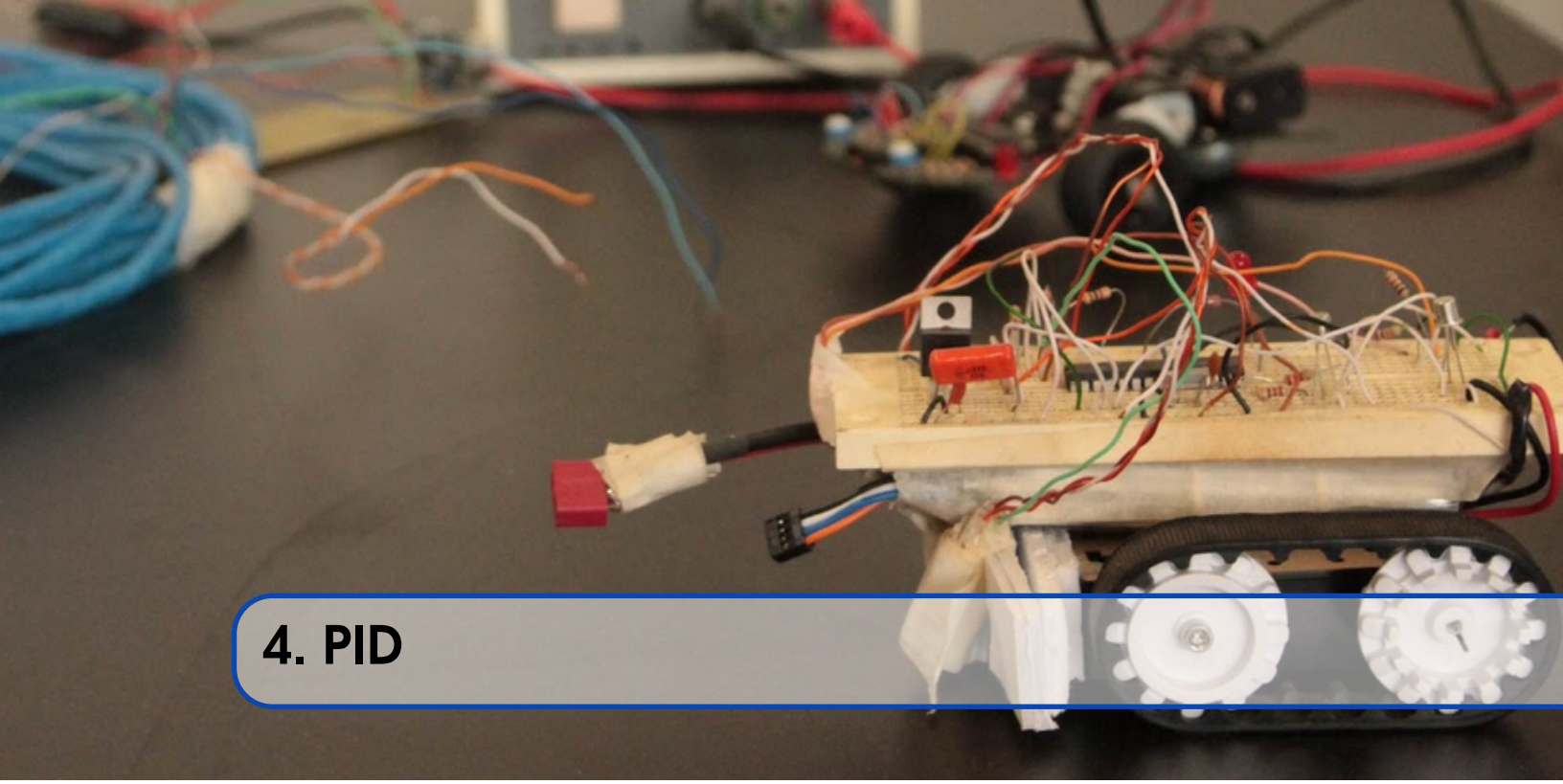
3.3 Malha Aberta e Malha Fechada

Outro conceito importante em Engenharia de Controle e que será discutido no contexto do Seguidor de Linha é a diferença entre malha fechada e malha aberta.

Os conceitos anteriormente discutidos são referentes à malha de controle fechada. No entanto, o que é uma malha de controle fechada?

A **malha fechada** é um sistema realimentado. Ou seja, é um sistema que adquire informações da saída através de sensores e utiliza essas informações para calcular o erro e realizar a ação de controle necessária. Por outro lado, a **malha aberta** é um sistema que não possui realimentação. Mais detalhadamente, o controle em malha aberta consiste em aplicar um sinal de controle na entrada de um sistema, esperando-se que na saída a variável controlada consiga atingir um determinado valor ou apresente um determinado comportamento desejado.

No seguidor de linha, quando o carrinho passa por um trecho desafio, como curvas de 90º ou rotatórias, muitas vezes é adotada uma **solução em malha aberta**. Em outras palavras, por um determinado trecho, o carrinho não mais seguirá conforme a informação dos sensores e o algoritmo de controle. Ele será forçado por um determinado trecho ou caminho não importa qual seja a leitura dos sensores. Sempre que uma solução desse tipo for adotada, dizemos que o carrinho está **funcionando em malha aberta**.



4. PID

4.1 Introdução

O **PID** (*Proportional Integral and Differential*) controller é uma categoria de controlador usado em uma vasta gama de aplicações atuais. O PID é muito utilizado tanto na robótica quanto em aplicações industriais. Seu funcionamento começa pelo cálculo contínuo do erro na malha fechada, conceito discutido no capítulo anterior. Então, realizam-se 3 operações matemáticas sobre o erro calculado, conforme a equação abaixo. A ação final do PID será a soma algébrica das correções obtidas nas 3 operações.

$$PID = K_p \cdot e(t) + K_i \cdot \int e(t)dt + K_d \cdot \frac{de(t)}{dt}$$

O primeiro termo da equação é a parcela de **controle proporcional**. Quanto maior for o erro no instante t atual maior será a correção do controlador. Este tipo de controle também é chamado de **ganho simples**.

O segundo termo é o parcela de **controle integral**. Sabendo que uma integral é uma soma, deduz-se que o controlador integral corrige proporcionalmente ao erro acumulado. O Controlador Integral fornece estabilidade para o sistema, além de diminuir, ou em alguns casos até anular o erro de estado estacionário. Ele também é responsável por acelerar a resposta do sistema.

O último termo é a parcela relativa ao **controle derivativo**. O controlador derivativo age conforme a velocidade de variação do erro. Ou seja, em situações onde o erro varia muito rapidamente (como é o caso do seguidor de linha), o controlador derivativo atua.

Os valores K_p,K_d e K_i são chamados de **constantes do PID**, e sua função é ponderar o valor de cada correção. Existem também métodos matemáticos para encontrar valores ótimos de constantes, no entanto, em nossa aplicação, adotaremos o teste empírico, que consiste em fazer diversos testes até encontrar valores funcionais.

Os efeitos sobre a respsota temporal e sobre o comportamento do carrinho serão discutidos nas seções seguintes.

4.2 OK, mas como fazer?

Neste ponto, deve-se fazer uma importante observação.

Uma dúvida frequente no que concerne o desenvolvimento do projeto é **como transformar toda a teoria em um carrinho seguidor de linha**.

Os tópicos referentes à Engenharia de Controle são complexos e requerem formulação matemática complexa e rigorosa, além de diversos métodos de projeto. No entanto, o uso do microcontrolador muito nos ajuda. Trata-se de um controlador digital implementado no Arduíno. **O real desafio reside em programar um código que realize o cálculo do erro e as operações matemáticas sobre ele**.

Em um sistema digital, o sinal de interesse passa por um processo de **amostragem**, e esse sinal passa a ter uma característica discreta no tempo. A equação do PID, discutida acima, pressupõe um sinal contínuo no tempo. Caso queiramos desenvolver o PID em um sistema discreto, tal como um computador, é necessário trabalhar com uma expressão matemática igualmente discreta. A equação abaixo descreve o **PID digital**.

$$PID = K_p \cdot e[k] + K_i \cdot \sum_{n=0}^{k} e[n] + K_d \cdot (e[k] - e[k-1])$$

Deste ponto em diante, **a solução será tratada em termos de código**.

4.3 Definição do Erro

Como discutido anteriormente, o primeiro desafio de implementação reside no cálculo do erro. Esse desafio por ser solucionado de diversas formas. Pode-se calcular o erro como sendo a diferença de velocidade linear entre os dois motores, ou então o ângulo entre os sensor central e o sensor atualmente detectado. No entanto, em código desenvolvido pela equipe **PETEE** trata-se o erro de maneira mais elegante. O erro é definido da seguinte forma:

- Um valor positivo caso o o carrinho esteja indo para a direita
- Um valor negativo caso o carrinho esteja indo para a esquerda

O erro nulo acontece quando somente o sensor mais centralizado detecta a linha. De forma progressiva, caso os sensores adjacentes detectem a linha, o erro sobe. A Tabela 4.1 mostra o funcionamento do algoritmo de cálculo do erro, para uma régua de 7 sensores. A sigla "SC"significa "Sensor Central", "SE"significa "Sensor Esquerdo"e "SD"significa "Sensor Direito".

Tabela 4.1: Cálculo do Erro

Erro	SE3	SE2	SE1	SC	SD1	SD2	SD3
0	0	0	0	1	0	0	0
1	0	0	0	1	1	0	0
2	0	0	0	0	1	0	0
3	0	0	0	0	1	1	0
4	0	0	0	0	0	1	0
5	0	0	0	0	0	1	1
6	0	0	0	0	0	0	1
0	0	0	0	1	0	0	0
-1	0	0	1	1	0	0	0
-2	0	0	1	0	0	0	0
-3	0	1	1	0	0	0	0
-4	0	1	0	0	0	0	0
-5	1	1	0	0	0	0	0
-6	1	0	0	0	0	0	0

O código abaixo mostra uma função desenvolvida para cálculo do erro do caso descrito acima. A função `readSensors();` não terá seu código explicitado aqui, sua função é realizar a chamada dos comandos `DigitalRead` ou `AnalogRead`.

```
void calculaErro()
{
  readSensors();
  if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==HIGH && LF[4]==LOW &&
   ↪  LF[5]==LOW && LF[6]==LOW)
    erro = 0;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==HIGH&&
   ↪  LF[4]==HIGH && LF[5]==LOW && LF[6]==LOW)
    erro = 1;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==HIGH && LF[3]==HIGH &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
    erro = -1;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW &&
   ↪  LF[4]==HIGH && LF[5]==LOW && LF[6]==LOW)
    erro = 2;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==HIGH && LF[3]==LOW &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
    erro = -2;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW &&
   ↪  LF[4]==HIGH && LF[5]==HIGH && LF[6]==LOW)
    erro = 3;
  else if(LF[0]==LOW && LF[1]==HIGH && LF[2]==HIGH && LF[3]==LOW &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
```

```
    erro = -3;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW && LF[4]==LOW
   ↪  && LF[5]==HIGH && LF[6]==LOW)
    erro = 4;
  else if(LF[0]==LOW && LF[1]==HIGH && LF[2]==LOW && LF[3]==LOW &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
    erro = -4;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW && LF[4]==LOW
   ↪  && LF[5]==HIGH && LF[6]==HIGH)
    erro = 5;
  else if(LF[0]==HIGH && LF[1]==HIGH && LF[2]==LOW && LF[3]==LOW &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
    erro = -5;
  else if(LF[0]==LOW && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW && LF[4]==LOW
   ↪  && LF[5]==LOW && LF[6]==HIGH)
    erro = 6;
  else if(LF[0]==HIGH && LF[1]==LOW && LF[2]==LOW && LF[3]==LOW &&
   ↪  LF[4]==LOW && LF[5]==LOW && LF[6]==LOW)
    erro = -6;
}
```

Uma vez definido o erro do sistema em malha fechada, podemos **desenvolver funções que calculem o PID**.

4.4 Cálculo do PID

O passo seguinte consiste em utilizar o erro definido anteriormente para calcular o PID. Antes de avançar, uma importante consideração deve ser feita. **Sobre o que o PID atua**?

Como explicado no capítulo de conceitos, o controlador atua sobre atuadores. No caso do seguidor de linha, os atuadores são os motores do carrinho. Felizmente, o Arduíno em conjunto com a ponte H fornecem um método fácil e eficiente para controlar a velocidade dos motores, a modulação PWM, abordada na Oficina 1.0. Logo, **o valor final calculado pelo PID** deve ser somado ou subtraído da velocidade dos motores. Percebe-se que em condições de não perturbação, ou seja, de erro 0, a velocidade do dois motores é mentida igual, e o PID tem a tarefa de subtrair um valor

Antes de prosseguir, outro ponto ainda é importante. **Na solução aqui descrita, cada motor é controlado de maneira separada**. Isto é, quando o erro é nulo, a velocidade dos dois motores é mantida igual. Quando o erro se torna positivo (curva para a direita) o PID é acionado sobre a roda direita. Quando o erro é negativo (curva para a esquerda), o PID é acionado sobre a roda esquerda, de forma a corrigir a trajetória do robô.

```
void calculaPID()
{
   if(erro==0)
    integral = 0;
   prop = erro;
```

```
    integral = integral + erro;
     if(integral > 255)
       integral = 255;
     else if(integral < -255)
       integral = -255;
    derivativo = erro - U_erro;
    PID = ((Kp * prop) + (Ki * integral) + (Kd * derivativo));
    U_erro = erro;
}
```

No código acima nota-se o controle da parcela integral. Toda vez que o erro se tornar nulo, a parcela relativa ao controlador integral também deve ser zerada. Um limite máximo também deve ser estabelecido. Como o controle integral é baseado em acumulação, caso o erro persista, ele pode chegar a valores extremamente altos, fato este indesejado. Logo, é necessário estabelecer limites superiores para a correção integral. Esses limites foram definidos como **os valores máximos do PWM**.

Por fim, a variável `PID` faz a soma de todas as parcelas e seu valor é aplicado sobre a velocidade dos motores.

O código abaixo mostra a aplicação do PID sobre os motores.

```
    if(PID>=0) //VIRA PARA A DIREITA
    {
      velesq = avgSpeedESQ;
      veldir = avgSpeedDIR - PID;
    }
    else //VIRA PARA A ESQUERDA
    {
       velesq = avgSpeedESQ + PID;
       veldir = avgSpeedDIR;
    }
```

4.5 Efeito do controlador sobre a resposta temporal

Neste ponto, o leitor provavelmente se pergunta: *Ok, mas qual o efeito real do PID sobre meu carrinho*? Não se inquiete, prezado leitor.

O PID, como mencionado anteriormente, tem como objetivo melhorar os parâmetros de regime transitório da resposta temporal. Em outras palavras, ele otimiza os parâmetros como pico (*Overshoot*), tempo de acomodação, erro de estado estacionário, etc.

4.5.1 Proporcional

A Figura 4.1 mostra o efeito do controlador proporcional sobre um sistema de segunda ordem, tal como o seguidor de linha.

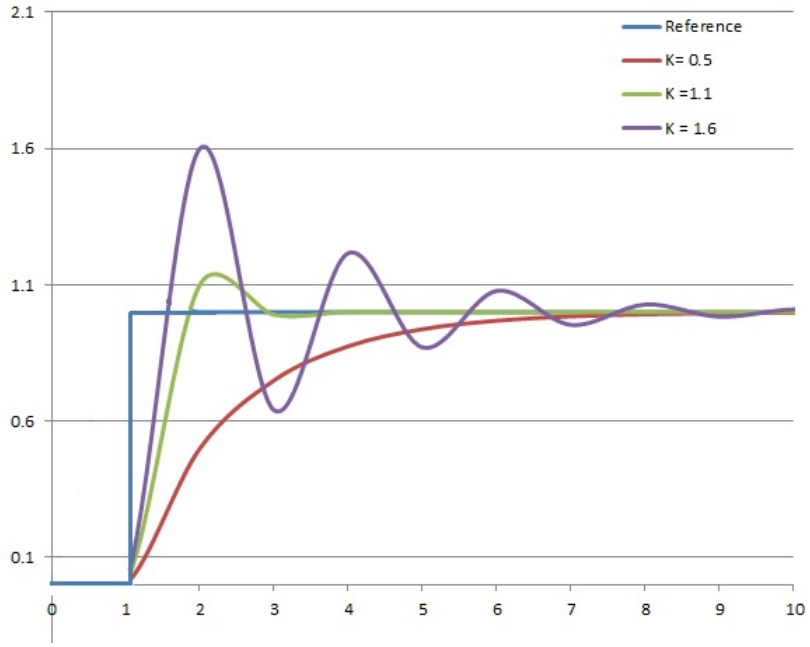


Figura 4.1: Efeito do controle proporcinoal sobre o sistema

Repara-se que um valor de Kp cada vez mais alto **diminui o overshoot, tempo de acomodação e tempo de pico** e não afeta o erro de estado estacionário. No entanto, **valores muitos altos de ganho proporcional geram alta instabilidade no sistema**, podendo desestabilizar facilmente o carrinho de sua trajetória. O controle proporcional é considerado o ganho bruto do PID e sua utilização é essencial, mas a **constante proporcional Kp** deve ter seu valor moderado pelo projetista.

4.5.2 Integral

A Figura 4.2 mostra o efeito do controlador integral sobre um sistema de segunda ordem, tal como o seguidor de linha.

O controlador integral tem como principal função **eliminar o erro de estado estacionário, e diminuir significativamente o tempo de acomodação**. No entanto, como a correção integral se baseia em erros passados, a mesma tem como efeito o aumento do *Overshoot* e aumento da instabilidade do sistema.

4.5.3 Diferencial

A Figura 4.3 mostra o efeito do controlador diferencial sobre um sistema de segunda ordem, tal como o seguidor de linha.

O controlador derivativo, ou diferencial, prevê o erro no sistema, logo, tem como efeito a **a diminuição do tempo de acomodação, aumento da estabilidade do sistema**.

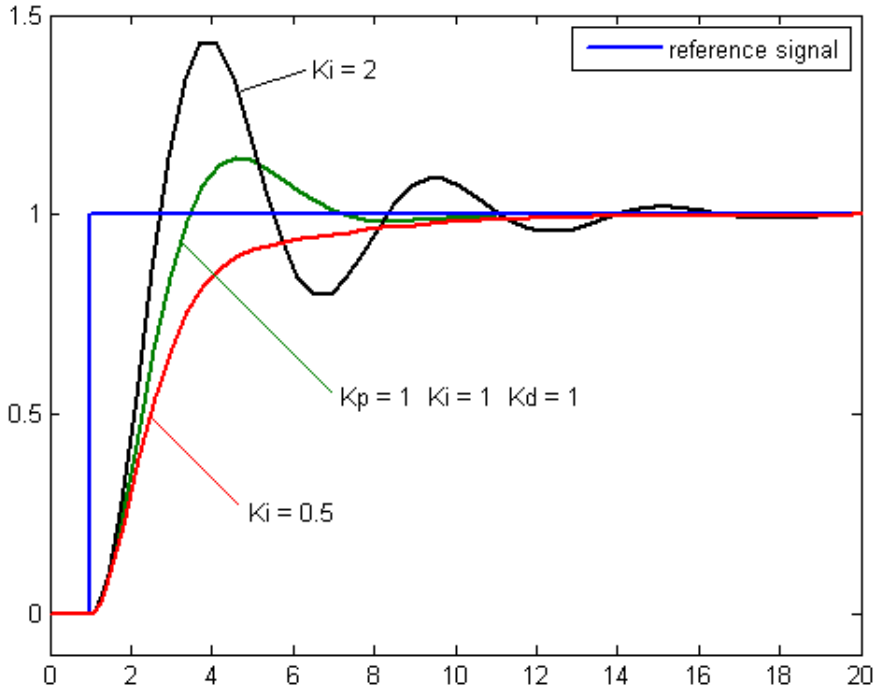


Figura 4.2: Efeito do controle integral sobre o sistema

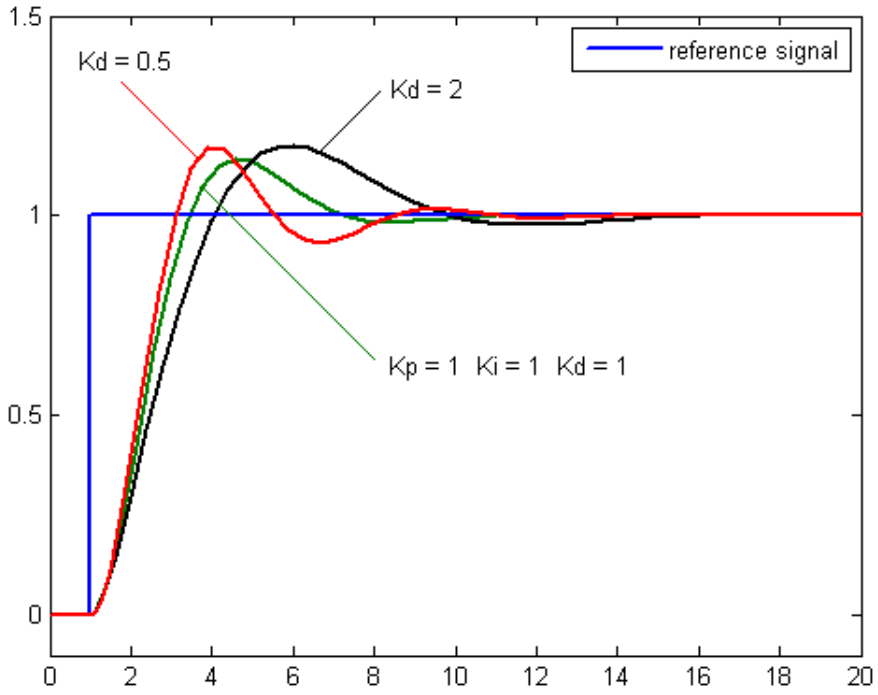


Figura 4.3: Efeito do controle derivativo sobre o sistema

Tabela 4.2: Efeitos de cada parcela do PID nos parâmetros da resposta transitória

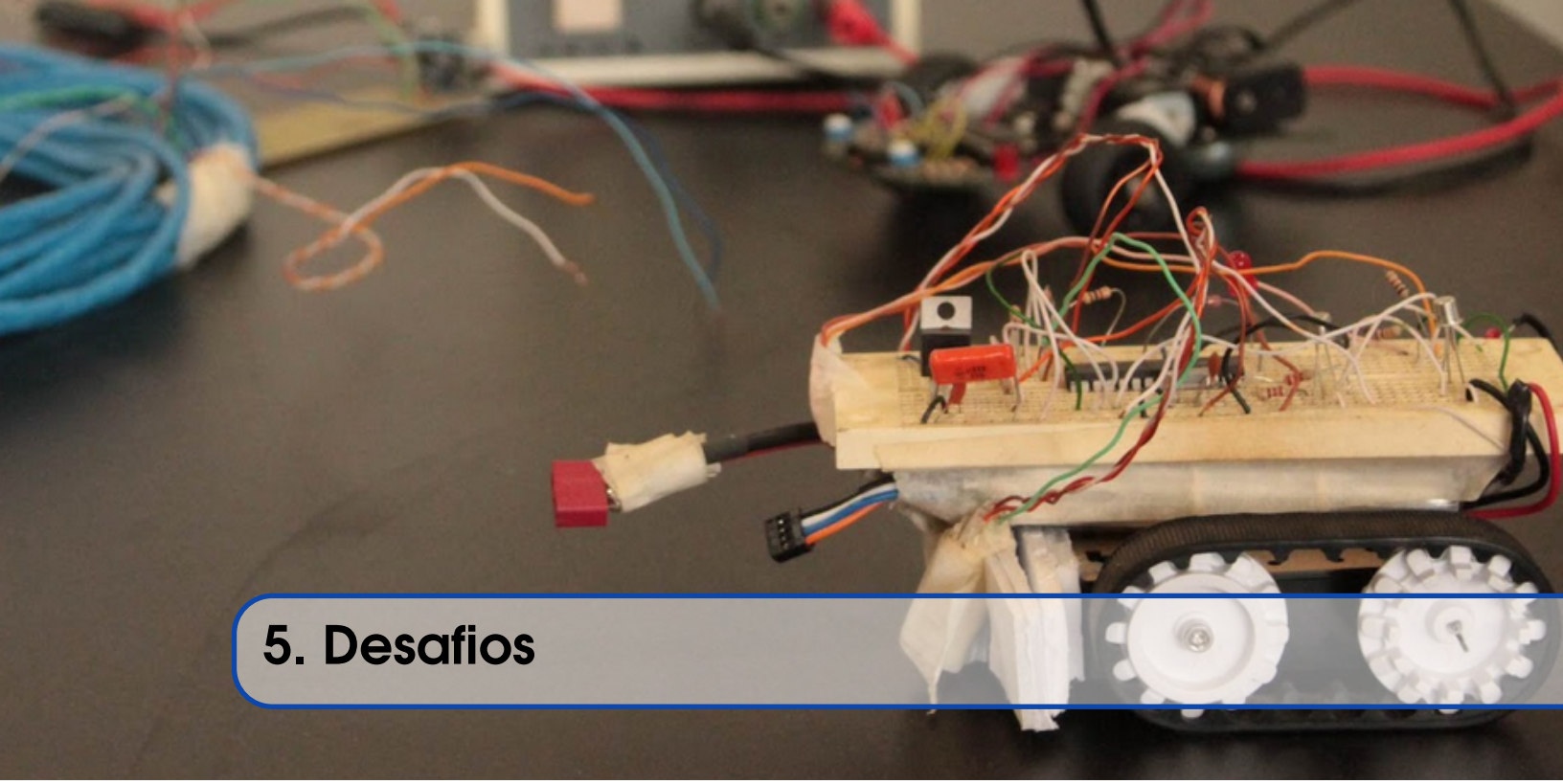
	Overshoot (Pico)	Tempo de acomodação	Erro de estado estacionário	Estabilidade
Proporcional	Diminui	Diminui	Aumenta	Diminui
Integrador	Aumenta	Diminui	Diminui	Diminui
Derivativo	Diminui	Diminui	Não altera	Aumenta

4.5.4 Ajuste de constantes

Por fim, a Tabela 4.2 resume os efeitos de cada termo do controlador.

Uma vez implementadas todas as parcelas do PID, o desafio passa a ser o valor ótimo para as contantes Kp, Ki e Kd. Cada carrinho apresenta uma característica diferente. Motores utilizados nessa aplicação muitas vezes apresentam características imprevisíveis. Os valores de constantes que funcionam para um carrinho são totalmente diferentes dos que funcionam para outro carrinho.

O real trabalho neste ponto reside na realização de diversos testes até que se encontrem valores ótimos.



5. Desafios

5.1 Curva de 90º

Na grande maioria de competições de Seguidores de Linha existentes, curvas de 90° são tratadas como um nível maior de dificuldade, e por isso devem ser sinalizadas. Algumas dessas competições, inclusive a CoRA, sinaliza essa dificuldade com um quadrado, ao lado do percurso, incidando para qual lado o robô deverá virar. Na 5ª edição da CoRA, isso é feito da seguinte forma:

"A identificação das curvas de 90° conta com uma marca branca de 5,0 x 5,0 cm, a 3,0 cm da linha do percurso, no lado de ocorrência da curva, a 5,0 cm da mesma. Vale ressaltar que, durante um percurso, poderão ocorrer indicações de curvas de 90º tanto de conversão à direita, quanto à esquerda."

Para realizar esse obstáculo é necessário uma boa integração entre a placa de sensores e a programação. É fundamental que os desenvolvedores do seguidor de linha projetem sua placa de sensores com um posicionamento de sensores que permita identificar com clareza esse tipo de obstáculo. Uma boa dica é separar um ou dois sensores para fazer essa atividade.

Existem diversas formas de realizar esse tipo de obstáculo. Abaixo são abrangidos dois deles.

- A primeira forma, é realizar a curva assim que o(s) sensor(es) de dificuldade detectar o quadrado. Quando isso ocorrer, o código deverá chamar uma função que realiza o procedimento de virar para o lado correspondente a marcação. Esse procedimento, então, diminui a rotação do motor do lado em que se deseja fazer a curva e coloca uma rotação maior no motor do lado oposto. Isso pode ser feito de duas formas diferentes, a primeira é permanecer realizando esse processo durante um tempo testado (através do procedimento delay(), já implementado nas bibliotecas padrão do Arduino) para que o robô complete a curva com segurança e depois

volte a seguir linha; e a segunda é permanecer no processo de virar até que o(s) sensor(es) central(centrais) voltem a detectar a linha necessária para seguir o percurso (esse processo é mais complexo que o anterior, mas ambos são eficazes se bem implementados);

- A segunda maneira consiste em só chamar o procedimento de curva de 90° após o(s) sensor(es) de dificuldade detectarem o quadrado e em seguida todos os sensores do lado respectivo a curva detectarem simultaneamente uma linha branca. Então deve-se parar totalmente o motor do lado em que será feita a curva e colocar uma rotação suficiente no outro motor para que ele faça a conversão. Assim como no caso anterior, tem-se as duas opções para realizar esse processo (utilizando delay ou permanecer no procedimento até que o robô encontre a linha que ele deverá seguir). Para robôs com uma velocidade avançada, apenas atribuir uma rotação zero para um dos motores não é suficiente para que ele consiga fazer a curva com segurança (pois como o motor demora a parar de fato, pode ocorrer do seguidor de linha acabar saindo da pista). Nesses casos, a técnica empregada é aplicar uma rotação reversa, por um breve período, no motor que deseja-se travar, pois então ele irá parar quase que instantaneamente e será possível a realização da curva.

Abaixo, na Figura 5.1, pode ser visto procedimentos para realizar curva de 90° para cada um dos dois lados. Os valores de rotação aplicados aos motores, bem como os delays, foram encontrados atarvés de testes.

```
void right_curve90(){
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
  analogWrite(pwmL, 255); //left motor
  analogWrite(pwmR, 60); //right motor
  delay(650); // Durante 0,65 segundos
}

void left_curve90(){
  digitalWrite(IN1,HIGH);
  digitalWrite(IN2,LOW);
  digitalWrite(IN3,LOW);
  digitalWrite(IN4,HIGH);
  analogWrite(pwmR, 255); //right motor
  analogWrite(pwmL, 60); //left motor
  delay(650); // Durante 0,65 segundos
}
```

Figura 5.1: Código referente às curvas de 90º