Seguidor de Linha Para LEGO ® Mindstorms Utilizando Controle PID

Robison C. Brito[1], Emanoeli Madalosso[1], Geovane A. O. Guibes[2]

[1]COENC (Coordenação de Engenharia de Computação)
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco, PR – Brasil

[2]COELT (Coordenação de Engenharia Elétrica)
Universidade Tecnológica Federal do Paraná (UTFPR) – Pato Branco, PR – Brasil

robison.brito@gmail.com, emanoeli.madalosso@gmail.com,
gguibes@gmail.com

Abstract. *This paper presents the methodology of developing a software line follower for LEGO ® Mindstorms platform using PID controller in order to obtain a faster and accurate response than methods that do not use control algorithms, resulting in undesirable performance in steep curves. Was used a brightness sensor to identify the black color of the circuit to be followed. It was programming in Java, whose use in LEGO ® Mindstrom is possible with the LeJOS firmware, which has a plugin for the Eclipse environment. The PID controller was adjusted by trial and error due to the lack of a plant. Tests were made to obtain the average time the robot took to complete a given circuit by comparing it with robots that did not use control techniques. The results were faster than the other robots.*

Resumo. *Este trabalho apresenta a metodologia de desenvolvimento de um software seguidor de linha para a plataforma LEGO ® Mindstorms utilizando controlador PID, visando obter uma resposta mais rápida e precisa comparado a métodos que não utilizam algoritmos de controle, obtendo desempenho indesejável em curvas mais íngremes. Foi utilizado um sensor de luminosidade para identificar a cor preta do circuito à ser seguido. Sua programação foi feita em linguagem Java, cuja utilização no LEGO ® Mindstrom é possível com o firmware LeJOS, que possui um plugin para o ambiente Eclipse. O controlador PID foi ajustado utilizando método de tentativa e erro, devido à falta de uma planta. Foram feitos testes obtendo o tempo médio que o robô gastava para completar um determinado circuito comparando-o com robôs que não utilizavam técnicas de controle. Os resultados obtidos foram mais rápidos que os demais robôs.*

1. Introdução

A robótica é definida como a ligação inteligente entre percepção e ação, sendo necessário certo grau de inteligência para realização de uma determinada tarefa, envolvendo uma interação física entre o sistema e o meio onde a tarefa está sendo realizada [Pio, Castro e Castro 2006].

Robôs são capazes, em algumas situações, de exercer tarefas sem a necessidade de constante supervisão, uma vez que podem ser equipados com diversos tipos de

sensores para perceber o que está acontecendo a sua volta e tomarem decisões de forma autônoma. Neste contexto, os robôs vêm sendo cada vez mais empregados em diversos setores, auxiliando no trabalho de pessoas, empresas e fábricas, realizando muitas vezes tarefas que trariam riscos a vida humana. Sendo assim, existe a preocupação de tornar estes robôs cada vez mais precisos e com uma resposta mais rápida diante das condição nas quais se encontram.

Existe uma grande quantidade de técnicas que visam atender esta preocupação, sendo uma delas o controle PID (Proporcional-Integral-Derivativo), que é o algoritmo de controle mais usado na indústria e tem sido utilizado em todo o mundo para sistemas de controle. A popularidade de controladores PID pode ser atribuída em parte ao seu desempenho robusto em uma ampla gama de condições de funcionamento e em parte à sua simplicidade funcional, que permite aos engenheiros operá-los de uma forma simples e direta. [Instruments 2011]

Procurando aplicar este conceito de controle, este trabalho apresenta o desenvolvimento de um robô seguidor de linha para a plataforma LEGO ® Mindstrom, usando controle PID pelo o método de tentativa e erro para a sintonia dos ganhos. Esta plataforma é o resultado de uma colaboração de mais de 20 anos entre o Media LAB do M.I.T (*Massachusetts Institute of Technology*) e a companhia LEGO ®. Os Mindstorms RIS (*Robotic Invention*) são baseados na tecnologia de bloco programável do MIT, onde um pequeno computador portátil é inserido dentro de um tradicional bloco LEGO ®. Com esta tecnologia agregada, o bloco é capaz de interagir com o mundo físico através de sensores e motores, permitindo que crianças construam e programem seus próprios robôs. Porém, esta tecnologia tem sido usado não só para brincadeira e aprendizado de crianças e adolescentes, mas também como uma ferramenta de prototipagem rápida por parte de empresas, como por exemplo, a *United States Postal Service*. [MIT 2013]

2. Fundamentação Teórica

Nesta seção são apresentados conceitos teóricos envolvidos no desenvolvimento do projeto, tais como características de funcionamento e programação da plataforma LEGO ® Mindstorms e conceitos sobre controladores PID.

2.1. Plataforma LEGO ® Mindstorms

A plataforma LEGO ® Mindstorms é um *kit* educacional que consiste em um conjunto tradicional de peças LEGO ®, um conjunto de sensores e atuadores e um "bloco inteligente", o NXT *Brick*, que funciona como a unidade de controle central.

O NXT *Brick* possui um microcontrolador ARM7 de 32 bits, com 256 kbytes de memória *Flash* e 64 kbytes de memória RAM (*Random Access Memory*). Este possui um display LCD (*Liquid Crystal Display*) monocromático de 100 x 60 pixels e quatro botões para navegar em sua interface com o usuário, usando menus hierárquicos, alimentação por 6 pilhas AA de 1.5 V (Volts) ou uma bateria de lítio, entre outras características.

Além disso, o NXT *Brick* possui quatro entradas para sensores e três saídas para acionar dispositivos de atuação. Por padrão, ao adquirir um *kit* LEGO ® Mindstorms, o usuário contará com um sensor ultrassônico para medir distâncias, um sensor de luz, um sensor de som, dois sensores de toque e três servos motores (com sensores de aceleração incorporados). No entanto, o usuário ainda pode obter sensores adicionais, tais como

sensores de temperatura, sensores de cor, entre outros. A Figura 1 apresenta o NXT *Brick* acoplado a seus sensores e atuadores.

Figura 1. NXT Brick.

O conjunto de peças disponível permite montar robôs de diferentes formas, tais como veículos, animais e humanoides, entre outros. Estes podem ser programados com as funcionalidades desejadas, já que o NXT *Brick* possui uma porta USB (*Universal Serial Bus*) 2.0 e também conta com comunicação Bluetooth, através das quais pode-se enviar programas específicos desenvolvidos em computador. A plataforma LEGO ® Mindstorms já conta com uma IDE (*Integrated Development Environment*) nativa para desenvolvimento de programas, o NXT-G. Esta é baseada no estilo *drag and drop* (do inglês, arrastar e soltar).

Por meio desta IDE é possível desenvolver programas de maneira simples, arrastando e conectando componentes mais básicos como loops (estruturas de repetição), *waits* (indicam que o robô deve esperar um determinado evento acontecer), *ifs* (estruturas de decisão) e até mesmo componentes mais avançados, tais como o Bluetooth. Existe um conjunto de botões que facilitam a compilação e execução do programa no NTX *Brick*.

2.2. LEJOS

Embora a IDE NXT-G proporcione uma forma rápida e de desenvolver aplicativos para o LEGO ® Mindstorms, sua principal limitação é que não permite o desenvolvimento de programas mais complexos. Para isso existe o LeJOS, um firmware substituto para o firmware original do NXT *Brick.* O LeJOS (pronunciado como a palavra espanhola "lejos" para "longe") é uma pequena máquina virtual Java que em 2006 foi portada para o NXT *Brick*. Este inclui todas as classes da API (*Application Programming Interface*) NXJ, bem como as ferramentas utilizadas para enviar o código para o NXT *Brick* [LeJOS 2013]. Este firmware fornece recursos como:

- Programação orientada a objetos (linguagem Java);
- Suporte a Threads;
- Vetores, incluindo multidimensionais;
- Exceções;
- A maioria das classes java.lang, java.util e java.io;
- Plugins para IDEs gratuitas como o Netbeans e o Eclipse;
- Vasta documentação.

2.3. Controle PID

O controle PID leva este nome por ser um sistema de controle Proporcional-Integrador-Derivativo, mais da metade dos controladores industriais utilizados atualmente empregam esquemas de controle PID ou PID modificado [Ogata 2003], um exemplo de PID modificado é o controle FPID, que é uma forma de controle PID que sintoniza os seus ganhos de uma forma inteligente por meio da lógica *Fuzzy*.

Este algoritmo de controle pode ser empregado na maior parte dos sistemas de controle, principalmente naqueles em que não conhecemos o modelo matemático do sistema à ser controlado, e métodos analíticos não podem ser empregados [Spandri 2003]. Alguns sistemas de controle não tem boas respostas com controles PID, neste caso deve-se procurar uma solução mais viável para o sistema.

O PID trabalha com malha-fechada, ou seja, lê valores de um sensor, calcula a resposta para cada atuador do sistema e então somar as três componentes calculadas, o resultado desta soma é o valor da saída do sistema. A Figura 2 apresenta o diagrama de blocos do sistema malha-fechada.

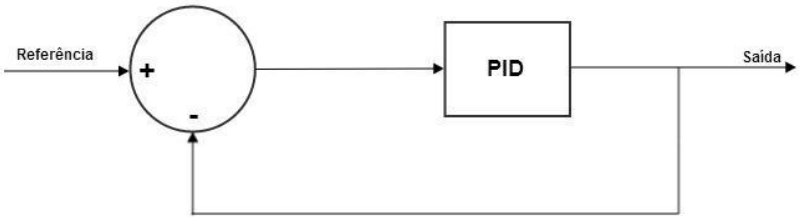

Figura 2. Exemplo de sistema malha-fechada.

A parte proporcional do algoritmo é calculada por uma multiplicação entre o erro e o ganho Kp, o erro é a diferença entre o valor lido pelo sensor e pela referência, esta referência é o ponto onde o sistema deve estabilizar conhecido como set-point. A parte integradora consiste no cálculo de uma integral, ou seja, ela realiza uma soma do valor anterior da integral com o erro, tudo isso multiplicado por um ganho que é chamado de Ki. A parte derivativa consiste por uma derivada, que é a subtração entre o valor do erro anterior e o erro atual, este valor também é multiplicado por um ganho, o ganho derivativo é chamado de Kd, melhor visualizado na equação 1. O algoritmo pode ser melhor visualizado na Figura 3.

$$u(t) = Kp * e(t) + Ki \int_0^t e(t)dt + Kd \frac{\partial e(t)}{\partial t} \quad (1)$$

onde, u(t) é a saída controlada, Kp é o ganho proporcional, e(t) é o erro, Ki é o ganho integral e Kd é o ganho derivativo.

2.4. Métodos para Sintonia de Controladores PID

Os métodos de ajuste dos ganhos para que o controlador responda da forma desejada são chamados de sintonia dos ganhos, os ganhos são Kp, Ki e Kd, existem vários métodos para realizar a sintonia desejada, os principais são Cohen-Coon e Ziegler-Nichols.

2.4.1. Método de Ziegler-Nichols

O método de Ziegler-Nichols é utilizado principalmente quando não se conhece o comportamento do sistema, este método consiste em conseguir a resposta do sistema quando aplicada uma rampa de degrau unitário de acordo com a Figura 4. Onde L é o atraso do sistema, T é a constante de tempo.

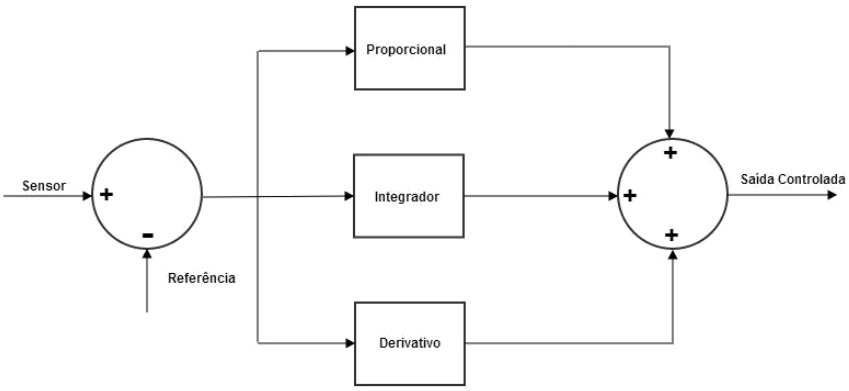

Figura 3. Diagrama de blocos de um sistema PID.

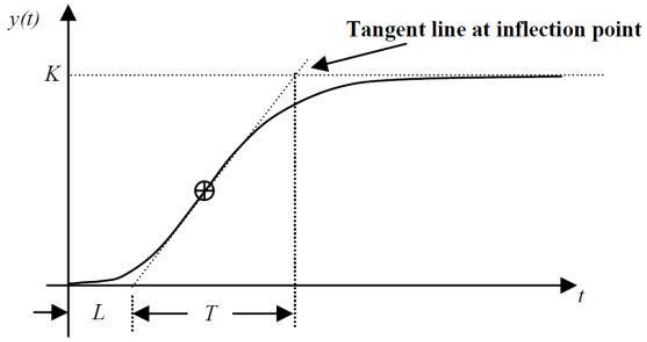

Figura 4. Exemplo de resposta de um sistema ao degrau unitário.

De acordo com Ziegler-Nichols o cálculo ideal dos ganhos do sistema quando se obtém a resposta em malha-aberta em degrau unitário é representado na tabela 1.

Uma das desvantagens do método de Ziegler-Nichols é que ao fechar a malha do sistema ele tente a ter uma resposta mais oscilatória. Para melhorar esta resposta em malha-fechada Cohen e Coon otimizaram o método de Ziegler-Nichols.

2.4.2. Método de Cohen-Coon

Tentando diminuir as oscilações do método de Ziegler-Nichols, utiliza-se Cohen-Coon que praticamente altera apenas o cálculo dos ganhos, ele utiliza os valores obtidos na resposta do sistema ao degrau unitário, assim como o método de Ziegler-Nichols, utiliza também o atraso L e a constante de tempo T. O cálculo dos ganhos é mostrado nas equações 2, 3 e 4 [do Carmo e F. J. Gomes 2005].

Tabela 1. Cálculo dos ganhos pelo método de Ziegler-Nichols.

Controlador	Kp	Ki	Kd
P	T/L	∞	0
PI	1.9 (T/L)	L/0.3	0
PID	1.2 (T/L)	2L	0.5L

$$Kp = \frac{\frac{L}{4T} + \frac{4}{3}}{K\frac{L}{T}} \quad (1) \qquad Ki = L\,\frac{\frac{3L}{4T} + 4}{\frac{L}{T} + \frac{11}{2}} \quad (2) \qquad Kd = L\,\frac{2}{\frac{L}{T} + \frac{11}{2}} \quad (3)$$

2.4.3. Método da Tentativa e Erro

Todos estes métodos são utilizados quando se tem a resposta em degrau do sistema em malha-aberta, porém quando não possuí estes dados a forma mais simplificada para se encontrar os ganhos são “chutados” valores, este método funciona da seguinte maneira:

- Zeram-se os ganhos Kd e Ki, e vai variando o Kp até encontrar a velocidade da resposta do sistema da maneira em que ele responda da forma desejada;
- Após encontrar o valor do Kp usa-se o Ki para diminuir as oscilações, ele é aumentado até conseguir encontrar um valor onde as oscilações fiquem da forma desejada, geralmente com a menor oscilação possível, porém este valor não deve ser muito alto pois ele pode causar uma instabilidade no sistema;
- Quando obter o valor do Kp e do Ki, ajusta-se o Kd, este ganho é ajustado até que o loop do algoritmo responda de uma forma rápida e aceitável, porém ele também pode causar instabilidade no sistema.

A desvantagem deste método é que não é possível obter a melhor resposta do sistema.

3. Metodologia

Esta seção apresenta informações sobre os materiais utilizados, desenvolvimento do software seguidor de linha, técnica para sintonia dos ganhos do controlador PID e procedimento de testes.

3.1. Materiais

O *kit* LEGO ® Mindstorms foi montado com o formato de um veículo com um sensor de luminosidade rente ao chão, conforme a Figura 5. Este foi chamado de "robô 1”.

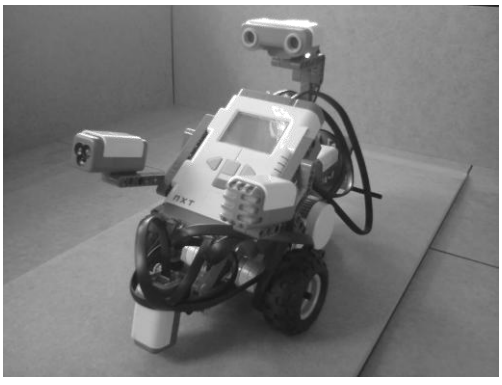

Figura 5. Robô montado em forma de veículo.

Para a programação do aplicativo seguidor de linha foi utilizado a linguagem Java (Versão 7u45). Para isto utilizou-se a IDE gratuita Eclipse (Versão Juno), com o *plugin* para LeJOS.

3.2. Desenvolvimento do Algoritmo

No intuito de obter uma melhor eficiência para o seguidor de linha, o programa desenvolvido foi baseado em controle PID. Inicialmente foram declaradas variáveis para os ganhos proporcional, integral e derivativo, respectivamente Kp, Ki e Kd. Também foi declarada uma variável contendo a potência máxima a ser usada pelo robô, e outra para a referência do sistema. Esta referência é dada pelo código 1.

Código 1. Cálculo da referência de cor.

```
1 ref = (preto + branco)/2;
```

Sendo que os valores da cor preta e branca foram obtidos fazendo capturas com o sensor de luminosidade sobre a cor branca e sobre a cor preta. Este processo era feito sempre antes que o robô iniciasse seu trajeto, garantindo assim que a referência esteja correta mesmo caso as condições de luminosidade mudassem.

Incialmente foram feitas as configurações do controlador PID. As bibliotecas do LeJOS já contam com uma classe chamada "PIDController". Ao ser iniciada uma instância dessa classe (código 2), passa-se por parâmetro a referência e um valor de *delay* em milissegundos. Este *delay* deve ser suficiente para que os cálculos do PID sejam feitos.

Código 2. Instância da classe PIDController em LeJOS.

```
1 PIDController pid = new PIDController(referencia, delay);
```

Em seguida passa-se por parâmetro os valores de KP, KI e KD, como no código 3.

Código 3. Setando valores para os ganhos.

```
1 pid.setPIDParam(PIDController.PID_KP, KP);
2 pid.setPIDParam(PIDController.PID_KI, KI);
3 pid.setPIDParam(PIDController.PID_KD, KD);
```

Estes ganhos foram ajustados pelo método de tentativa e erro, citado na subseção 2.4.3. Também devem ser passados por parâmetro um limite máximo e um limite mínimo de saída, que limita a saída do controlador PID para que não haja um overflow. Pode ser visto no código 3.

Código 4. Setando limites para o controlador PID.

```
1 pid.setPIDParam(PIDController.PID_LIMITHIGH, limiteMax);
2 pid.setPIDParam(PIDController.PID_LIMITLOW, limiteMin);
```

Finalmente, após feitas as configurações do PID, dentro de uma função de loop foi feita a leitura dos valores do sensor de luminosidade. Também nesta função era definida a potência de cada motor, a partir uma variável chamada "potencia" que recebia o valor resultante da operação PID, que é demonstrada no código 5.

Código 5. Cálculo atuador do controlador PID.

```
1 potencia = pid.doPID(luminosidade);
2 Motor.A.setPower(potenciaMaxima + potencia);
3 Motor.B.setPower(potenciaMaxima - potencia);
```

O programa seguidor de linha desenvolvido tinha como objetivo ser capaz de fazer o robô completar um circuito no formato do autódromo de Interlagos (Figura 7), que tem como característica possuir curvas íngremes.

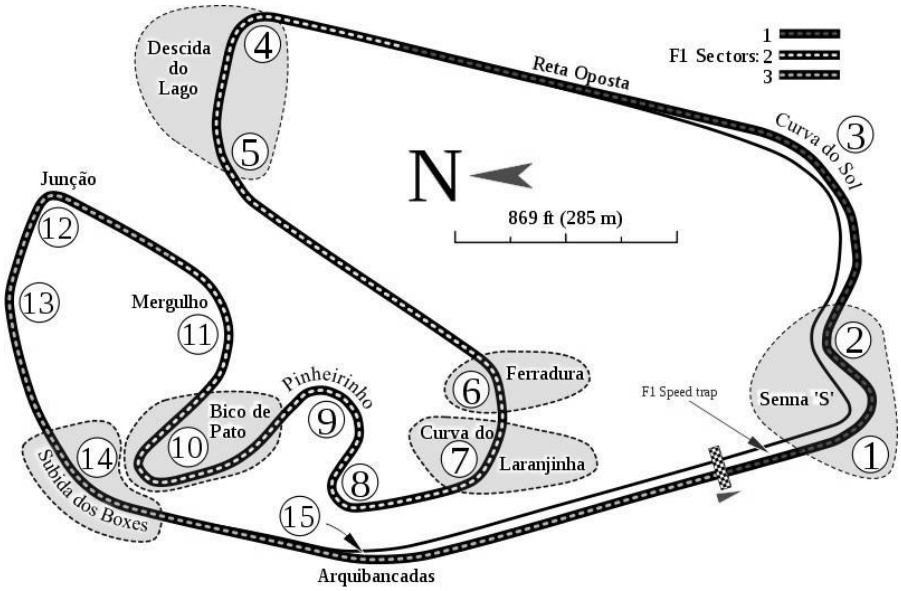

Figura 6. Circuito de Interlagos.

Sendo assim foi impresso uma miniatura deste circuito em plástico fosco com 2 metros de comprimento e 1,20 metro de largura. Um robô chamado "juiz", que contava com um sensor ultrassônico, foi posicionado na metade da "reta oposta" do circuito (conforme a Figura 6). Quando o sensor do juiz detectasse um objeto a menos de 25 centímetros, este interpretava que o robô a ser testado havia iniciado o circuito (em sentido anti-horário) e começava a contar o tempo. Quando o robô passava pela segunda vez em frente ao juiz, este interpretava que o robô havia completado o circuito, encerrando a contagem de tempo e apresentando em seu *display* o tempo gasto pelo robô para completar o circuito.

4. Resultados

Após programado com a versão final do aplicativo seguidor de linha com controle PID, o robô 1 foi testado 5 vezes no circuito, com as mesmas condições de iluminação em todas as vezes. Os testes aconteceram na final do concurso de desenvolvimento de software seguidor de linha para LEGO ® Mindstorms que ocorreu na primeira Inventum - Feira de Tecnologia, Informação e Inovação, no dia 09 de Novembro de 2013, em Pato Branco, sendo o robô 1 um dos finalistas. Os resultados dos testes são apresentados na Tabela 2.

Foram também realizados testes com outros 4 robôs. Os robôs 2 e 5, foram programados utilizando a IDE nativa NXT-G. A estratégia utilizada por estes robôs pode ser verificada no fluxograma da Figura 7. Nesta pode-se perceber que se a cor detectada fosse branca, o robô fazia um pequeno movimento para a esquerda até encontrar novamente a faixa preta. Encontrando a faixa preta, este fazia um pequeno movimento para a direita para voltar a seguir em frente. A diferença entre os robôs 2 e 5 consistia basicamente no ajuste de potência dos seus motores. Esta mesma estratégia também foi adotada pelo robô 3, porém este foi programado utilizando a linguagem Java. A vantagem de utilizar esta estratégia é que os robôs 2, 3, e 5 conseguiram realizar

as curvas com precisão, porém para isso tiveram que manter uma velocidade baixa, para não se perderem nas curvas.

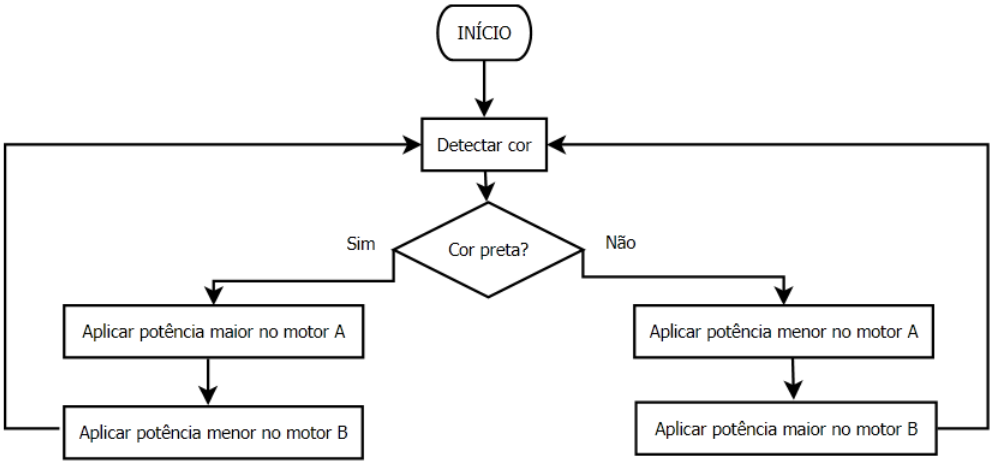

Figura 7 – Fluxograma da estratégia dos robôs 2, 3 e 5.

O robô 4 também utilizou a linguagem Java, porém adotou uma estratégia diferente, conforme o fluxograma da Figura 8. O robô seguia em linha reta enquanto estivesse sobre a cor preta (ganhando velocidade nestes trechos). Ao encontrar a cor branca, o robô verificava que havia chegado em uma curva, mas não sabia se esta era para a esquerda ou para a direita, necessitando assim fazer uma verificação. Para isso o robô fazia um pequeno giro primeiramente para a esquerda e, em caso de detecção da cor preta, voltava a seguir reto. Caso a cor preta não fosse encontrada o robô girava então para a direita, verificando que esta deveria ser a direção da curva. Uma das vantagens deste robô era que podia ganhar mais velocidade nas retas, porém tinha como desvantagem a complexidade e o volume de código, que envolvia o uso de várias *flags* para indicar a situação do robô, além de perder um tempo considerável identificando as curvas.

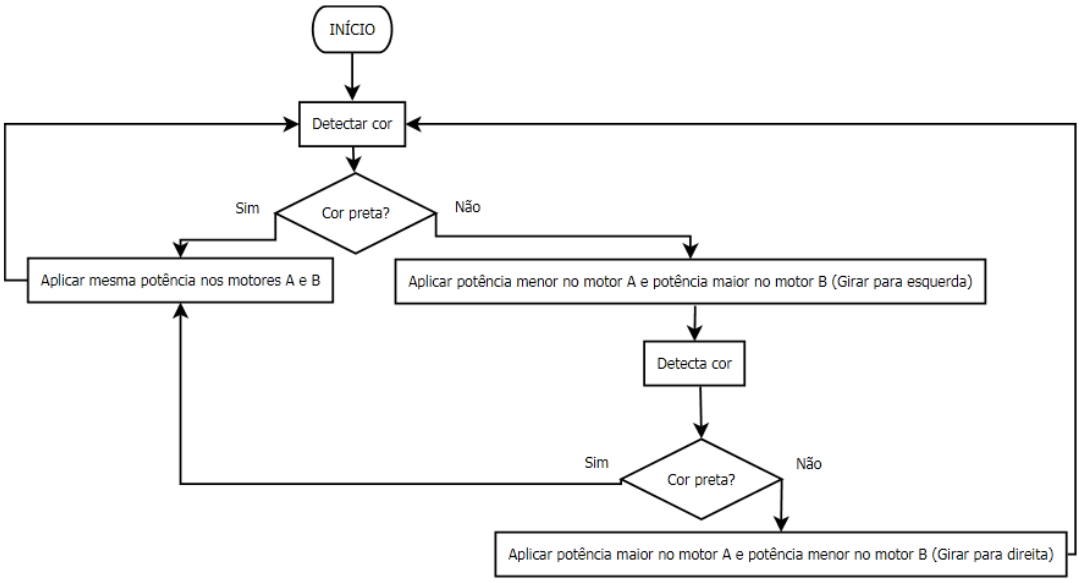

Figura 8 – Fluxograma da estratégia do robô 4.

Realizando uma comparação entre os robôs pode-se notar pela Tabela 2 a diferença de tempos obtida utilizando seguidores de linha com métodos diferentes.

Tabela 2. Comparação de tempos entre os robôs.

	Robô 1	Robô 2	Robô 3	Robô 4	Robô 5
Tempo médio (5 voltas)	37.5496 segundos	88.6200 segundos	92.1000 segundos	60.4890 segundos	62.3250 segundos

5. Conclusão

Para o desenvolvimento e testes deste projeto, foi utilizada uma miniatura do circuito de Interlagos. O software foi desenvolvido em Java, na IDE Eclipse, que contém um *plugin* para o LeJOS. No intuito de obter uma resposta melhor para o seguidor de linha o programa utilizou a técnica de controle PID, uma vez que outras tentativas de desenvolver um seguidor de linha sem métodos de controle apresentaram um desempenho indesejável como baixa velocidade, falta de precisão nas curvas e desvio do trajeto do circuito.

Uma das maiores dificuldades no desenvolvimento deste projeto foi encontrar os valores dos ganhos para o controlador PID, uma vez que pequenas alterações nesses ganhos geravam grandes mudanças na resposta do robô. Outro fator que necessitou de mais atenção foi o balanceamento da potência aplicada nos motores, pois ao deixar o robô rápido demais este podia realizar as curvas com imperfeição fazendo-o desviar-se do circuito.

Analisando os resultados obtidos, o robô 1 mostrou-se mais eficiente que os robôs que não usavam técnicas de controle, realizando melhor as curvas, mantendo uma velocidade desejável e completando o circuito em tempo menor, sendo 37.93% mais rápido que o robô 4, que obteve o segundo melhor tempo. Outra vantagem foi simplicidade do código, já que este foi feito em poucas linhas e utilizando poucas variáveis. Nota-se também que embora técnicas de controle aparentemente pareçam complicadas, o controle PID foi aplicado na prática sem grandes dificuldades, mostrando que com um pouco mais de estudos este tipo de técnica pode ser usada desde aplicações simples até mais complexas para se obter um sistema mais eficiente.

Referências

do Carmo e F. J. Gomes, M. J. (2005). Sintonia de controladores e análise funcional de malhas idustriais em ambiente multifuncional integrado. COBENGE.

Instruments, N. (2011). Explicando a Teoria PID. National Instruments. Disponível em: <http://www.ni.com/white-paper/3782/pt/> Acesso em: 25 Out. 2013.

Lejos, Java for LEGO Mindstorms (2013). NXJ technology. Disponível em: <http://www.lejos.org/nxj.php> Acesso em: 23 Out. 2013.

MIT Media Lab (2013). LEGO's Mindstorms. Disponível em: <http://media.mit.edu/sponsorship/getting-value/collaborations/mindstorms> Acesso em: 20 Out. 2013.

Pio, J. L., Castro, T., e Castro, A. (2006). A robótica móvel como instrumento de apoio a aprendizagem de computação. XVII Simpósio Brasileiro de Informática na Educação.

Ogata, K. (2003). In Hall, editor, Engenharia de Controle Moderno, pages 554–565.

Spandri, R. (2003). Sintonia de controladores pid. Boletim Tecnológico Petrobras.