
# Desenvolvimento de um Robô Autoguiado Seguidor de Linha Utilizando Controladores Proporcional, Integral e Derivativo

César Augusto Victor cesar-tri@hotmail.com IFCE
Renan Corrêa Basoni renan.basoni@ifc.edu.br IFC
Joel Carneiro Rosalino Sousa joelcrs96@gmail.com IFCE
Rigoberto Luis Silva Sousa rigoberto.sousa@ifce.edu.br IFCE
Rodolfo de Souza Zanuto rodolfo.zanuto@ifce.edu.br IFCE

## Resumo

Sabe-se que a escassez de informações se torna um obstáculo para inserção dos estudantes no mundo da robótica, resultando num baixo número de pesquisas sobre o assunto. Considerando-se a relevância que esse tema pode agregar na vida dos alunos, faz-se necessário o desenvolvimento de mecanismos de inclusão e disseminação da robótica educacional, sendo assim, o objetivo desta pesquisa é o de tornar mais acessível os passos necessários para a construção do robô seguidor de linha para ser utilizado na robótica educacional e/ou como auxilio na transmissão e assimilação dos conteúdos da disciplina de Controle Clássico. O intuito deste projeto é disponibilizar todos os meios necessários para que professores e alunos consigam, simultaneamente, replicarem o projeto proposto e o utilizarem como um *kit* didático durante as aulas da disciplina de Controle Clássico. Ao final do artigo é possível visualizar que os resultados obtidos foram satisfatórios, pois o robô seguidor de linha projetado cumpriu todos os requisitos de funcionamento, percorrendo com eficácia o trajeto de uma pista de competição de seguidores de linha. O trajeto utilizado continha tracejados, interseções entre retas e curvas com diferentes ângulos, inclusive em 90 graus.

## Palavras chaves

Robótica Educacional, Seguidor de linha e Controle PID.

## 1. Introdução

Pretende-se através da proposta deste projeto disponibilizar informações a respeito da montagem e funcionamento de um sistema embarcado aplicado num robô seguidor de linha, para utilização na robótica educacional e/ou na disciplina de Controle Clássico.

O intuito do protótipo é alcançar o maior número possível de estudantes, fornecendo uma ferramenta lúdica e didática para o aluno se inserir no mundo da robótica e correlacionar

com mais clareza os conceitos teóricos aprendidos na sala de aula referente aos controladores proporcional, integral e derivativo (PID) pertencentes a disciplina de Controle Clássico.

## 2. Referencial Teórico

Dado o amplo uso de controladores PID (controle de potência em motores de indução, controle de nível, vazão, entre outros), o uso de microcontroladores para o desenvolvimento desse tipo de aplicação ganhou força graças à incorporação de linguagens de alto nível que facilitam muito esse tipo de implementação, além dos baixos custos de aquisição desses dispositivos, distribuição de *software* de desenvolvimento gratuito e informações abrangentes na Internet (RUGE, 2015).

A robótica educacional desperta nos alunos a curiosidade e o sentimento de alcançar o entendimento natural sobre o funcionamento dos robôs, ao ponto de ser um fator motivacional para o cumprimento de estudos multidisciplinares. Dessa maneira a robótica educacional se torna uma ferramenta prazerosa para estudar disciplinas como cálculo, física, lógica de programação, controle clássico e etc.

Segundo (Brito, R. S; Moita, F. M. G. S. C; Lopes, M. C):

> A competência surge com a aquisição dos conhecimentos, e considerando que esta vem do conhecimento, o trabalho desenvolvido através da robótica, possibilita ao aluno planejar, projetar, criar, desenvolver e avaliar a aquisição e a apropriação dos conhecimentos. A robótica constitui-se uma área multidisciplinar, que estimula os educandos a buscar soluções associando conceitos e aplicações em outras disciplinas envolvidas. Nesse sentido, ela permite verificar o processo fundamental de aprendizagem do aluno, pela experimentação e testagem de hipótese, bem como favorece a mediação pedagógica do professor, produzindo novos e diferentes conhecimentos que colaboram na formação do cidadão alfabetizado tecnologicamente, capaz de atuar ativamente na sociedade a partir de um novo tipo de ambiente educacional que produz novos e diferentes conhecimentos.

Os sistemas embarcados estão presentes em grande parte dos dispositivos modernos como os computadores, tablets e *smartphones*. Esses dispositivos fazem uso de tecnologias embarcadas, desenvolvidos através da linguagem de programação para se comunicarem com outros equipamentos eletrônicos controlando suas funções e trocando informações na forma de dados (NOERGAARD, 2005, apud, BASONI, 2015).

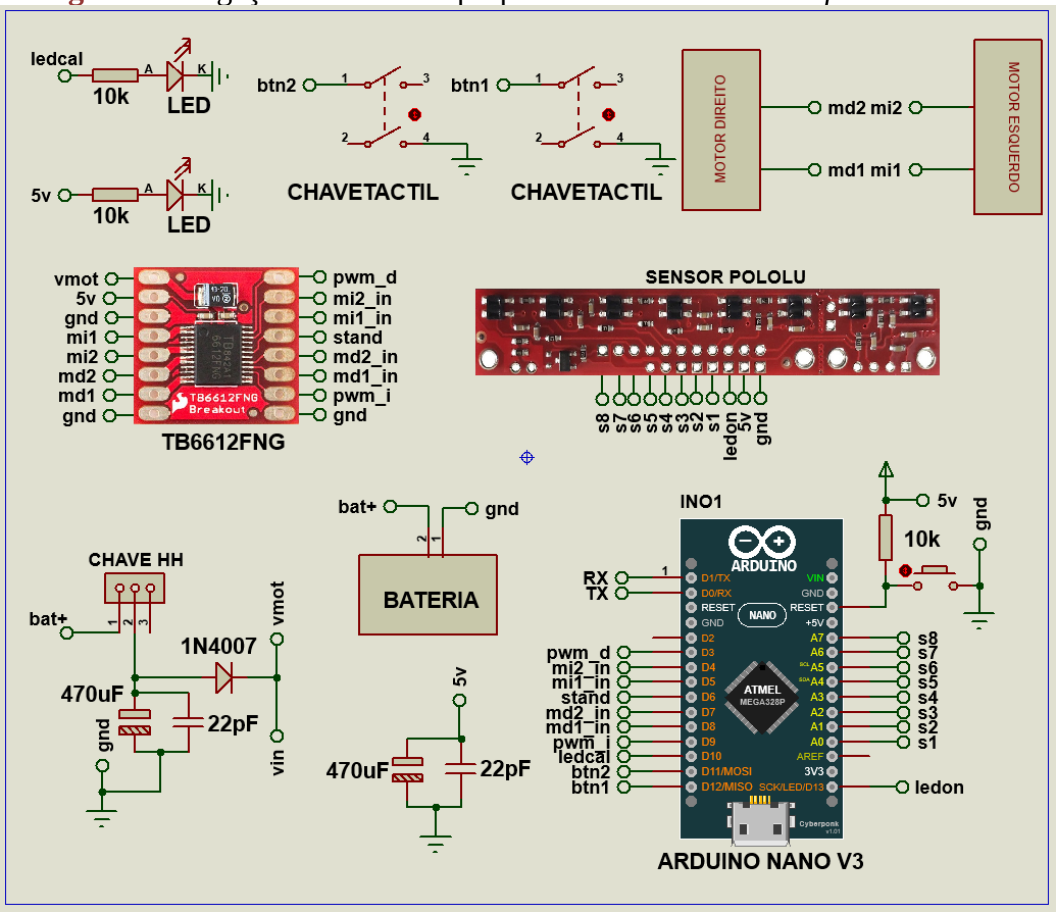
## 3. Materiais e Métodos

O estudo e desenvolvimento do projeto foi realizado nos Laboratórios de Eletrônica 2, Usinagem e do GEPRO do Instituto Federal do Ceará Campus Sobral.

### 3.1. Sistema Embarcado Desenvolvido

O sistema embarcado deste projeto utiliza o microcontrolador AVR ATmega328p, que possui um conversor A/D de oito canais com resolução de 10 bits e quatorze portas digitais, na qual seis são saídas PWM (*Pulse Width Modulation*). A Figura 1 mostra o esquemático proposto desenvolvido no *software* Proteus.

**Figura 1** – Ligações do circuito proposto desenvolvido no *software* Proteus.




**Fonte:** Produção do próprio autor (2020).

### 3.1.1. Sensores

Os principais componentes do robô seguidor de linha são os sensores ópticos. Esses sensores emitem luz com um diodo (geralmente infravermelho) e leem a luz refletida com um sensor de luz (geralmente um fotodiodo).

O sensor utilizado no protótipo foi o modulo QTR-8A (Figura 2) que contém 8 sensores QRE, através das informações dos sensores o microcontrolador consegue interpretar a posição do seguidor de linha.

**Figura 2** – Modulo de sensores QTR-8A.

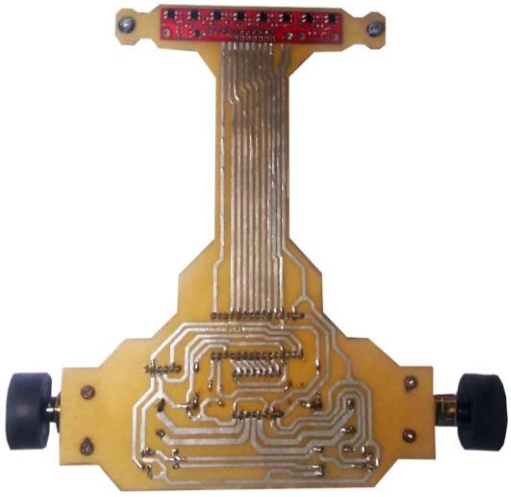
**Fonte:** Pololu (2019).

### 3.1.2. Chassi

Na construção do chassi do robô projetado foi utilizado uma placa de circuito impresso (PCI) de fenolite, com espessura de 0,8 mm. Durante a realização da pesquisa para o desenvolvimento do projeto, ocorreu a preocupação da possibilidade de o chassi quebrar em caso de colisão com paredes de alvenaria, com outro seguidor de linha e etc, porém, o chassi apresentou boa resistência mecânica. Ao contrário de chassis produzidos com outros materiais, como por exemplo plástico e madeira, o tipo escolhido possibilitou a não utilização de vários

*jumpers* para a realização das ligações elétricas dos componentes eletrônicos, tornando o aspecto do projeto muito mais agradável como mostra a Figura 3.

**Figura 3** – Parte posterior robô.



**Fonte:** Produção do próprio autor (2020).

### 3.1.3. Motores

O motor é o elemento principal e crítico que determinará as características do robô, por isso é preciso fazer bem a escolha de qual utilizará, uma vez que o desempenho dependerá dele, deve-se considerar que tipo de redução, consumo e torque ele possui. O motor foi o *Micro Metal GearMotor MP* 6V, sua velocidade é de 3000 RPM com caixa de redução de 10:1 e sua corrente é de 0,67 A. A Figura 4 exibe o motor utilizado no protótipo.

**Figura 4** – Micro Metal GearMotor.

**Fonte:** Pololu (2019).

### 3.1.4 Drive (Ponte H)

Como o microcontrolador Atmega328p não é capaz de fornecer corrente para os motores utilizados chegarem na velocidade nominal, tornou-se necessário a utilização de um *driver* para energizar os motores. O *driver* utilizado neste projeto foi o TB6612FNG e pode ser visualizado na Figura 5. Este *driver* possui dois canais de saída e por este motivo podem ser utilizados dois motores. O *driver* TB6612FNG é capaz de fornecer uma corrente contínua de três amperes de pico por canal e suporta uma tensão de até 13 volts.

**Figura 5** – *Driver* TB6612FNG.

**Fonte:** Pololu (2019).

### 3.1.5 Bateria

Atualmente a bateria de Lipo é a melhor opção para se usar em seguidores de linha, pois apresentam maior durabilidade de carga. A bateria utilizada no robô foi a Zippy com 500 mAh e 7,4 V. Recomenda-se utilizar baterias com 7,4V porque os motores para seguidores de linha operam aproximadamente nessa tensão. A bateria utilizada no projeto pode ser visualizada na Figura 6.

**Figura 6** – Bateria Zippy 500mAh, 35C.


**Fonte:** Adaptado de Mercado Livre (2020).

### 3.1.6. Plataforma de Prototipagem Eletrônica

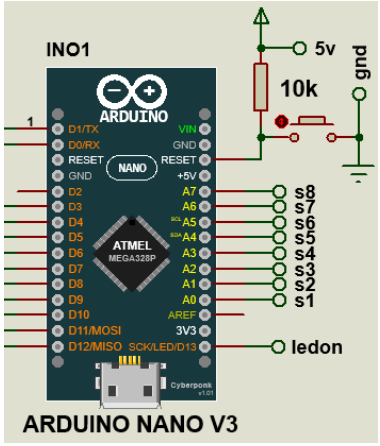
Em teoria, qualquer Arduino poderia ter sido utilizado. Porém, o mais aconselhável é que em um robô seguidor de linha (que geralmente é muito leve) se use um Arduino Nano ou Pro Mini. Optou-se por utilizar o Arduino Nano (Figura 7), porque o mesmo permite carregar o programa diretamente do cabo usb, recurso indisponível no Arduino Pro Mini.

**Figura 7** – Arduino Nano.

**Fonte:** Loja Arduino Belém (2019).

Os pinos analógicos (A0, A1, A2, A3, A4, A5, A6 e A7) do arduino foram utilizados para as leituras dos sensores. A Figura 8 exemplifica a ligação entre o sensor e o Arduino Nano.

**Figura 8** – Conexão entre sensor e Arduino.




**Fonte:** Produção do próprio autor (2020).

### 3.1.7. Valores dos Componentes

A Tabela 1 mostra a lista dos componentes que foram utilizados para a construção do robô seguidor de linha. A escolha deles foi selecionada a partir de estudos, pesquisas e protipagem de seguidores de linha para ter um único modelo com componetes mais adequados.

**Tabela 1** – Lista de componentes.

| Componente | Quantidade | Preço R$ |
|---|---|---|
| Arduino Nano | 1 | 30,00 |
| *Driver* TB6612FNG | 1 | 33,79 |
| *Micro Metal GearMotor* | 2 | 42,42 |
| Modulo QTR-8A | 1 | 124,47 |
| Chave Táctil | 2 | 0,70 |
| PCI de Fenolite 15x20cm | 1 | 12,30 |
| Bateria 7.4V 500Mah | 1 | 105,00 |
| Capacitor cerâmico 22pF | 2 | 0,80 |
| Capacitor eletrolítico 100uF | 2 | 1,50 |
| Diodo 1N4007 | 1 | 0,60 |
| Resistor 220Ω | 2 | 1,00 |
| LED difuso | 2 | 0,40 |
| Chave HH | 1 | 9,00 |
| Barra de Pino Macho | 2 | 1,00 |
| Suporte para Micromotor | 2 | 5,00 |
| *Silicone Wheel* | 2 | 66,24 |
| Total | 25 | 434,22 |

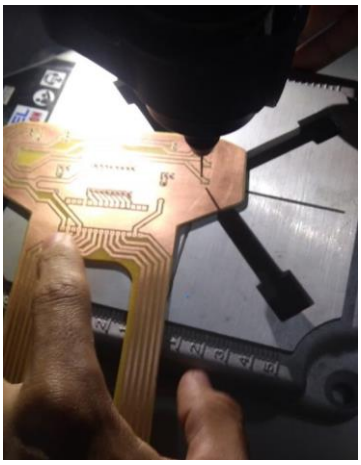
**Fonte:** Produção do próprio autor (2020).

## 4. Construção do Robô

O projeto foi construído em uma PCI pelo método da transferência térmica. A confecção de uma PCI pelo método da transferência térmica pode ser encontrada através de vários tutoriais na internet, um deles pode ser encontrado em (SOARES, 2020).

Para fazer os furos na PCI projetada foi utilizada uma microretifica com uma broca de 0,5 mm. A Figura 9 exibe a PCI em conjunto com a microretifica.

**Figura 9** – Desenvolvimento da PCI.



**Fonte:** Produção do próprio autor (2020).

Ao realizar todos os acabamentos e ajustes necessários, foi utilizado um multímetro para verificar se todas as trilhas estavam perfeitas, após essa verificação é o momento de soldar os componentes. Através da Figura 10 é possível visualizar um tutorial de soldagem.

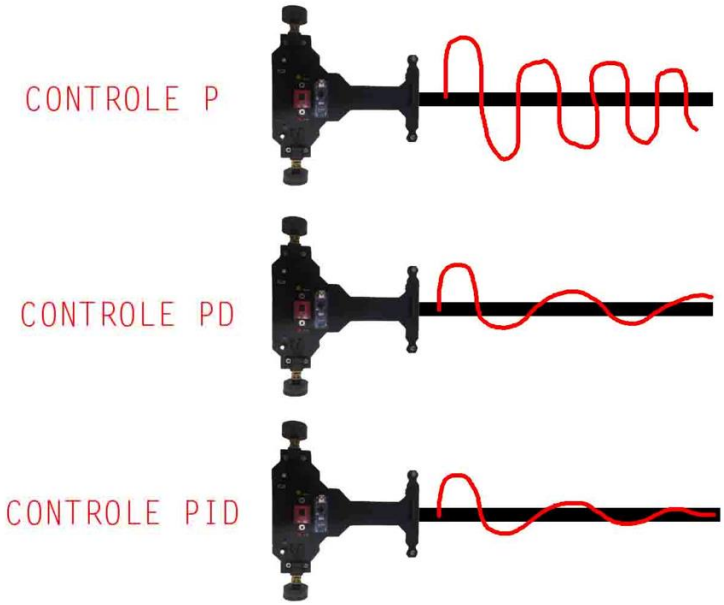
**Figura 10** – Tutorial de soldagem de componentes eletrônicos.


**Fonte:** Adaptado da Loja Tarzan Soluções em Eletrônica em (2020).

## 5. Ajustes no Algoritmo PID

O PID atua sobre as medidas dos erros entre o *setpoint* (robô centralizado na linha) e a posição do robô (leitura dos sensores). Kp, Ki e Kd, são respectivamente, as constantes proporcional, integral e derivativa. Essas constantes são utilizadas pelo algoritmo com o objetivo de zerar o erro de posição entre o robô e a linha, o erro sempre será zero quando o robô ficar centralizado sobre a linha. A Figura 11 exemplifica possíveis características das ações de controle P, PD e PID.

**Figura 11** – Atuação do Controle P, PD e PID.




**Fonte:** Produção do próprio autor (2020).

Antes de demonstrar como sintonizar as constantes Kp, Ki e Kd. No Quadro 1 são apresentadas algumas definições importantes para auxiliar na compreensão do funcionamento do PID.

**Quadro 1** – Definições utilizadas na sintonia das constantes Kp, Ki e Kd.

| Siglas/Efeito | Explicação |
|---|---|
| P (Correção Proporcional ao Erro) | A correção a ser aplicada ao processo deve crescer na proporção que cresce o erro entre o valor real e o desejado. |
| I (Correção Proporcional ao Produto Erro x Tempo) | Erros pequenos, mas que existem e requerem correção para obtenção do erro zero. |
| D (Correção Proporcional à Taxa de Variação do Erro) | Se o erro está variando muito rápido, esta taxa de variação deve ser reduzida para evitar oscilações. |
| *Overshoot* | É quando a função do PID excede o *setpoint* desejado, ou seja, o robô provavelmente está com uma instabilidade em seu controle. |
| MV | É a variável de manipulação, na qual o controlador atua, no algoritmo do robô desenvolvido, MV é a variável "*Velocidade*". |
| *Setpoint* | É o valor que o sistema de controle tentará alcançar. No caso do robô desenvolvido, o *Setpoint* é 3500, exatamente o mesmo valor que os sensores retornam quando o robô permanece centralizado sobre a linha, ou seja, erro zero. |

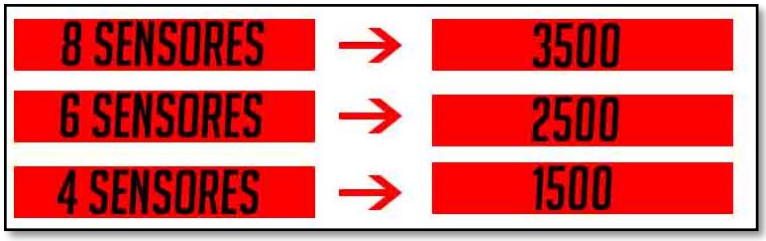
**Fonte:** Adaptado de Novus (2020).

A elaboração do algoritmo foi realizada na *Integrated Development Environment* (IDE) do Arduino e todo o código fonte está disponível no tópico "*Anexos*" deste artigo. Para facilitar o entendimento, recomenda-se que os próximos parágrafos deste tópico "*5 – Ajustes no Algoritmo PID*" sejam lidos juntamente com o algoritmo disponibilizado nos "*Anexos*".

Na primeira linha do código fonte "#include <QTRSensors.h>", pode-se visualizar a utilização da biblioteca *QTRSensors*. A referida biblioteca foi utilizada para a leitura dos sensores. A versão utilizada foi a 3.0 e pode ser encontrada em (POLOLU, 2019).

Para o controle PID atuar bem é necessário realizar alguns ajustes no algoritmo. O primeiro ajuste a ser realizado é no *define* NUM_SENSORS, este *define* é utilizado para indicar o número de sensores que serão utilizados. É através do número de sensores que é definido o valor do ponto de ajuste, as Figuras 12 e 13 exibem a relação entre o número de sensores e o ponto de ajuste.

**Figura 12** – Valores para o ponto de ajuste.




**Fonte:** Produção do próprio autor (2020).

**Figura 13** – Ajuste do erro.

```
qtra.read(sensorValues);
position = qtra.readLine(sensorValues,QTR_EMITTERS_ON,0);
                Posição da leitura do sensor QTR-8A
ERRO = ((position) - (3500));
                              Valor do ponto de Ajuste
```

**Fonte:** Produção do próprio autor (2020).

Analisando a Figura 12, pode-se perceber que considerando a utilização de 8 sensores, o "*ERRO*" sempre será zero quando a variável "*position*" for igual a 3500. Utilizando do mesmo raciocínio, haverá duas condições nas quais o "*ERRO*" será máximo, quando "*position*" for igual a 7000 e 0.

Quando "*position*" for igual a 7000, o "*ERRO*" será 3500, significando que o robô está à esquerda da linha, nesse caso, com o propósito de zerar o erro, é certo afirmar que o motor direito precisará se desacelerar, e o esquerdo acelerar, o controle PID executa automaticamente essa função.

O segundo ajuste é na determinação dos *defines* das constantes Kp, Ki e Kd. De acordo com a velocidade do motor, quanto mais rápido o robô, mais difícil será encontrar uma relação para Kp, Ki e Kd que possibilitem que o robô permaneça com eficiência centralizado sobre a linha. O cálculo do PID pode ser visualizado na Figura 14 através da equação que calcula a variável "*Velocidade"*.

**Figura 14** – Equação do PID.

```
Proporcional = ERRO*KP;
Integral =(ERRO + ERRO_ANTERIOR)*KI;
Derivativo = (ERRO - ERRO_ANTERIOR)*KD;
velocidade= Proporcional + Integral + Derivativo;
if(velocidade > VelMax)
{
 velocidade = VelMax;
}
else
   if(velocidade < -VelMax)
   {
    velocidade = -VelMax;
   }
```

**Fonte:** Produção do próprio autor (2020).

Ao começar a ajustar as constantes do PID, é importante que se inicie com um PWM baixo, em torno de 50. Para facilitar no momento do ajuste, coloque Kp = 1 e Ki = Kd = 0. Em seguida, dê valores ao termo Kd, por fim ajuste Ki para zerar o erro de regime estacionário.

Nos Quadros 2 e 3 é possível visualizar dicas para ajustar com eficiência as constantes Kp, Ki e Kd.

**Quadro 2** – Comportamento dos parâmetros Proporcional, Integral e Derivativo.

| **Parâmetro** | **Ao aumentar, o processo:** | **Ao diminuir, o processo:** |
|---|---|---|
| Proporcional | Torna-se mais rápido. | Torna-se mais lento. |
| | Geralmente se torna mais estável, logo, menos oscilante. | Geralmente se torna mais instável, logo, mais oscilante. |
| Integral | Torna-se mais rápido, atingindo rapidamente o *setpoint*. | Torna-se mais lento, demorando para atingir o *setpoint*. |
| | Fica mais instável. | Fica mais estável. |
| | Tem mais *overshoot*. | Tem menos *overshoot*. |
| Derivativo | Torna-se mais lento. | Torna-se mais rápido. |
| | Tem menos *overshoot*. | Tem mais *overshoot*. |

**Fonte:** Adaptado de Novus (2020).

**Quadro 3** – Dicas para ajuste manual das constantes Kp, Ki e Kd.

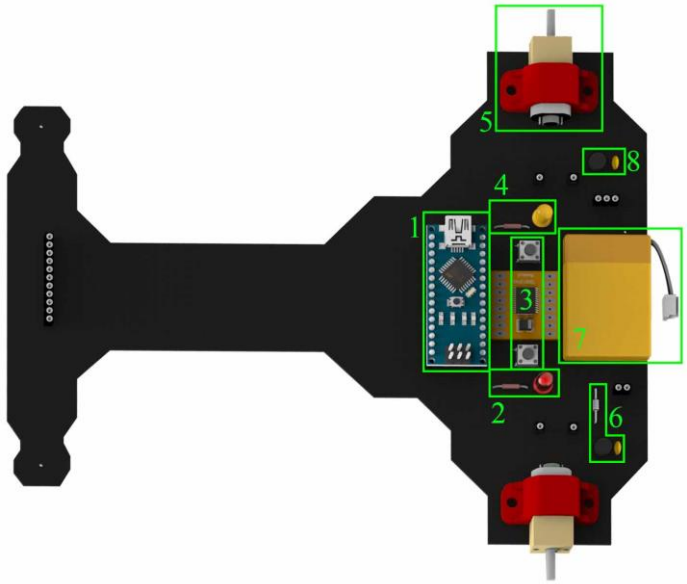
| **Se o desempenho do processo ...** | **Tente uma a uma as opções:** |
|---|---|
| Está quase bom, mas o *overshoot* está um pouco alto. | Aumentar Kp em 20%<br>Diminuir Ki em 20%<br>Aumentar Kd em 50% |
| Está quase bom, mas não tem *overshoot* e demora para atingir o *setpoint*. | Diminuir Kp em 20%<br>Aumentar Ki em 20%<br>Diminuir Kd em 50% |
| Está bom, mas MV está sempre variando entre 0% e 100% ou está variando demais. | Aumentar Kp em 20%<br>Diminuir Kd em 50% |
| Está ruim após a partida, o transitório dura vários períodos de oscilação, que reduz muito lentamente ou não reduz. | Aumentar Kp em 50% |
| Está ruim. Após a partida avança lentamente em direção ao *setpoint*, sem *overshoot*. Ainda está longe do *setpoint*. | Diminuir Kp em 50%<br>Aumentar Ki em 50%<br>Diminuir Kd em 70% |

**Fonte:** Adaptado de Novus (2020).

## 5. Funcionamento do Sistema Proposto

Utilizando o recurso de visualização 3D do *software* Proteus é possível visualizar a posição dos componentes eletroeletrônicos dispostos na PCI. Esse recurso torna possível prever possíveis erros de montagem, facilitando as conexões e a utilização. A Figura 15 foi levada ao *software* Solid Edge 2019 e foi renderizada para obter uma visualização aprimorada dos componentes eletroeletrônicos na face superior da PCI.

**Figura 15** – Visualização 3D do Protótipo.




**Fonte:** Produção do próprio autor (2020).

A Tabela 2 mostra os componentes, valores utilizados e exemplifica o funcionamento de cada bloco mostrado na Figura 15.

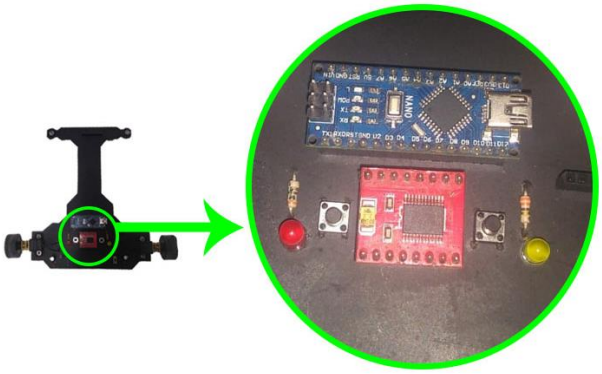
**Tabela 2** – Explicação dos blocos.

| Bloco | Explicação |
|---|---|
| **1** | Arduino Nano. |
| **2** | Led para indicar que o robô está alimentado. |
| **3** | Botões de navegação para calibragem e iniciar. |
| **4** | LED para informar o termino da calibração e o tempo para o robô iniciar. |
| **5** | Motor *Micro Metal GearMotor MP* 6V;3000RPM; 10:1. |
| **6** | Diodo para proteger de alimentação invertida e capacitores (um cerâmico de 100 nF e um capacitor de 100 µF), ambos em paralelo com o ponto de alimentação dos motores, situado no *driver* TB6612FNG. Os capacitores são utilizados para suprimir transientes de tensão proveniente da FEM dos motores. |
| **7** | Bateria Zippy com 500mAh e 7.4V. |
| **8** | Capacitores (um cerâmico de 100 nF e um capacitor de 100 µF), em paralelo com alimentação de 5V do Arduino para suprimir transientes de tensão e servir como filtro para ruídos de alta frequência. |

**Fonte:** Produção do próprio autor (2020).

Na Figura 16 é possivel visualizar que o robô contém duas chaves tácteis. A chave táctil da esquerda serve para indicar ao robô que o mesmo se encontra sobre a linha, ao pressionar a referida chave e estando os sensores do robô sobre as partes brancas e preta (linha) da pista, o robô realizará automaticamente a calibração dos sensores. A chave táctil do lado direito tem a funçao de dar a partida após o robô ter sido calibrado.

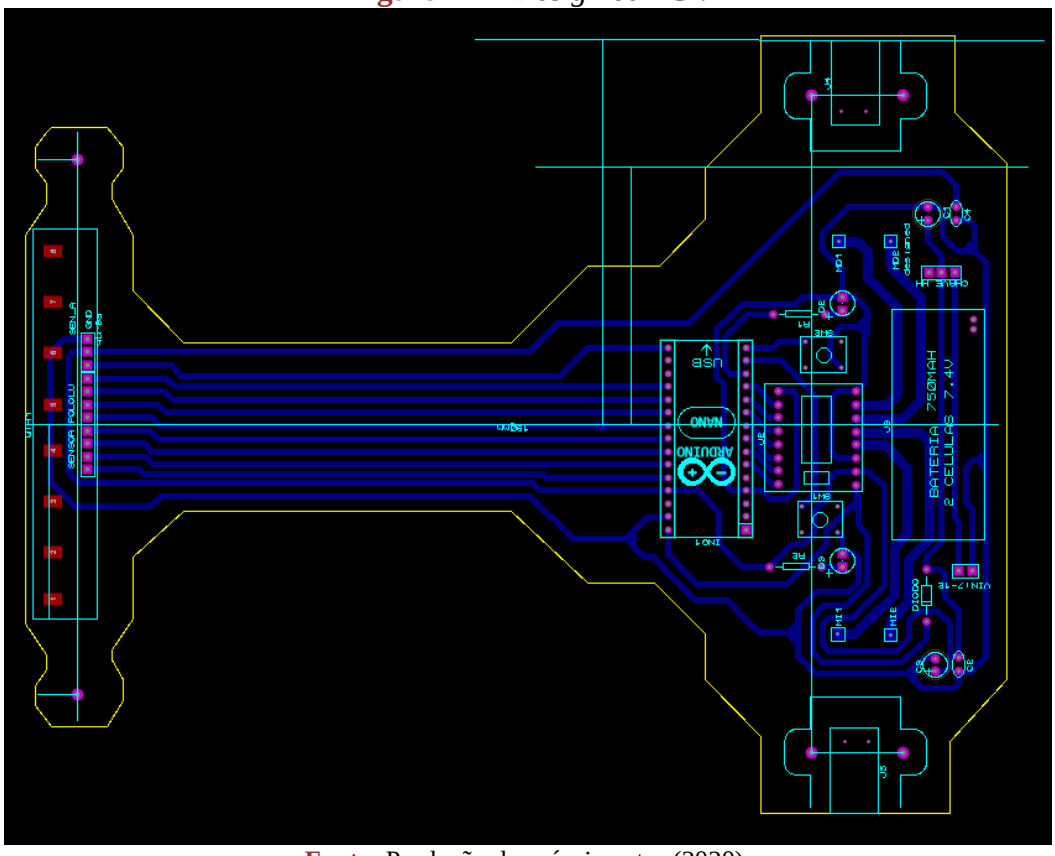
**Figura 16** – Chaves tácteis do robô.



**Fonte:** Produção do próprio autor (2020).

O *design* da PCI, mostrado na Figura 17, foi desenvolvido no *software* Proteus. Durante seu desenvolvimento, a disposição dos componentes eletrônicos foi pensada da melhor maneira possível, a fim de conservar a simetria do projeto e favorecer a distribuição de peso. Além do mencionado, preocupou-se em manter a correta espessura das trilhas, considerando os tipos de espessuras diferentes para trilhas de alimentação e de sinais elétricos de baixa potência.

**Figura 17** – *Design* da PCI.

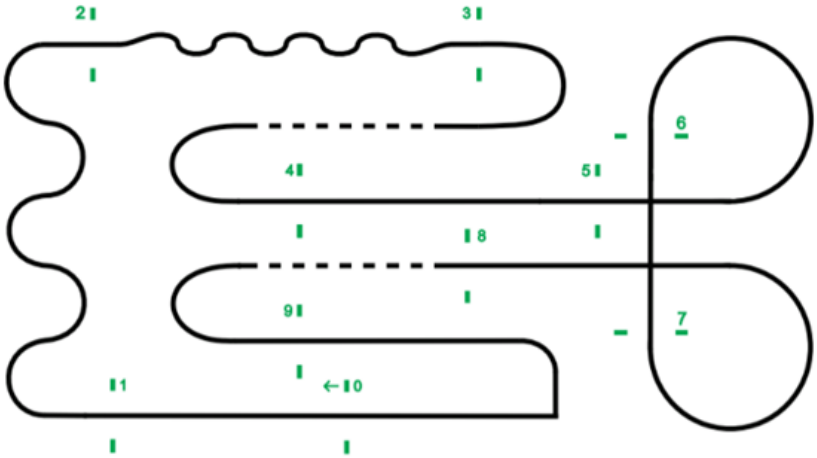



**Fonte:** Produção do próprio autor (2020).

## 6. Resultados

O protótipo desenvolvido seguiu a linha perfeitamente em uma superfície branca (lona) com linha preta. Ele conseguiu passar por tracejados, fazer curvas de 90 graus e curvas com ângulos bem reduzidos, conforme a Figura 18.

**Figura 18** – Trajeto da pista de seguidor de linha utilizada para realização dos testes.




**Fonte:** Produção do próprio autor (2020).

Após a conclusão de todos os testes, seguramente, pode-se afirmar que o robô autoguiado seguidor de linha desenvolvido foi finalizado com êxito, como mostra a Figura 19.

**Figura 19** – Robô autoguiado seguidor de linha desenvolvido.

**Fonte:** Produção do próprio autor (2020).

## 7. Conclusão

Após todos os estudos, desenvolvimento do projeto e realização dos testes de funcionamento do robô, verificou-se algumas dificuldades, uma delas foi a construção da placa de circuito impresso, pois na ausência de uma *plotter*, o método utilizado foi o da transferência térmica, este que requer tempo e paciência para passar as trilhas da folha de papel fotográfico para a placa. Em contrapartida, fazer com que o chassi do seguidor de linha fosse uma placa PCI trouxe vantagens, a principal é que elimina a utilização de jumpers para fazer ligações entre o Arduino, ponte H e sensores, consequentemente evitando o mal contato.

O robô foi projetado para alunos que pretendem entrar na área da robótica e construir um seguidor de linha, além disso, o protótipo se tornou um excelente *kit* didático para a

realização de estudos dos controladores PID na disciplina de controle clássico, porque é possível realizar através do protótipo várias análises do comportamento do sistema através da variação das constantes Kp, Ki e Kd, tornando mais didática a visualização do comportamento da atuação do controle PID.

## Agradecimentos

Agradecemos aos de Laboratórios de Eletrônica 2, Usinagem e do GEPRO do Instituto Federal do Ceará Campus Sobral por terem possibilitado a criação do projeto desenvolvido.

## Referências

ADONAYT, I.; RUGE, R. **Metodo Basico Para Implementar Un Controlador Digital PID En Un Desarrollo De Aplicaciones A Microcontrolador PIC Para Bajo Costo**. 2015.

BASONI, R. C. **Sistema Integrado Para Supervisão E Controle De Equipamentos Eletroeletrônicos Utilizando Uma Rede De Comunicação Sem Fio De Microcontroladores**. Monografia apresentada no programa de Engenharia Elétrica do Instituto Federal do Espírito Santo.

Brito, R. S; Moita, F. M. G. S. C; Lopes, M. C. **Robótica Educacional: desafios e possibilidades no trabalho interdisciplina entre matemática e física.** Revista PUCSP, Ensino de Matemática em Debate, EMD, ISSN: 2358-4122, 2018.

CRAUC. **Tutorial Robot Seguidor de Línea Velocista**. Disponível em: https://grupocrauc.wixsite.com/crauc/linetronics. Acesso em 13 de Maio de 2020.

Escuela Ingeniería Robótica. **Como hace un Robot Velocista # 10 – Prueba final**. Disponível em:

https://www.youtube.com/watch?v=8NS-26lTXzA&feature=youtu.be. Acesso em 13 de Maio de 2020.

Loja Arduino Belém. **Arduino Nano V3.0 – com cabo usb.** Disponível em: https://www.arduinobelem.com.br/produto/arduino-nano-v3-0-com-cabo-usb/. Acesso em 04 de Junho de 2020.

Mercado Livre. **Bateria de LIPO ZIPPY 500 mah.** Disponível em: https://produto.mercadolivre.com.br/MLB-1201753966-bateria-lipo-74v-500mah-2s-25c-zippy-compact-21329-_JM?quantity=1#position=5&type=item&tracking_id=15da0591-52f6-45ff-81a4-1bc6bfffbfaf. Acesso em 04 de Junho de 2020.

NOERGAARD, T. **Embedded Systems Architecture – A Comprehensive Guide for Engineers and Programmers.** Burlington: Elsevier, 2005.

NOVUS. **Indo Além do P.I.D.** Disponível em: https://www.novus.com.br/downloads/Arquivos/indoalemdocontrolepid.pdf. Acesso em 12 de Maio de 2020.

NOVUS. **Controle PID Básico.** Disponível em: https://www.novus.com.br/artigosnoticias/arquivos/ArtigoPIDBasicoNovus.pdf. Acesso em: 20 de Maio de 2020.

POLOLU. **Robotics & Electronics.** Disponível em: https://www.pololu.com. Acesso em: 20 de Maio de 2020.

ROBOTSHOP. **PID Tutorials For Line Following.** Disponível em:

https://www.robotshop.com/community/forum/t/pid-tutorials-for-line-following/13164. Acesso em 13 de Maio de 2020.

SOARES, Marcio José**. "COMO CONFECCIONAR PLACAS DE CIRCUITO IMPRESSO COM ACABAMENTO SEMI-PROFISSIONAL MÉTODO - Transferência Térmica".** Disponível em: http://www.arnerobotics.com.br/eletronica/como_confeccionar_placas_semi_profissionais.htm. Acesso em: 25 Maio de 2020.

TARZAN SOLUÇÕES EM ELETRÔNICA. **Soldar é Fácil – Dicas Básicas para Soldar Componentes Eletrônicos**. Disponível em: https://www.facebook.com/tarzancomponentes/photos/a%C3%AD-vai-uma-dica-de-como-soldar-bem-os-componentes-nesse-carnaval/2117923934988991/. Acesso em: 02 de Junho de 2020.

## Anexos

Abaixo segue algoritmo utilizado no protótipo desenvolvido.

**Fonte:** Algoritmo adaptado de (Escuela Ingeniería Robótica, 2017), de (CRAUC, 2020) e de (ROBOTSHOP, 2014).

```
#include <QTRSensors.h>

#define SW1      12
#define SW2      11
#define NUM_SENSORS 8
#define EMITTER_PIN 13
#define TIMEOUT 2500
#define KP 0.1
#define KI  0.0015
#define KD 3

int PWMA = 3;
int AIN2 = 4;
int AIN1 = 5;
int PWMB = 9;
int BIN2 = 8;
int BIN1 = 7;
int val;
int STANDBY = 6;
int LED = 2;
int ERRO = 0;
int ERRO_ANTERIOR = 0;
float Proporcional = 0;
float Integral = 0;
float Derivativo = 0;
float Velocidade;
float Velocidademotordireita = 0;
float Velocidademotoresquerda = 0;
int VelMax = 87;

QTRSensorsRC qtra((unsigned char[]) {A0, A1, A2, A3, A4, A5, A6, A7},NUM_SENSORS, TIMEOUT,
EMITTER_PIN);

unsigned int sensorValues[NUM_SENSORS];
unsigned int position = 0;

void setup()
{

  Serial.begin(9600);
  pinMode(PWMA,OUTPUT);
  pinMode(AIN1,OUTPUT);
  pinMode(AIN2,OUTPUT);
  pinMode(PWMB,OUTPUT);
  pinMode(BIN1,OUTPUT);
  pinMode(BIN2,OUTPUT);
  pinMode(LED,OUTPUT);
  pinMode(STANDBY,OUTPUT);
  digitalWrite(STANDBY, HIGH);
  delay(1500);

  do
  {
    val = digitalRead(SW2);
  }while (val == HIGH);

  analogWrite(PWMA,100);
  analogWrite(PWMB,0);
```

```
 digitalWrite(AIN1,LOW);
 digitalWrite(AIN2,HIGH);
 digitalWrite(BIN1,HIGH);
 digitalWrite(BIN2,LOW);

 for(int j = 0; j <100; j++)
 {
   digitalWrite(LED,HIGH);
   delay(20);
   qtra.calibrate();
   digitalWrite(LED,LOW);
   delay(20);
 }

 digitalWrite(LED,HIGH);
 delay(1000);
 digitalWrite(LED,LOW);
 delay(1000);
 digitalWrite(LED,HIGH);
 delay(1000);
 digitalWrite(LED,LOW);
 analogWrite(PWMA,0);
 analogWrite(PWMB,0);
 digitalWrite(AIN1,LOW);
 digitalWrite(AIN2,HIGH);
 digitalWrite(BIN1,HIGH);
 digitalWrite(BIN2,LOW);

 do
 {
   val = digitalRead(SW1);
 }while (val == HIGH);

 analogWrite(PWMA,255);
 analogWrite(PWMB,255);
 digitalWrite(AIN1,LOW);
 digitalWrite(AIN2,HIGH);
 digitalWrite(BIN1,HIGH);
 digitalWrite(BIN2,LOW);
 }

void loop()
{

 if( (analogRead(A0)<=500)&&(analogRead(A1)<=500) && (analogRead(A2)<=500)&&
     (analogRead(A3)<=500)&&(analogRead(A4<=500) && (analogRead(A5)<=500)&&
     (analogRead(A6)<=500) && (analogRead(A7)<=500)))
 {
   analogWrite(PWMA,200);
   analogWrite(PWMB,200);
   digitalWrite(AIN1,LOW);
   digitalWrite(AIN2,HIGH);
   digitalWrite(BIN1,LOW);
   digitalWrite(BIN2,HIGH);
 }
 else
     {
      Serial.print(A0);
      qtra.read(sensorValues);
      position = qtra.readLine(sensorValues,QTR_EMITTERS_ON,0);
      ERRO = ((position) - (3500));
      if(ERRO <=- 3500)
      {
        analogWrite(PWMA,75);
```

```
        analogWrite(PWMB,180);
        digitalWrite(AIN1,LOW);
        digitalWrite(AIN2,HIGH);
        digitalWrite(BIN1,LOW);
        digitalWrite(BIN2,HIGH);
       }
       else
          if(ERRO >= 3500)
          {
           analogWrite(PWMA,75);
           analogWrite(PWMB,180);
           digitalWrite(AIN1,HIGH);
           digitalWrite(AIN2,LOW);
           digitalWrite(BIN1,HIGH);
           digitalWrite(BIN2,LOW);
          }
          else
             {
              Proporcional = ERRO*KP;
              Integral = (ERRO + ERRO_ANTERIOR)*KI;
              Derivativo = (ERRO - ERRO_ANTERIOR)*KD;
              Velocidade = Proporcional + Integral + Derivativo;
              if(Velocidade  > VelMax)
              {
               Velocidade = VelMax;
              }

              else
                  if(Velocidade < -VelMax)
                 {
                   Velocidade = -VelMax;
                  }
              Velocidademotordireita = VelMax - Velocidade;
              Velocidademotoresquerda = VelMax + Velocidade;
              analogWrite(PWMA,Velocidademotordireita);
              analogWrite(PWMB,Velocidademotoresquerda);
              digitalWrite(AIN1,LOW);
              digitalWrite(AIN2,HIGH);
              digitalWrite(BIN1,HIGH);
              digitalWrite(BIN2,LOW);
              ERRO_ANTERIOR = ERRO;
             }
    }
}
```