

**UNIVERSIDADE FEDERAL DE OURO PRETO**  
**Mestrado em Ciência da Computação**

**Relatório de Projeto e Desenvolvimento do Carro Robô  
Seguidor de Linha Utilizando Controle Proporcional**

Disciplina:  
Sistemas Embutidos Avançados

Autor:  
Rodolfo Labiapari Mansur Guimarães -  
[rodolfolabiapari@decom.ufop.br](mailto:rodolfolabiapari@decom.ufop.br)

Professor:  
Ricardo Augusto Rabelo Oliveira - [rrabelo@gmail.com](mailto:rrabelo@gmail.com)

Ouro Preto - MG  
26 de maio de 2017

# Sumário

<b>I Especificação de Projeto Embarcado</b>	<b>1</b>
<b>1 Introdução</b>	<b>1</b>
<b>2 Especificações</b>	<b>1</b>
<b>3 Referencial Teórico</b>	<b>2</b>
3.1 Malhas em um Sistema de Controle . . . . .	2
3.1.1 Malha Aberta . . . . .	2
3.1.2 Malha Fechada . . . . .	2
3.2 Sistema Controlador Proporcional Usando Equação Diferencial Lineal de Primeira Ordem . . . . .	3
3.3 Velocidade Angular . . . . .	5
<b>4 Elementos do Projeto</b>	<b>5</b>
4.1 Visão Geral da Especificação . . . . .	5
4.2 Sensores do Carro $\mathcal{A}$ . . . . .	6
4.2.1 Sensor de Luz - Fototransistor . . . . .	6
4.2.1.1 Tecnologia . . . . .	6
4.2.1.2 Propósito do Projeto . . . . .	7
4.2.2 Sensores de Movimento . . . . .	8
4.2.2.1 Codificador de Rotações . . . . .	8
4.3 Atuadores . . . . .	9
4.3.1 Motores de Corrente-Contínua . . . . .	9
4.4 PWM . . . . .	9
4.4.1 Engrenagens de Redução . . . . .	10
4.4.2 Aplicação . . . . .	10
4.5 Visão Geral do Controle . . . . .	11
4.5.1 Comportamentos de um Sistema Sem Controle . . . . .	12
4.5.2 Comportamentos Sobre Controle Proporcional . . . . .	12
4.6 Microcontrolador . . . . .	13
4.7 Alimentação . . . . .	13
4.8 Nuvem . . . . .	14
<b>5 Modelamento Matemático</b>	<b>15</b>
5.1 Teoria da Equação Diferencial Linear . . . . .	15
5.2 Definições . . . . .	15
5.2.1 Teoria de Controle Proporcional (P) . . . . .	16
<b>II Desenvolvimento</b>	<b>17</b>
<b>6 Construção e Uso de Serviços e Equipamentos</b>	<b>17</b>
6.1 Carro Robô $\mathcal{A}$ . . . . .	17
6.1.1 Programação . . . . .	17
6.2 Nuvem . . . . .	18

6.3	Carro Robô $\mathcal{B}$	20
6.3.1	Programação	21
<b>7</b>	<b>Relação de Recorrência para o Modelo Implementado</b>	<b>21</b>
<b>8</b>	<b>Análise dos Componentes e Obstáculos</b>	<b>22</b>
8.1	Motores de Corrente Contínua com Suas Caixas de Reduções	22
8.1.1	Tempo de Atuação	22
8.1.2	Partida dos Motores	22
8.1.3	Uso de dois <i>packs</i> de pilha	23
8.1.4	Controle de Velocidade Utilizando Encoder	23
8.2	Equipamentos para a Aprendizagem de Trajeto e Reprodução	23
<b>9</b>	<b>Análise Analítica</b>	<b>24</b>
9.1	Análise da Alta Rotação	25
9.2	Análise da Baixa Rotação	25
<b>10</b>	<b>Discussão</b>	<b>26</b>
<b>11</b>	<b>Conclusão</b>	<b>27</b>
<b>12</b>	<b>Códigos</b>	<b>29</b>
12.1	Código em Linguagem Arduino	29
12.1.1	ploudy Node MCU	29
12.2	Código em Linguagem Python	41
12.2.1	ploudy Server	41

# Parte I

# Especificação de Projeto Embarcado

## 1 Introdução

Atualmente é possível construir Carro Robô Seguidor de Linha utilizando poucos componentes e com poucas linhas de programação, além da facilidade em encontrar em sites de venda de eletrônicos kits de sensores e atuadores de baixo custo prontos para serem acoplados à placa de prototipagem e assim a construção de um carro seguidor.

O desenvolvimento do sistema controlador também segue o mesmo princípio de facilitação de uso. Utilizando uma plataforma de prototipagem tal como Arduino, é possível escrever o controle de um carro seguidor com poucas linhas de instruções. Isso pode ser levado como um desafio até para crianças e adolescentes com criatividade e entusiasmo para desenvolver um sistema completo e funcional como incentivo à robótica.

Na Seção 2 será exibido a especificação deste trabalho e na Seção 3 é feito toda a introdução teórica sobre o tema. Em Seção 4 é descrito com detalhes todos os elementos que serão utilizados para fazer este projeto ser concretizado, em 5 é feito uma exibição das definições matemáticas que o projeto levará em consideração para seu funcionamento. Na Segunda parte do documento (Parte II), é relatado como foi realizado a Construção do Projeto (Seção 6), a Relação de Recorrência (Seção 7), os Obstáculos (Seção 8), Análise Analítica (Seção 9) e por fim a Conclusão (11).

## 2 Especificações

Abaixo é escrito a especificação segundo Professor Rabelo Oliveira rrabelo@gmail.com.

Neste trabalho, o robô será controlado por uma instância externa na Cloud, que irá efetuar a leitura dos dados transmitidos pelo robo via NodeMCU para retornar como feedback para os atuadores.

O caminho executado/aprendido pelo robô deverá ser usado para controlar um segundo robô que não possui sensores. Considere o ponto de partida similar para ambos.

A programação do segundo robo será enviada pela Cloud que contem os dados e algoritmo do primeiro robo.

O trabalho consistirá em duas entregas

Primeira Entrega - Especificação do projeto: deverá conter de maneira detalhada as seguintes características:

- a- Referencial teórico
- b- Proposta dos sensores do primeiro robo
- c- Proposta dos atuadores dos robos, explicando como serão programados
- d- Alimentação do sistema, indicando como sera montada a parte da alimentação dos motores e do NodeMCU
- e- Modelo matemático considerando os dados dos sensores discretos.
- f- A SOLUÇÃO DA EQUAÇÃO DEVERÁ SER DISCRETA

SOMENTE DE PRIMEIRA ORDEM

g- Arquitetura para a execução do controle de primeira ordem nas nuvens e o envio dos dados pela internet, Indicar a solução de Cloud para uso (azure, amazon, watson, etc)

Data entrega:08/02/2017

### 3 Referencial Teórico

#### 3.1 Malhas em um Sistema de Controle

##### 3.1.1 Malha Aberta

A principal característica do sistema em malha aberta é a inexistência de realimentação.

Os valores assumidos pela variável de controle não dependem dos valores da variável de saída, sendo assim, a ação de controle é função apenas do processamento da variável de referência pelo controlador. Exemplo demonstrado em Figura 1

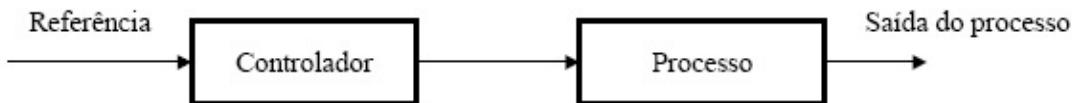


Figura 1: Sistema onde utiliza-se Malha Aberta. Sistemas que não possuem realimentação.

Dessa forma, o processamento de dados é puramente absoluto, sem observar dados de processamento anteriores tendo assim uma visão unicamente momentânea.

Um exemplo disso no caso aqui trabalho é o armazenamento dos valores de *Pulse-Width Modulation* (PWM) do primeiro carro, e a execução direta desses valores num segundo carro ignorando o erro relativo que os motores geram entre si no ambiente testado. Essa prática não permite o controle de velocidade dos motores de corrente contínua utilizando apenas os valores de PWM tornando o controle do tipo Malha Aberta.

##### 3.1.2 Malha Fechada

Já o sistema de controle em malha fechada, também é denotado de sistema de controle por realimentação.

A saída  $y$  é medida e comparada com a saída desejada, indicada através da referência  $r$ , para processamento através do controlador e a consequente definição da ação de controle  $u$ , exibido em Figura 2.

Neste trabalho é utilizado Malha Fechada. O sistema de realimentação utiliza como entrada, dados de processamentos anteriores criando assim uma proporcionalidade mais autêntica à situação a ser trabalhada. Ela será discutida ao decorrer do relatório.

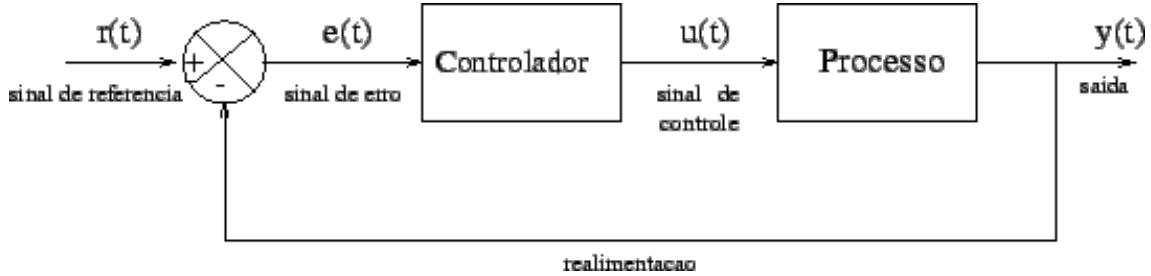


Figura 2: Sistema onde utiliza-se Malha Fechada. Sistemas onde existem um mecanismo de realimentação no controle de determinada ação.

### 3.2 Sistema Controlador Proporcional Usando Equação Diferencial Lineal de Primeira Ordem

Sistemas de controle é um sistema que possui o propósito de gerenciar o comportamento de outros dispositivos. Pode-se dizer que é uma interconexão de componentes conectados de maneira a comandar, controlar ou ajustar a si mesmo ou outro sistema obtendo uma precisão maior em seus procedimentos.

A teoria de controladores proporcionais se baseiam em sistemas realimentados. Tais podem ser divididos em basicamente três partes sendo elas:

- Sistema a ser controlado;
- Controlador; e
- Realimentação.

O sistema a ser controlado é constituído por atuadores capazes de efetuar as ações necessárias. Os outros dois elementos têm como finalidade fazer com que o desempenho do sistema possua estabilidade e opere com certa precisão e agilidade, seguindo as especificações uma vez estabelecidas. Uma vez estabelecido como o sistema deverá ser desenvolvido, os controladores e sua realimentação serão item essencial para que o processo ocorra de forma estável. Tudo isso pode ser observado no diagrama exibido na Figura 3.

A realimentação é o ponto chave para a diferenciação de sistemas controladores comuns. Um sistema de controle realimentado compara, instantaneamente, o valor de saída anterior, com o valor de referência existente na entrada do sistema como os sensores. O resultado desta comparação é o centro de toda adaptabilidade estudada e é denominado erro atuante. Este é levado ao controlador, que produz o chamado sinal de controle, cuja função resume-se em reduzir o desvio entre a saída e o sinal desejado. Como o nome sugere, em um controlador proporcional a saída do mesmo, também conhecido como sinal de controle (ou ação de controle), é diretamente proporcional ao sinal de erro, ou seja, ao erro atuante.

Sabendo-se em da proporcionalidade direta entre o sinal de controle e o sinal de erro, é possível afirmar que

$$a(t) \propto e(t) \quad (1)$$

onde o controle é diretamente proporcional à seu erro.

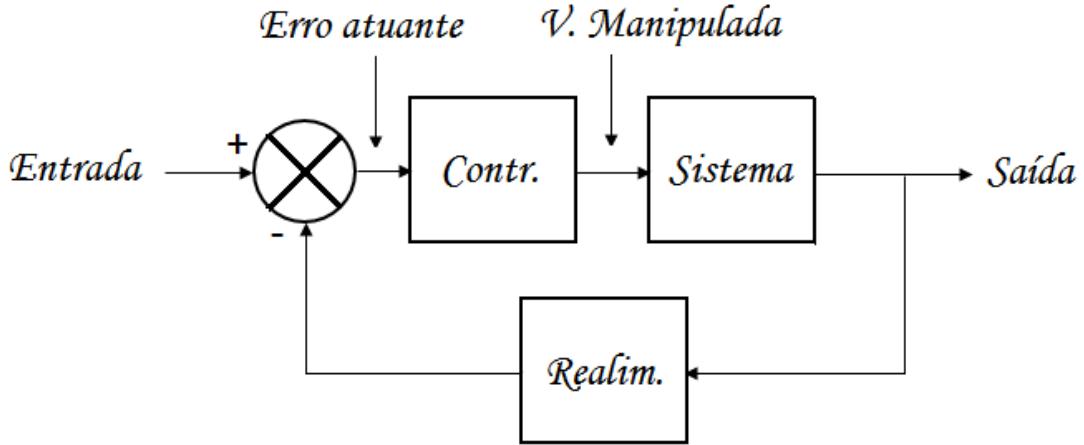


Figura 3: Estrutura de Sistema de Controle Geral Realimentado.

Entretanto, não é possível realizar operações matemáticas exatas com o sinal de proporção da fórmula. Para isso, deve-se admitir então uma constante de proporcionalidade entre as mesmas. Esta possui o nome de ganho proporcional e é representado pela variável  $f$ . Sendo assim

$$a(t) = f * e(t) \quad (2)$$

e dessa forma, o sistema matemático mostrado na Equação 2 poderá ser representado pelo diagrama exibido na Figura 4.

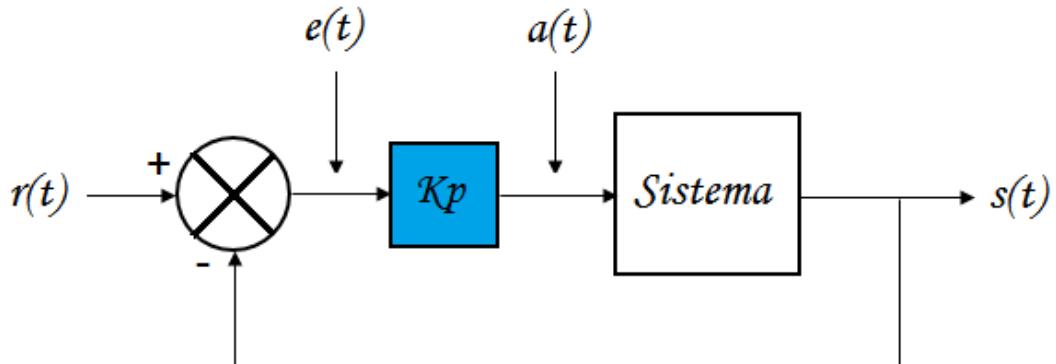


Figura 4: Sistema no Domínio do Tempo Geral.

Sendo assim, como fórmula final, temos

$$a(t) = f * (r(t) - s(t)) \quad (3)$$

onde  $a(t)$  é a atuação,  $e(t)$  representa o erro,  $r(t)$  é o valor de início de execução de controle e  $s(t)$  é saída do controle. Todos representando o valor no tempo  $t$ .

### 3.3 Velocidade Angular

A diferença gerada entre cada intervalo de tempo permite calcular a velocidade angular de cada motor como é exibido na Figura 5.

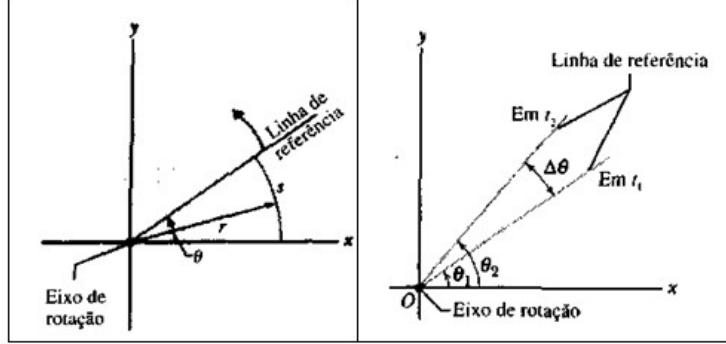


Figura 5: Ilustração do cálculo da velocidade angular.

Para exemplificação, suponha que motor está em movimento de rotação. Em um dado instante  $t_1$ , encontra-se à um certo ângulo  $\theta_1$  medido em relação à certo ponto. Após certo tempo, no instante  $t_2$ , encontra-se à um certo ângulo  $\theta_2$  medido em relação ao mesmo ponto. Denomina-se velocidade angular média a taxa de variação temporal de tais ângulos. Ou seja,

$$\omega = \frac{\theta_2 - \theta_1}{t_2 - t_1} = \frac{\Delta\theta}{\Delta t}, \begin{cases} \Delta\theta & \text{é o deslocamento angular} \\ \Delta t & \text{é a variação de tempo} \end{cases} \quad (4)$$

sendo a velocidade angular instantânea é determinada quando o valor de  $\Delta t$  tende a zero

$$w = \lim_{\Delta t \rightarrow 0} \frac{\Delta\theta}{\Delta t} = \frac{d\theta}{dt} \quad (5)$$

lembrando que  $2\pi \text{ rad} = 360^\circ$ .

## 4 Elementos do Projeto

### 4.1 Visão Geral da Especificação

Como já mencionado na Especificação (Seção 2), existirá dois robôs e uma computação em nuvem.

O trabalho consiste em replicar as ações de um carro  $\mathcal{A}$  repleto de sensores em um carro  $\mathcal{B}$  inexistindo os sensores de luz para detecção de faixa. Além disso, o processamento de dados e tomada de decisões será totalmente feita em nuvem, deixando os carros unicamente como plataformas de leitura de sensores e atuadores.

Como é representado na Figura 6, o carro  $\mathcal{A}$  terá sensores e atuadores. Seus dados não serão processados nele em si, mas sim em um computador externo ao sistema atuador. Após a execução do trajeto, a *cloud* teria dados suficientes, providos de vários sensores, para fazer o carro  $\mathcal{B}$  realizar o mesmo trajeto.

Para que isso possa ser realizado, cada carro terá seus dispositivos e um componente Wi-Fi para envio e recebimento de dados sem fio à *access points* e assim conexão com a *cloud*.

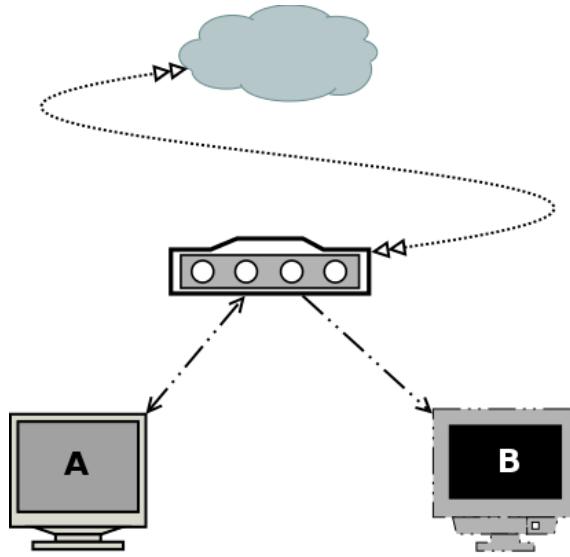


Figura 6: Diagrama Geral Abstrato do Sistema.

Deve-se atentar que os dois carros possuem características e propósitos diferentes e que todo o processamento é realizado na *cloud* e não nos carros sendo estes somente para captação de dados e execução de tarefa com seus atuadores.

Abaixo será descrito cada componente do sistema e algumas propriedades importantes.

## 4.2 Sensores do Carro $\mathcal{A}$

O projeto do carro  $\mathcal{A}$  utilizou-se de vários tipos de sensores.

Utilizar somente o sensor fototransistor seria suficiente para o projeto de seguido de linha, mas não para este em si que necessita de um item controlador. Sua função é unicamente para direcionar o carro informando o controle a situação para que ele não saia da linha completando o trajeto. Usando somente ele, é impossível que o carro saiba sua posição e seus movimentos para a reprodução no segundo carro.

Para contornar este problema, necessita-se de do sensor de contador de giros da roda chamado *Encoder*. Ele, junto com o fototransistor, serão mencionados nas Seções 4.2.1 e 4.2.2.

### 4.2.1 Sensor de Luz - Fototransistor

#### 4.2.1.1 Tecnologia

Para o carro, será utilizado o sensor fototransistor. Em sua superfície, existirá um LED que fará a iluminação da área no qual refletirá sobre a superfície e chegará até o sensor fototransistor. Como o seguidor de linha move sobre dois tipos de superfície (branca e preta), é possível identificar quando ele sairá da direção correta. Essa ideia melhor visualizada com a Figura 7.

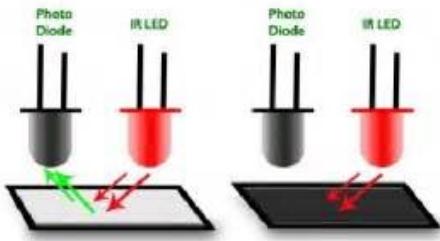


Figura 7: Funcionamento do sensor Fototransistor.

#### 4.2.1.2 Propósito do Projeto

O projeto utilizará não somente dois sensores deste tipo mas uma série deles, posicionados logo à extremidade da faixa de direção como é exibido na Figura 8.

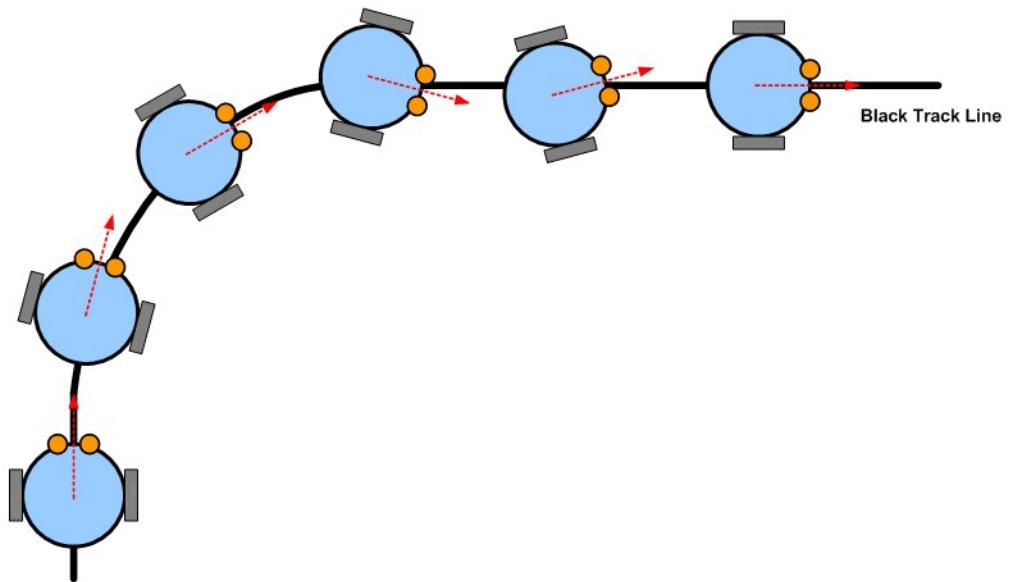


Figura 8: Posição dos Sensores. Exemplo utilizando dois sensores.

Utilizando uma série de sensores, cada sensor será responsável por avaliar se o carro saiu da linha original. Quanto mais à extremidade a faixa estiver em relação à série de sensores, maior será o erro dela e assim, a consequente uma correção mais rápida. Quanto maior a quantidade de arranjo de sensores melhor será a performance do carro pelo motivo que mais sensores representarão melhor o nível do erro. Por este motivo, será utilizado seis sensores, posicionando três em cada extremidade da faixa. Eles serão dispostos o mais próximo possível de seu adjacente pois a resolução de operação será melhor do que distâncias grandes como três centímetros ou mais. Um exemplo com 8 sensores é exibido na Figura 9.

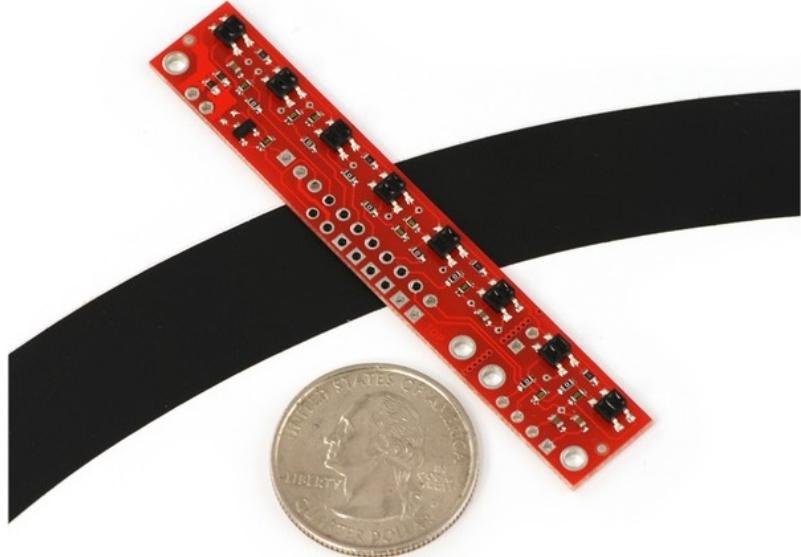


Figura 9: Sensores posicionados em uma *board* em série.

#### 4.2.2 Sensores de Movimento

Como já descrito na Seção 4.2.1, sensores fototransistores não conseguem realizar gravações de que detectam movimento. Para suprir essa necessidade utilizar-se-á um sensor que contará a quantidade de rotações que a roda fará.

##### 4.2.2.1 Codificador de Rotações

Um *rotary encoder*, é um dispositivo eléctro-mecânico que converge a movimento do eixo para valores digitais/analógicos. O sensor utilizado foi do tipo incremental/relativo no qual a saída do *encoder* provê informações sobre o movimento do eixo no qual é possível obter informações de velocidade, distância e posição.

Ele trabalha percebendo alteração de posição no qual deve ser analisado e processado por um componente externo a ele, que no caso deste trabalho é o próprio controlador.

A Figura 10 exibe o mecanismo de leitura de movimento dos *encoders* utilizados no trabalho.

Cada roda possui o formato 65mm por 30mm de largura. Uma rotação completa, segundo a fórmula  $P = d\pi$ , mostra que o carro percorreu  $204,2035224833 = 64\pi$  mm, ou seja, cerca de 20,42 centímetros.

O *encoder* utilizado em cada roda possui 20 furos. Isso permite um total de 40 *trick* ao analisar espaços vazados ou não. Sendo assim é possível calcular movimentos a cada  $9^\circ$  rodados, significando detecções de movimentos a cada  $\frac{360}{20,42} = \frac{9}{x} \equiv 0,5105$  centímetros percorridos.

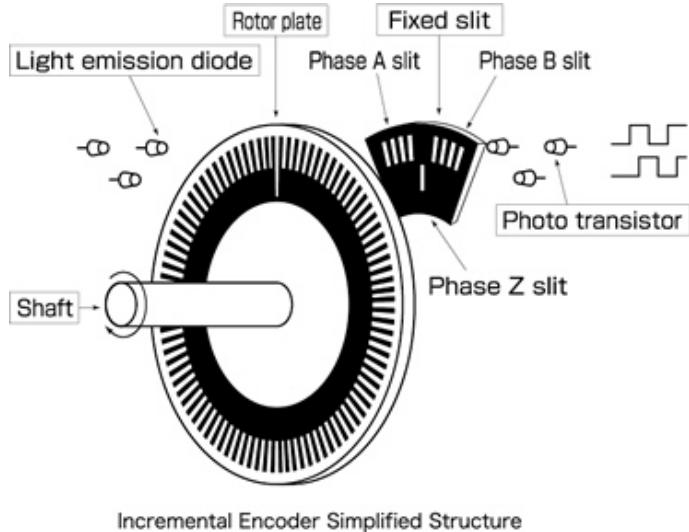


Figura 10: Visão técnica sobre o funcionamento do *encoder* para detecção de movimento.

## 4.3 Atuadores

### 4.3.1 Motores de Corrente-Contínua

Motores de corrente contínua baseia-se no fluxo ordenado de elétrons sempre numa direção. Pode ser alimentado por tensões desde 1,2V até 24V dependendo de sua característica física e dessa forma, pode-se dizer que é um equipamento que é capaz de converter energia elétrica em mecânica.

Fisicamente, possui um conjunto de ímãs permanentes fixos, criando um campo magnético fixo enquanto o rotor é percorrido por uma corrente. Por meio de escovas e comutadores, a direção da corrente é alterada constantemente, fazendo com que o rotor gire continuamente. De forma analítica, se ímãs permanentes são usados para gerar o campo magnético, o torque de saída  $T$  é proporcional ao fluxo magnético  $\phi$  e à corrente nos enrolamentos do motor  $i$ .

Possuem como vantagens a qualidade de serem bons para quaisquer tamanhos de robôs, engrenagens para redução de inércia, confiáveis, de baixa manutenção. Como pontos negativos possuem baixa rigidez, necessidade de engrenagens para inúmeros projetos, folgas, custo, peso e necessidade de frenagem caso não alimentado.

Plataformas de prototipagem fornece controles de tais atuadores por meio de sinais PWM que serão descritos a seguir.

## 4.4 PWM

Os controles de potência PWM (*Pulse Width Modulation*) são altamente indicados para aplicações em robótica pela possibilidade de se manter o torque mesmo em baixas velocidades.

Nos controles PWM o que se faz é variar a largura do pulso de uma tensão retangular aplicada à carga de modo obter-se um controle sobre a potência média aplicada, como é demonstrado na Figura 11.

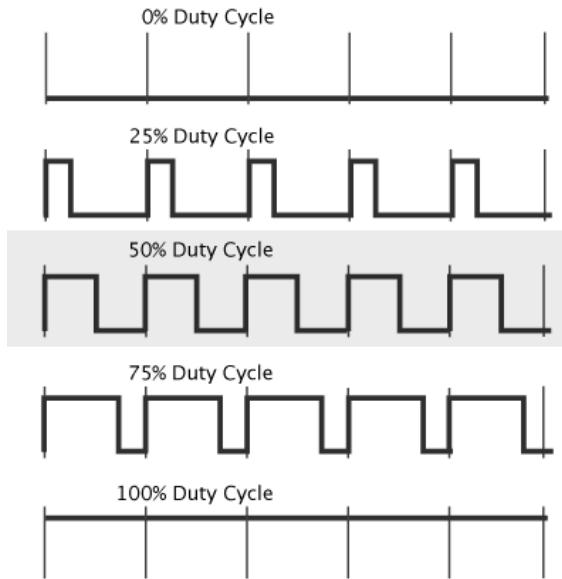


Figura 11: Sinais PWM.

#### 4.4.1 Engrenagens de Redução

Motores elétricos giram em altas velocidades, e devem ser usados em conjunto com engrenagens de redução para aumentar seu toque e diminuir sua velocidade.

Naturalmente, isso aumenta o custo, o número de peças, a folga, a inércia do corpo rotativo, e assim por diante, mas também a resolução do sistema já que com os trens de engrenagem é possível girar o elo de um pequeno ângulo e é demonstrado na Figura 12.

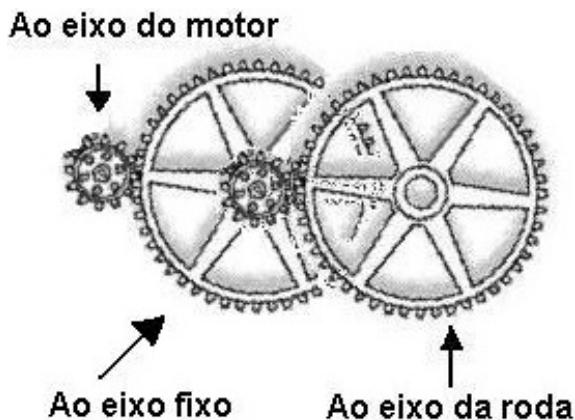


Figura 12: Sistema simples de trens de engrenagens no qual gerará reduções.

#### 4.4.2 Aplicação

Os motores possuem propriedade física que *não* permite o cálculo exato de distância e velocidade do equipamento sem a utilização de sensores de marcação de passo/angulação. Para isso, será utilizado vários sensores para captação de dados do ambiente, onde

seus dados serão enviados para a *cloud*, processados e recebidos novamente para assim realizar a atuação no carro. No caso do Carro *B*, não haverá processamento, mas sim a atuação, sendo este somente um receptor escravo.

Serão acoplados à *board* utilizando um *motor shield* intermediário entre o microcontrolador e os motores. Tal *motor shield* realizará o processo de interface de controle e energização dos motores criando assim um sistema estável. Seu nome é L293DD e possui suporte total para interface de pinos NodeMCU. Seu sistema de operação utiliza Ponte-H dupla e com isso é possível controlar até dois motores além de conectores para seleção de interface serial UART, SPI, e entrada analógica, além de conexões para habilitar as opções de *Enable* e *Reset* do microcontrolador. Ele pode ser visto por meio da Figura 13.

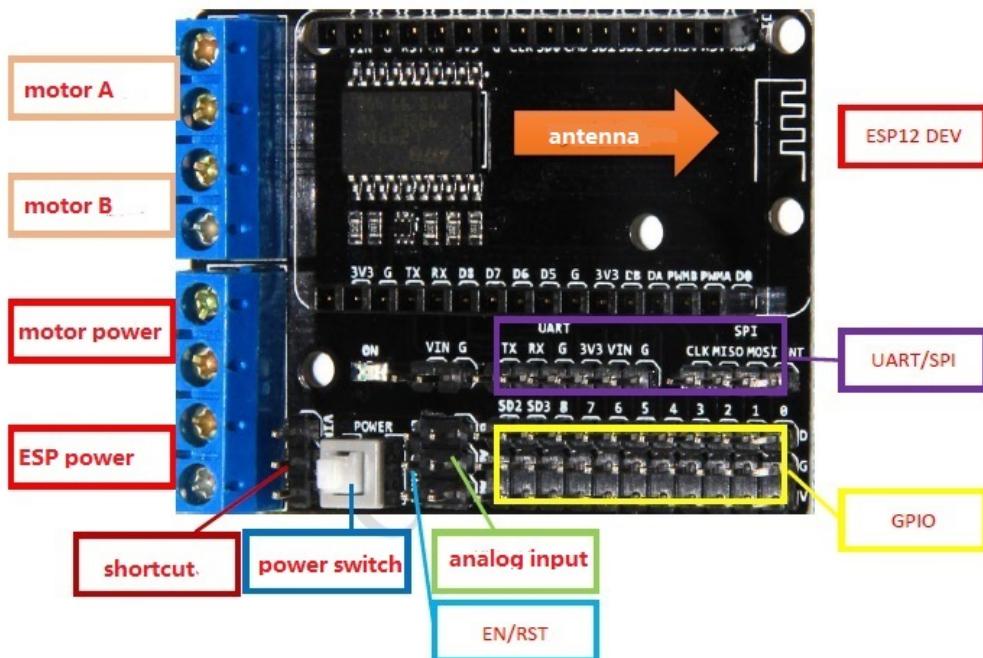


Figura 13: Motor *Shield* a ser Utilizado.

No conjunto também é incluído duas rodas de borracha com 65mm e 30mm de largura.

## 4.5 Visão Geral do Controle

O carro, ao andar pra frente, ficará numa velocidade<sup>1</sup> máxima fixa. Quando algum de seus sensores perceber que o carro saiu da linha gerando uma interrupção, o procedimento de correção de direção será acionada para avaliar os dados dos sensores e assim verificar os passos a serem realizados nos atuadores para que o carro volte a operar normalmente na direção correta em cima da faixa.

Existem dois tipos de comportamentos em um seguidor de linha comumente conhecido. Eles serão descritos e explicados a seguir.

---

<sup>1</sup>Velocidade monitorada pelo *encoder* e o controlador.

#### 4.5.1 Comportamentos de um Sistema Sem Controle

Neste sem realimentação mostrado na Figura 14, é possível perceber que seu erro após à primeiro desvio será grande e sua forma de conserto fará com que ele aumente ainda mais sua margem de erro. isso causará grande distúrbio em seu movimento criando assim um padrão geométrico construído por uma sequência de segmentos lineares alternados quanto à direção, formando linhas quebradas com alternância de ângulos salientes e reentrantes.

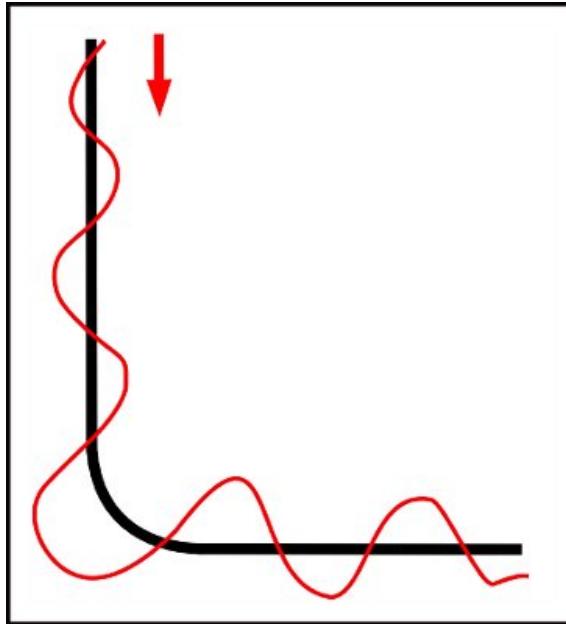


Figura 14: Trajeto esperado de um seguidor de linha comum.

Esse movimento oscilatório faz com que o carro gaste tempo em completar sua tarefa de percorrer a linha e também energia, item essencial para sistemas embarcados. Nesta situação relacionando à especificação do trabalho, o carro  $\mathcal{A}$  com seus movimentos oscilatórios gerarão um mapa do trajeto complexo o suficiente para que o carro  $\mathcal{B}$  não consiga finalizar sua tarefa. Tais movimentos oscilatórios podem fazer o carro  $\mathcal{B}$  se perder facilmente com o acúmulo de erros em cada movimento realizado.

#### 4.5.2 Comportamentos Sobre Controle Proporcional

Utilizando o controle proposto é possível que este padrão seja decrementado ao ponto de tornar o movimento do carro mais suave e portanto seu trajeto mais conciso quanto à linha do trajeto. O resultado esperado é exibido na Figura 15.

Um fato importante a se atentar é que, para que o carro funcione como esperado, o algoritmo, seus parâmetros e os parâmetros da fórmula de controle proporcional devem estar devidamente calibrados para que o carro obtenha a melhor performance possível.

Este trajeto faz com que seus movimentos sejam mais suaves, estáveis, ágeis e eficientes em relação ao mostrado na Figura 14, exibida anteriormente.

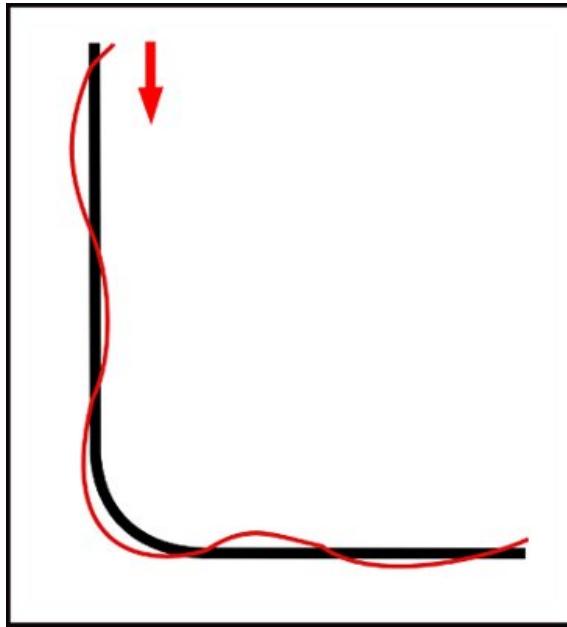


Figura 15: Trajeto esperado de um seguidor de linha utilizando Controle Proporcional.

## 4.6 Microcontrolador

Microcontrolador utilizado para este trabalho é o NodeMCU. É um plataforma IoT<sup>2</sup> *open-source*.

Como esperado de um microcontrolador para IoT o sistema possui integrado um componente de comunicação Wi-Fi para troca de dados. Utiliza linguagem de *script* Lua desenvolvida por brasileiros ou IDE e linguagem Arduino.

Suas especificações são uma CPU ESP8266 possuindo 128 KB de memória, 4 MB de armazenamento e suporta o sistema operacional chamado XTOS. Permite comunicação pelo Wi-Fi e USB onde também é energizada. Possui um total de 10 pinos de entrada e saída de propósito geral (GPIO) onde suportam funções como PWM, comunicação I<sup>2</sup>C e 1-wire. Além da antena Wi-Fi, possui também um conversor USB-TTL para comunicação serial.

A Figura 16 exibe um esquemático de seu protótipo.

Sua programação pode ser feita por diversas maneiras. As duas principais são utilizando um terminal serial para envio de comandos diretamente à placa usando a porta USB do controlador.

Como ela possui placa de comunicação Wi-Fi e um controlador poderoso, é possível também transformá-la num *Web Service* para que realize toda a comunicação sem fio.

## 4.7 Alimentação

Os motores devem ter como entrada 6V e possui velocidade média de  $90 \pm 10 \text{ rpm}$  com corrente de 250 mA sem carga. Sua corrente de partida pode chegar a 1mA.

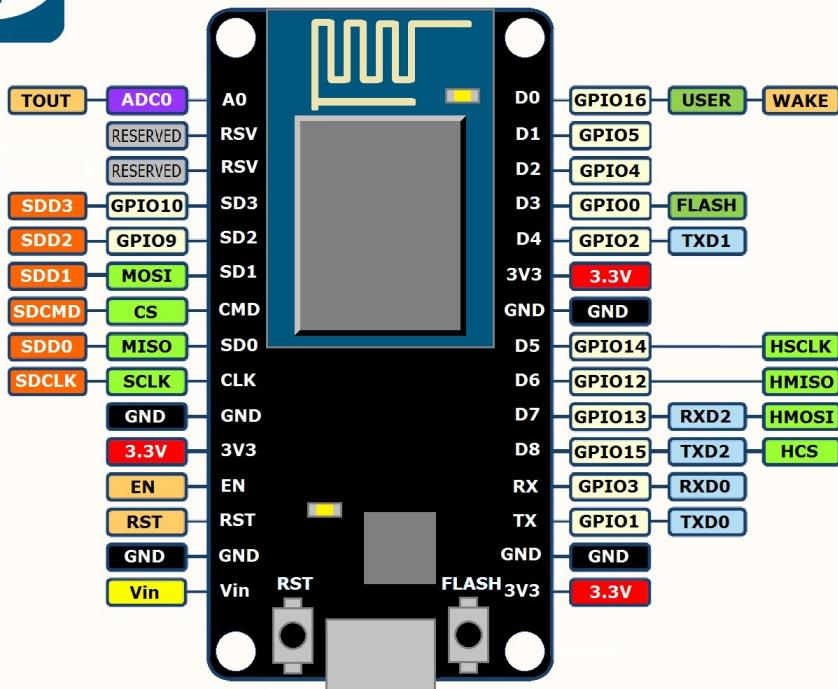
---

<sup>2</sup>*Internet of Things*, ou seja, internet das coisas.



## NodeMCU ESP-12 development kit V1.0

### PIN DEFINITION



[Arduining.com](http://Arduining.com)

Figura 16: O Microcontrolador NodeMCU e seus Componentes e I/O.

Utilizando um *jump*, é possível criar um curto de tal forma que os motores e o NodeMCU poderiam ser energizados por tal, permitindo total controle e a utilização de nenhum fio para alimentação do sistema.

Sua energização será realizada por meio de fonte externa utilizando dois *pack* de 4 pilhas em série.

Utilizar uma fonte por meio da interface USB diretamente no microcontrolador não seria suficiente para energizar todos os atuadores e sensores do sistema. Dessa forma, a fonte externa alimentará os atuadores e o microcontrolador e este alimentará todos os sensores nele contido.

## 4.8 Nuvem

Como os processadores não podem realizar processamento seguindo a especificação, utilizou-se de um servidor construído em Python para a comunicação e processamento de ambos os carros. Utilizar este tipo de sistema traz várias vantagens com redução de gasto energia nos carros e centralização de processamento e *backup* de informações em local seguro além da transparência do sistema de controle. Também permite que as configurações de execução do carro sejam feitas totalmente via servidor, já que este enviará controles aos carros.

Entretanto, latências altas em comunicações ou pacotes de informações corrompi-

das e interferências de ambiente podem tornar o sistema inconsistente ou até mesmo comprometido para seu propósito.

## 5 Modelamento Matemático

### 5.1 Teoria da Equação Diferencial Linear

Equações Diferenciais Lineares são equações que suas soluções podem ser somadas a fim de produzir uma nova solução.

É considerada linear quando satisfaz as características de que cada coeficiente a  $a_n$  e o termo de não-homogeneidade só dependem da variável independente, no caso  $x$ ; e a variável dependente, no caso  $y$ , e suas derivadas são de primeiro grau. Assim, ela deve representar a seguinte forma

$$f(x) = a_n(x) \frac{d^n y}{dx^n} + a_{n-1}(x) \frac{d^{n-1} y}{dx^{n-1}} + \cdots + a_1(x) \frac{dy}{dx} + a_0(x)y \quad (6)$$

No uso para problemas que utilizam Controle Proporcional descrito no Seção ??, basta desligar todas as derivadas e integrais da equação obtendo

$$f(x) = a_n(x) + a_{n-1}(x) + \cdots + a_1(x) + a_0(x)y \quad (7)$$

### 5.2 Definições

Para iniciar a discussão sobre sistema de controle proporcional, primeiramente será definido os termos utilizados.

**Alvo:** É a posição que desejamos que o carro esteja em relação à faixa, ou seja, centralizado respeitando a devida linha.

**Erro:** Diferença entre a posição atual e o *Alvo*. Pode ser qualquer valor no conjunto dos  $\mathbb{R}$ .

**Proporção:** Fator que determinará quão distante o carro está da linha. Por exemplo são as proporções: ‘para a esquerda’, ‘para a direita’, ‘para a extrema esquerda’, ‘pouco para a direita’, etc. É representado pela constante de variação  $K_p$ .

O procedimento de execução é baseado no seguinte pseudocódigo:

1. Realiza o cálculo inicial da posição atual.
2. Calcula o erro baseado na posição atual.
3. Ele então comandará os motores para fazer uma girada:
  - Brusca caso o erro for grande; ou
  - Leve caso o erro for pequeno.

Assim, basicamente, a magnitude do giro tomado é proporcional ao erro.

*4. Repete os passos até completar o objetivo inicial.*

Com esses passos, saímos de um controle simples para um Controle Proporcional mais eficaz.

São amplamente utilizados em situações que se baseiam em controladores eletrônicos chamados “*single-loop*”.

Abaixo é descrito a formulação matemática do conceito de controlador proporcional

### 5.2.1 Teoria de Controle Proporcional (P)

Produz um sinal de saída que é proporcional à parâmetro do erro

$$P_{\text{saída}} = K_p e(t) \quad (8)$$

onde:

$e(t)$ : Erro no tempo  $t$ .

$K_p$ : Constante relativa à proporção.

Esse método possui a propriedade de eliminar as oscilações do sinal de saída. Para tal, o sistema permanece sempre ligado e o sinal de saída é diferente de zero.

# Parte II

## Desenvolvimento

### 6 Construção e Uso de Serviços e Equipamentos

#### 6.1 Carro Robô $\mathcal{A}$

A construção do carro  $\mathcal{A}$  se baseou-se num sistema de sensores e atuadores que realiza leituras do ambiente por meio de interrupções de sistema e o envio desses dados para a nuvem. Tais dados são processados e retornados para o carro para assim ser realizado a atuação apropriada à situação segundo os cálculos do controle proporcional situado na nuvem.

Ele possui dois sensores fototransistores na sua dianteira (Figura 6.1) para detectar as faixas que orientam o carro e dois sensores encodificadores de rotação (Figura 6.1) para controle de velocidade/rotações.

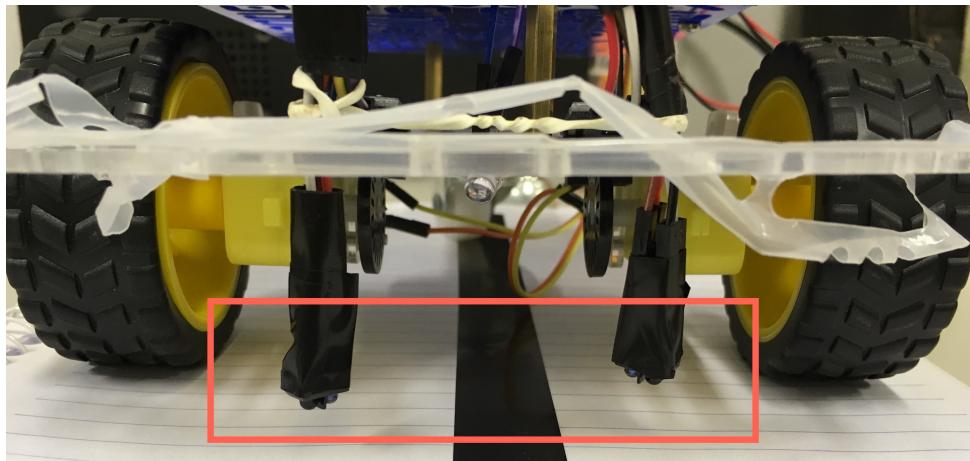


Figura 17: Carro Robô  $\mathcal{A}$  e os sensores fototransistores.

Além desses sensores, o carro é composto pois dois motores de corrente-contínua, dois *pack* de pilhas e um NodeMCU junto com seu *motor shield* e protoboard para confecção do circuito elétrico, como mostrado na Figura 6.1.

##### 6.1.1 Programação

O carro não possui nenhum tipo de processamento importante, sendo seu propósito único de leitura de sensores e atuação implicando numa programação simples de conexão com a nuvem, leitura, tratamento e envio dos dados e por fim sua atuação.

A leitura do *encoder* é feito por interrupção sistêmica no qual, a cada *trick* alterado, o sensor realiza uma interrupção para incrementar o valor do grau de rotação naquele intervalo de tempo  $t$ . Tais detalhes serão explicados na respectiva seção da descrição do controle na nuvem, na Seção sec:nuvem.

A detecção de faixa é feita por meio de interrupção de sistema gerando um procedimento de emergência de envio de dados para a nuvem a fim de obter resposta mais apropriada para a situação.

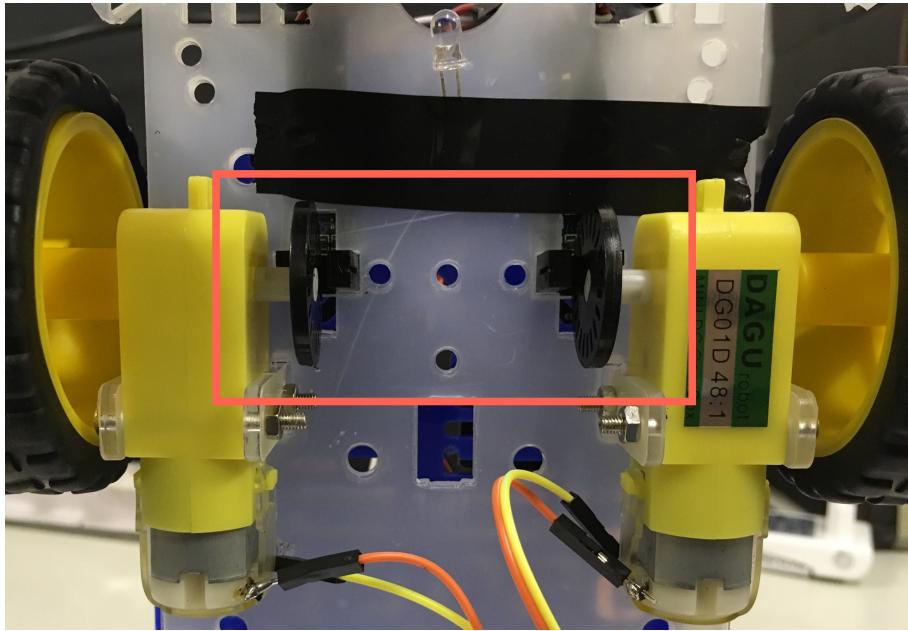


Figura 18: Carro Robô  $\mathcal{A}$  e os sensores encoders.

## 6.2 Nuvem

O servidor foi desenvolvido utilizando linguagem de programação Python versão 2.7 que possui todos os artefatos necessários para a construção e utilização de uma aplicação com *socket* para comunicação com aplicações externas.

A nuvem recebe os dados lidos pelos sensores do carro a cada instante de tempo, definido como 30 milissegundos, processa-os e responde o carro com os devidos comandos que ele deverá realizar sobre a situação imposta. Essa estratégia de arquitetura faz com que *o processamento realizado seja inteiramente pela nuvem* sendo a configuração de velocidade e de controle proporcional de execução do carro na pista sejam todas definidas via nuvem, eliminando a necessidade de compilação do carro a cada alteração de projeto.

O teste de controle proporcional descrito na Análise Teórica do trabalho baseou-se na situação de que, caso encontre a faixa em algum lado do carro, a respectiva roda teria sua velocidade reduzida a ponto de que o carro possua a velocidade angular (Seção 3.3) correta em cada roda para realizar a curva. Entretanto, os motores necessários não possuíam torque suficiente para o carro construído, impedindo este de realizar as movimentações necessárias como descrito na Seção 8.1.

De forma a contornar este problema, construiu-se outra estratégia de controle proporcional que respeitasse o tempo de atuação que motor necessita para o funcionamento prático. O algoritmo necessita de duas funções especiais para concluir seus cálculos sendo elas o Cálculo Proporcional (Algoritmo 1) e o Cálculo de Rotação (Algoritmo 2). O primeiro utiliza da Teoria de Controle Proporcional para gerar um valor de erro em relação à posição perfeita. Dado a posição atual  $\Delta\Phi$  e a referência esperada *reference*, calcula-se a proporção de erro. Caso a referência seja o propósito de parar o motor, o erro gerará um erro grande negativamente o suficiente para solicitar a parada total do motor requerido. Obtido o erro proporcional, é feito o cálculo de velocidade do motor a fim de que tenha controle fechado sobre sua

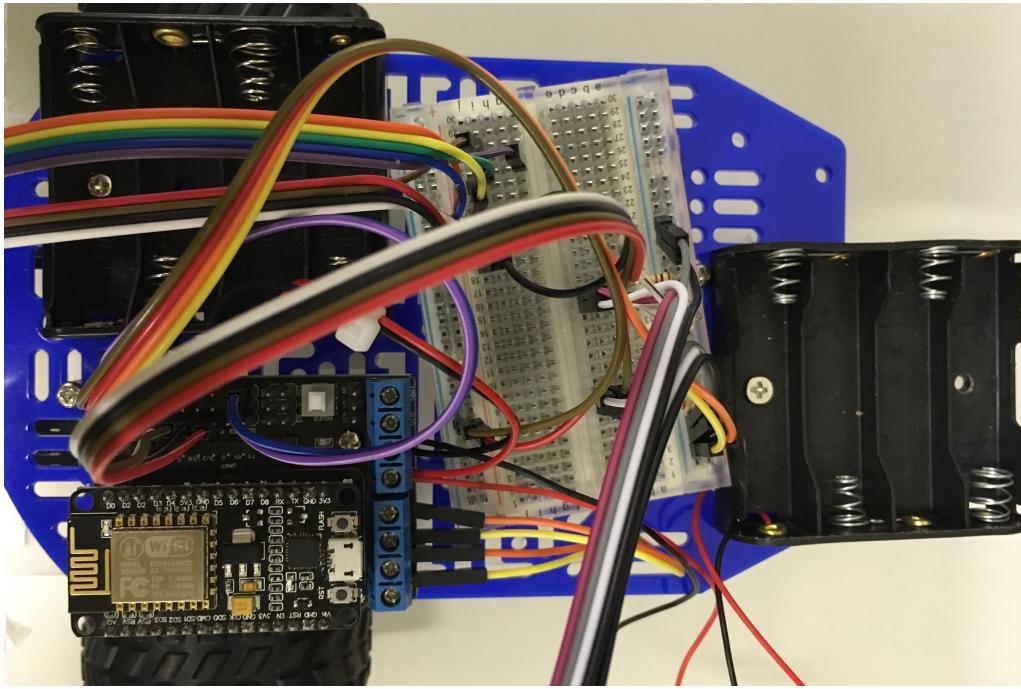


Figura 19: Carro Robô  $\mathcal{A}$ .

atuação.  $erro = 0$  indica que o motor está na velocidade ideal. Valores positivos indicam que a potência do motor deve aumentada a fim de aproximar do requerido e negativo possui a ideia de decremento.

---

**Algorithm 1** Cálculo da proporção do erro em relação à posição referência.

---

```

1: function PROPORTION( $\Delta\Phi$ , reference)
2:   position  $\leftarrow \Delta\Phi$ ;
3:   perfect_position  $\leftarrow$  reference;
4:   proportion  $\leftarrow$  perfect_position - position;
5:   error_value  $\leftarrow$  proportion *  $K_p$ ;
6:   return error_value;
```

---

**Algorithm 2** Cálculo da proporção de rotação dos motores sobre controle de malha fechada.

---

```

1: function MOTORS_SPIN( $\Phi$ , error,  $\Lambda$ )
2:   if  $\Lambda = NOT\_VISIBLE$  then
3:     rotation  $\leftarrow$  (error * 35.0 * ABS(max_rotation_motor - rotation)) / 100;
4:   else
5:     rotation  $\leftarrow$  -max_rotation_motor;
6:   if rotation > MAX then       $\triangleright$  Verify if PWM value is in correct interval
7:     rotation  $\leftarrow$  MAX;
8:   else if rotation < MIN then
9:     rotation  $\leftarrow$  MIN;
10:  return rotation;
```

---

Explicado as funções básicas, é possível descrever o funcionamento do algoritmo de controle (Algoritmo 3). O algoritmo receberá por parâmetro os valores de tempo, visualização de faixa do lado esquerdo e direito respectivamente, e somatório de *tricks* de cada lado do carro, sendo respectivamente  $t, \lambda_l, \lambda_r, \phi_l, \phi_r$  suas variáveis. Cada procedimento descrito no algoritmo deverá ser processado para cada motor separadamente.

O primeiro passo é o cálculo do deslocamento angular de cada motor no intervalo de tempo. Com o deslocamento, têm-se todos os dados de distância percorrido pelo carro. Verificando se o carro encontra-se sobre uma faixa, verifica-se seu erro proporcional. Caso o carro esteja sobre, o respectivo motor deverá parar enquanto o outro continua sua velocidade estipulada normalmente. A partir do momento que o carro volta a encontrar uma reta, o controle proporcional entra para calcular a potência requerida ao motor para conseguir inércia suficiente para deslocar e chegar à velocidade angular requerida que é dada por *reference*.

Tendo o erro referencial, calcula-se o valor de PWM pela função *motors\_spin* e todos os dados relevantes da execução são salvos fisicamente em arquivo. Ao final, os novos valores de potência são enviados aos motores para que seja realizada a sua atuação e em seguida uma nova leitura dos sensores.

---

### Algorithm 3 Controle Proporcional em Nuvem.

---

```

1: procedure PROPORTIONAL_CONTROL( $t, \lambda_l, \lambda_r, \phi_l, \phi_r$ )
2:    $\Delta\Phi \leftarrow \Phi_i - \Phi_{i-1};$                                       $\triangleright$  Angular displacement
3:   if  $\Lambda$  then
4:      $error \leftarrow \text{PROPORTION}(\Delta\Phi, reference);$ 
5:   else
6:      $error \leftarrow \text{PROPORTION}(\Delta\Phi, 0);$ 
7:   MOTORS_SPIN( $\Phi, error, \Lambda$ );
8:   SAVE_DATAS( $t, rotation, \Phi$ );
9:   SEND_DATAS( $rotation$ );

```

---

sendo:

$t$ : Instante de tempo da leitura dos sensores;

$\lambda$ : Leitura do respectivo sensor fototransistor indicando existência visual de faixa de cada lado do carro. Sendo  $\Lambda$  representando operações em cada lado (esquerdo e direito), separadamente;

$\phi$ : Leitura dos *tricks* de cada motor. Com  $\Phi$  representando operações em cada lado (esquerdo e direito), separadamente.

## 6.3 Carro Robô $\mathcal{B}$

O carro robô  $\mathcal{B}$  é equipado com os mesmos equipamentos, com exceção dos sensores fototransistores. Sua única função é conectar com a nuvem mostrando sua identificação como carro com etiqueta  $\mathcal{B}$ , receber o mapa do trajeto gerado pelo  $\mathcal{B}$  e executá-lo de forma a tentar reproduzir o trajeto feito e calculado pelo carro  $\mathcal{A}$ .

### 6.3.1 Programação

Primeiramente utilizou-se da técnica mais simples que baseia-se na única tarefa de reproduzir os valores de PWM obtidos pelo carro  $\mathcal{A}$ . Mas como essa abordagem de controle é do tipo de malha aberta, o carro ficaria totalmente desgovernado, já que os valores não representariam de fato os movimentos realizados pelo primeiro utilizando este método.

Realizado essa tentativa, utilizou-se das leitura dos *encoders* para auxiliar no controle de velocidade do carro a fim de criar um intervalo de confiança mais próximo de reprodução do carro  $\mathcal{A}$ .

## 7 Relação de Recorrência para o Modelo Implementado

Para modelar como o robô move em relação a pista, inicia-se o então representando dois item importantes para o calculo no qual são o erro em relação à pista e a velocidade de cada motor. O erro de cada pista é calculado por um conjunto de dados *error\_vel* e sua velocidade  $v$ , no qual  $error\_vel[n]$  e  $v[n]$  no  $n^{th}$  representam respectivamente o valor de erro e velocidade de cada lado do carro sendo que  $error\_vel[0] = 0$  e  $v[0] = 0$  pois o carro encontra-se parado sem qualquer tipo de leitura inicial. A diferença entre cada item pode ser relacionado com a taxa de variação de tempo em relação a cada natureza, sendo essa a continuação ou não no erro ou velocidade.

Assim, o erro no tempo  $n$  é calculado pelo modelo

$$error\_vel[n] = \alpha - v[n - 1] \quad (9)$$

no qual:

$\alpha$ : constante de velocidade referência em *tricks* no qual relaciona-se com o estado desejado do robô;

$v$ : velocidade do motor em *tricks*;

Há duas situações pois deve existir o controle pra duas situações principais: controle de velocidade e parada emergencial. A primeira situação representa a situação quando o carro necessita manter-se numa velocidade, determinada pelo  $\alpha$  situação não qual não existe a visualização da faixa. A segunda, ao encontrar a faixa, o controle deve parar a respectiva roda imediatamente a fim de que o carro não ultrapasse a faixa, se perdendo totalmente do percurso. Dessa forma, a referência altera-se para o valor 0 afim de exigir a parada do motor. O resultado da função é o valor de erro da velocidade atual para a velocidade requerida como referência.

Uma vez calculado a proporção de erro em relação à velocidade referência, utiliza-se desse para a geração dos novos valores de potência do carro lembrando que, matematicamente, a velocidade do robô é alterada de acordo com o grau e sinal do erro atual. São eles

$$power[n] = 35 \times \frac{K_p \times abs(MAX - power[n-1]) \times error}{100} \quad (10)$$

$$= 35 \times \frac{K_p \times abs(MAX - power[n-1]) \times error\_vel[n]}{100} \quad (11)$$

$$= \begin{cases} 35 \times \frac{K_p \times abs(MAX - power[n-1]) \times error\_vel[n]}{100} & \text{ctrl. proporc. de vel.} \\ 0 & \text{parada motor} \end{cases} \quad (12)$$

sendo:

$K_p$ : constante de ganho proporcional;

$MAX$ : limite máximo de potência ao motor.

Com o valor calculado por tais fórmulas, o controle, realizado em nuvem, calcula o erro a cada momento da velocidade atual sobre a referência e com tais entradas, gera o valor de PWM para o carro realizar sua atuação, repetindo todo o processo para as novas leituras de sensores.

## 8 Análise dos Componentes e Obstáculos

Com a construção do carro e teste com todos os componentes disponíveis para uso, verificou-se alguns problemas de projeto não previstos na Análise Teórica e esses serão descritos a seguir.

### 8.1 Motores de Corrente Contínua com Suas Caixas de Reduções

#### 8.1.1 Tempo de Atuação

Após realizados alguns testes com os motores, percebeu-se que eles possuem um pequeno atraso de resposta de comando. Por exemplo, quando o carro se depara com uma faixa em um lado, indicando que deveria iniciar o processo de curva, as ações de alteração de velocidade ou mesmo de parada do motor não aconteciam num intervalo de tempo necessário para que o carro consiga de fato continuar dentro de sua trajetória.

A primeira hipótese foi a latência gerada pela comunicação sem-fio na arquitetura distribuída, entretanto, realizou-se novos testes com o sistema de controle totalmente local e o problema persistia, refutando a ideia da latência. Feito isso, concluiu-se que tal atraso é uma característica física do motor e que este fato deveria ser levado em consideração à construção do controle.

#### 8.1.2 Partida dos Motores

Nas experiências com os motores, para o carro conseguir vencer a inércia inicial, deveria injetar uma potência tão alta nos motores que o carro perdia o controle de direção por causa de sua busca aceleração e este será discutido nas análises de desempenho do carro.

O problema do descontrole gerado pela aceleração é tamanho que faz com que o carro perca o controle de seus movimentos tornando o mapa gerado pela aprendizagem mais caótico que o esperado pelo projeto deixando impossível de ser reproduzido pelo carro  $\mathcal{B}$ .

### 8.1.3 Uso de dois *packs* de pilha

O problema citado anteriormente seria teoricamente resolvido ao adicionar mais corrente aos motores adicionando mais *pack* de pilhas em paralela ao primeiro. Entretanto, realizou-se testes com um *pack* adicional para fornecer maior alimentação mas não obteve melhores resultados.

Tanto com um e dois *packs* de pilha, houve momentos que o controle colocou potência máxima nos motores e mesmo assim eles não possuíam torque suficiente para movimentar o carro gerando dúvidas sobre o nível da resistência dos trens de engrenagens das caixas de reduções.

### 8.1.4 Controle de Velocidade Utilizando Encoder

Mesmo utilizando o *encoder* para realizar o controle de velocidade, só é possível detectar movimento após 0,5105 centímetros percorridos, de acordo com a Seção 4.2.2.1, o que implica que, para o cálculo de velocidade do carro com mais precisão, ele precisaria andar numa velocidade maior para que obtivesse mais *tricks* por intervalo de tempo.

Isso pois, como os intervalos de tempo para cálculo e envio à nuvem são pequenos (na casa de dezenas de milissegundos, *ms*), o cálculo da velocidade é dificultado pois taxa de *tricks* por intervalo de tempo é pequena, cerca de 1 à 2, por tempo, impedindo de um controle de velocidade mais preciso.

## 8.2 Equipamentos para a Aprendizagem de Trajeto e Reprodução

Para que o carro  $\mathcal{B}$  consiga reproduzir os passos realizados pelo carro robô  $\mathcal{A}$ , é necessário que o  $\mathcal{A}$  tenha um mecanismo mecânico/lógico para armazenamento de trajeto realizado.

Na primeira tentativa, utilizou-se do sensor de movimento 10-DOF<sup>3</sup>. O sensor disponibilizado possui 3 eixos de acelerômetro combinados com 3 eixos de giroscópio com mais 3 eixos magnetrônicos e um barômetro.

A leitura dos dados do sensor foi realizada com sucesso, mas após o manuseio dos dados, percebeu-se que não é possível ter a localização do carro no tempo  $t$  utilizando tais dados já que o trajeto entre o  $t - 1$  e  $t$  pode acontecer inúmeras situações como desorientações bruscas ou até mesmo simples zig-zagues que impedem do robô  $\mathcal{B}$  de reproduzir o trajeto levando em conta o problema do motor mencionado na Seção 8.1. Tais sensores informam a posição do ambiente em relação ao carro, o que o inverso não é válido. Isso pois é possível conseguir a velocidade e angulação no tempo  $t - 1$ , mas não é possível extrair a relação desta com a posição  $t$ .

---

<sup>3</sup>Degrees Of Freedom.

O problema seria resolvido ao utilizar um GPS no qual seria possível obter a posição do carro em relação ao ambiente e vice-versa. Entretanto, o GPS deveria ter precisão de centímetros o que o torna um equipamento caro e inviável para o projeto.

Como tentativa de contornar o problema, utilizou-se de sensores que detectam o movimento das rodas de forma incremental, como descrito na Seção 4.2.2.1. Dessa forma, é possível identificar quando a roda é movimentada verificando quantos *tricks* foram contabilizados por meio de interrupções geradas pelo sensor ao microcontrolador e utilizar isso no controle proporcional do carro  $\mathcal{B}$  tornando-o de malha fechada para a tentativa de reprodução de trajeto com base em velocidade.

## 9 Análise Analítica

De forma a adentrar na causa que impediu a conclusão do trabalho com sucesso, realizou-se testes analíticos a fim de demonstrar a causa dos problemas citados.

Para o funcionamento do carro de forma correta, seria necessário o controle proporcional ter gerência de vários intervalos de velocidade, desde a velocidade máxima até a menor, incluindo quando o motor estiver parado. Ao possuir um intervalo grande de velocidade, permite-se que o carro fizesse curvas com melhor controle evitando as oscilações grandes. Eliminando a situação de quando o motor está parado, para validação, analisou-se das duas ocasiões mais importantes enfrentadas pelo carro, sendo essas quando ele está em alta e baixa rotação. Esses dados são importantes pelo fato que ambas (alta e baixa rotação) são velocidade extremas do intervalo usado pelo controle e que elas possuem outras variáveis que influenciam na sua atuação como a intensidade da corrente de partida e a resistência dos trens de engrenagem situados em cada motor, por exemplo.

Dessa forma, como o NodeMCU gera um intervalo de 0 à 1023 para valores de PWM para os motores, utilizou-se dos valores 1023 e 600 para alta e baixa rotação respectivamente, visando a análise de atuação de cada um dos valores. O carro que comporte em ambas as situações permite-se concluir que ele possui todos os requisitos físicos necessários para que o controle consiga operar com mais facilidade obtendo melhor precisão em suas tarefas propostas. Assim, o carro que não realizar bem tais movimentos, poderá ter problemas futuros como estabilidade e precisão, por exemplo.

Outro item a mencionar sobre o teste são os itens mensurados. Como em algumas situações o acionamento não necessariamente implica no giro da roda, comparou-se a intensidade de corrente do motor em modo de partida, nominal e em movimento. Descrevendo melhor:

**Partida:** Intensidade de corrente que o motor necessita para gerenciar a partida, ou seja, a tentativa de movimento;

**Nominal:** Intensidade de corrente após a tentativa de partida inicial. Valores normativos do motor ao longo do tempo seguindo um mesmo valor de entrada de PWM;

**Movimento:** Intensidade do motor quando este está realmente realizando o movimento giratório, ou seja, quando realmente conseguiu sair da inércia inicial

que é estar sem movimento.

Executou-se dez testes mensurando os valores de intensidade de corrente em todos as situações Partida (Tabela 1), Nominal (Tabela 2) e Movimento (Tabela 3).

Tabela 1: Tabela de valores médios de intensidade de corrente de partida aos principais movimentos requeridos pelo carro robô.

	$\bar{X}$ Corrente de Partida	$\sigma$	% Erro
<b>Alta Rotação</b>	313, 7mA	26, 36938	8, 338732
<b>Baixa Rotação</b>	283, 2mA	19, 65706	6, 216108

Tabela 2: Tabela de valores médios de intensidade de corrente nominal aos principais movimentos requeridos pelo carro robô.

	$\bar{X}$ Corrente Nominal	$\sigma$	% Erro
<b>Alta Rotação</b>	138, 9mA	14, 22400	4, 498024
<b>Baixa Rotação</b>	215, 7mA	11, 75727	3, 717974

Tabela 3: Tabela de valores médios de intensidade de corrente com o motor rotacionando aos principais movimentos requeridos pelo carro robô.

	$\bar{X}$ Corrente c/ Motor em Rotação	$\sigma$	% Erro
<b>Alta Rotação</b>	142, 0mA	5, 656854	1, 788854
<b>Baixa Rotação</b>	130, 8mA	18, 68927	5, 910067

Sendo o cálculo do erro seguindo a fórmula  $\frac{\sigma}{\sqrt{|x|}}$  no qual  $x$  sendo os total de valores coletados.

## 9.1 Análise da Alta Rotação

Olhando primeiramente a alta rotação, o carro possui força suficiente para iniciar seu movimento com a corrente de partida (313, 7mA) e, depois de vencer a inércia, a corrente estabiliza em valores próximos à 138, 9mA fazendo o motor girar.

Todos os dez testes realizados com esta potência o carro teve sucesso suficiente para vencer sua inércia inicial. Isso deve-se ao fato da aceleração do motor ser muito alta, sobrepondo qualquer resistência física que impede o movimento da roda.

## 9.2 Análise da Baixa Rotação

Diferentemente do teste de alta rotação, o teste com baixa velocidade não obteve tanto sucesso. Nos testes de baixa rotação, o valor de intensidade de corrente de partida teve uma média de 283, 2mA, entretanto, em todos os testes esse valor de potência não foi suficiente para iniciar a rotação no motor, ou seja, os testes realizados com o valor 600 de PWM fracassaram ao girar a roda, mesmo o motor demonstrando aplicar a força ao emitindo barulho de acionamento.

Verificando a dificuldade do início da rotação nos testes de baixa velocidade, realizou-se uma pequena ajuda manual à roda para o giro inicial e percebeu que, para este giro inicial, o motor aplica uma força suficiente para a movimentação do carro mas insuficiente para vencer a resistência dos trens de engrenagens da caixa de redução.

## 10 Discussão

Realizado todos os testes, o principal problema encontrado no projeto no qual influencia na execução e completude é a resistência dos trens de engrenagens existente na caixa de redução de cada motor. Esse problema físico foi facilmente perceptível ao visualizar a situação na qual o motor de fato injetava potência para iniciar o giro e não conseguia. A resistência das engrenagens é tamanha que, após vencido e iniciado o giro da roda, a potência é forte o suficiente para conseguir mover o carro. Ou seja, a potência é suficiente para arrastar o carro, mas não para dar sua partida.

Esse problema poderia sim ser resolvido pelo controle, acionando mais potência no início e depois equilibrando no início de seu movimento. Entretanto, injetar mais potência na partida faz com que sua aceleração seja alta de tal forma que, após iniciada, o controle não consiga corrigí-la. Isso é causado pelo item mencionado em Seção 8.1 no qual é descrito um certo atraso na atuação do motor, podendo causar descontrole de seus movimentos e consequentemente a perdendo o caminho a seguir. O problema das engrenagens é tamanha que, houve situações de execução em que o carro injetou potência máxima nos motores (valor 1023 de PWM) e eles não conseguiram vencer a resistência das engrenagens. O problema da resistência facilmente resolvido ao adicionar mais pares de motores ao carro, entretanto, o chassi do carro não possui suporte para a instalação de mais motores com seus *encoders*.

Explicando melhor o descontrole do carro, pelo controle, na sua aceleração, pois primeiro, o uso de um controle proporcional simples, sem uso de ferramentas Integrais ou Derivativas para maior controle da situação o tornam um controle mais precário. Segundo, pela situação na qual o carro deve percorre  $9^\circ$  (ou seja, 0,5105 centímetros) para que o controle note o início do movimento. Esse valor teoricamente pode parecer insignificante, mas essa distância percorrida somada com a distância sobre de processamento do carro (30 milissegundos), do processamento da nuvem e sua a latência de comunicação de envio e do recebimento de dados até a atuação do controle faz com que o carro percorra cerca de 1,742 centímetros até que a atuação seja totalmente completada. Esse valor de distância é suficiente para fazer com que o carro extrapole o tamanho da faixa e saia da pista, mesmo tenho realizado o procedimento de alteração de movimento.

Além deste problema de controle, as acelerações e comportamentos entre cada carro são realizadas de forma bastante diferentes. Uma simples aceleração mais brusca ou lenta faz com que tal erro seja exponencial a qualquer movimento futuro que o carro fará. Com ele não possui um item de controle de posição, o carro fica a mercê da precisão exercida pelo controle à velocidade angular e principalmente sobre sua aceleração no qual torna-se o maior problema do carro neste projeto.

Dessa forma, sabendo de tais problemas físicos do carro, realizou-se uma análise das situações mais importantes que o robô enfrenta com base em movimento teóricos perfeitos que fariam o carro realizar a tarefa proposta com sucesso. Na teoria, a baixa

rotação deveria ser acionada e estabilizada com valores de intensidade de corrente nominal  $I_{nominal} = 130,8$ . Entretanto, a corrente de partida  $I_{partida} = 283,2$  não é suficiente para iniciar a movimentação do carro. Dos testes realizados com propósito do motor conseguir dar a partida, necessitou-se de intensidades de correntes maiores que  $I_{giro} = 311,1$  para obter seu início. Isso da uma diferença de corrente de  $I_{diferenca\_partida} = 27,9mA$ , ou seja 10,1505% de potência a mais para as rotações.

## 11 Conclusão

Como demostrativo do resultado final do projeto, segue os *links* disponibilizou-se três vídeos para exemplificar o funcionamento do carro. O primeiro vídeo <https://youtu.be/RWjgYUrU1Fo> exibe o carro  $\mathcal{A}$  realizando seu dever de completar o trajeto. O mesmo vídeo pode ser visto em câmera lenta, gravado em  $240fps$  percebendo cada detalhe do carro e seu procedimento de controle, disponível em <https://youtu.be/xk-tbqkAV0s>. Também foi disponibilizado um vídeo demonstrando o tempo de demora na atuação dos motores após a leitura da faixa no qual pode ser verificado em câmera lenta, disponibilizado em <https://youtu.be/pGvHHe4h54I>. Por fim, outro vídeo (com música de fundo) demonstrando outra atuação do carro, disponível em <https://youtu.be/SD5c9YnoTfo>. Não há demonstração do carro  $\mathcal{A}$  pois todos os testes realizados com tal foram falhos.

O primeiro item físico notado na construção e uso do carro que dificultaria a conclusão do projeto é a latência de resposta do motor em virtude ao intervalo de resposta requerido ao controle para manter o carro na pista. Uma vez que o motor possui alta latência de resposta, o carro fica totalmente à mercê da atuação final do motor.

As caixas de reduções são componente que mais causou empecilho ao trabalho. Sua resistência é tamanha que o carro, mesmo com torque suficiente para mover o carro em baixas rotações, não consegue vencer à resistência dos trilhos de engrenagens quando o carro encontra-se estacionado. Assim, para garantir movimentos mais suaves, será viável a utilização de um sistema de engrenagens que não obtivesse tanta resistência.

Obviamente, utilizar componentes mais leves faz com que o carro tenha mais facilidade de movimentação dispondendo menos resistência aos motores. Quanto menor a resistência enfrentadas pelos motores, mais será a eficiência energética e seu fator de controle.

Propor um trabalho fornecendo um chassi que tenha suporte à mais motores e seus *encoders* faz com que o trabalho possa ser desmembrado em mais opções de construções. Alterar a estrutura física do carro, adicionando mais motores por exemplo, faz com que o desenvolvimento do carro seja abordado de mais formas, escolhendo a que mais atrela ao ambiente testado e equipamentos disponibilizados.

Utilizar um *encoder* que foneça uma percepção maior de movimento faz com que o controle gerencie a velocidade do carro com mais precisão. O modelo atual utilizado usa apenas 3 valores de velocidade pois o tempo de captura de taxa de variação é muito rápido e o intervalo de *tricks* é a cada  $9^\circ$  criando curta amostra de velocidade. Os três valores de velocidade representam os estados onde o carro está com o motor parado (ou seja, zero *tricks*), e os outros valores de velocidade 1 e 2 que representam respectivamente a taxa de *tricks* ao tempo, projetado para ser de

30 milissegundos.

Discutindo um pouco sobre algumas dificuldades encontradas no projeto e desenvolvimento do carro, todos os dois *motor shield* fornecidos para o trabalho possuíam problema de circuito físico no qual, o componente *not* que permitia o motor girar em sentido contrário estavam danificados. Não se sabe o motivo da danificação. Gastou-se muito tempo descobrindo a falta deste circuito integrado e por isso o projeto foi fundado sobre o *motor shield* incapacitado de girar os motores em sentido reverso.

O modelamento matemático do projeto a ser implementado deve ser descrito com conhecimentos prévios dos componentes tal como seus problemas físicos do trabalho para incrementar mais variáveis à formula a fim de aprimorar mais os resultados.

A geração de um mapa de trajeto para que o carro  $\mathcal{B}$  realize também foi tido como uma tarefa problemática ao projeto. Realizou-se várias tentativas de geração e aplicação de mapas utilizando várias combinações de sensores mas todas foram fracassadas ao ponto de tentar gerar pontos de localização a fim de que o segundo, carro  $\mathcal{B}$ , possa utilizar tal estratégia para reprodução.

O controle proporcional desenvolvido neste trabalho é capaz de realizar a tarefa do carro  $\mathcal{A}$  de completar o circuito do carro em seu trajeto, mas não é otimizado o suficiente para extrair máximo proveito dos recursos do carro. Sendo assim, é possível realizar um novo estudo aprofundado em teoria de controle com os conhecimentos adquiridos neste trabalho sobre detalhes físicos com o propósito de melhorar o carro  $\mathcal{A}$  de forma a realizar suas tarefas com maior precisão.

Como utilizou-se uma variante do controle proporcional (PD) simples, o trabalho de pesquisa e projeto poderá ser continuado utilizando controles proporcionais mais estáveis usando por exemplo técnicas mais elaboradas como a mescla de controles proporcionais com integrais e derivativos.

Neste trabalho foi possível perceber claramente a aprendizagem de controles proporcionais de primeira ordem. Este exemplo simples de *follow line* do trabalho mostrou-se claro ao comparar as vantagens da utilização de tais controles utilizados corretamente para obter melhores resultados na sua tarefa. Qualquer tipo de sistema que utilize malha aberta poderia ser aprimorado a utilizar malha fechada a ponto de obter resultados mais controlados e seguros em duas atuações, tendo em mente que para isso, sensores e atuadores são necessários para que haja o processamento do controle na malha. Outro tópico aprendido na realização deste trabalho é o aperfeiçoamento das técnicas básicas de confecção, manuseio e aferição de circuitos eletrônicos simples. Neste trabalho, foi necessário conhecimentos básicos sobre corrente elétrica, teoria de circuitos eletrônicos e operação com algumas ferramentas do multímetro.

## 12 Códigos

### 12.1 Código em Linguagem Arduino

#### 12.1.1 ploudy Node MCU

---

```
1 #include <SoftwareSerial.h>
2 #include <stdio.h>
3 #include <ESP8266WiFi.h>
4 #include <Wire.h>
5 #include "MPU9250.h"
6
7
8
9 // ===== Constants =====
10 #define RADIUS      3.4
11 #define LARGURE     13.42
12 #define NUMBER_FRAMES 40
13 #define DELTA_TEMPO 75
14 #define SERIALDEBUG true
15 #define SSID          "Imobilis"
16 #define SSID_PASSWORD "bolinha de sabao"
17 #define URL          "192.168.0.174"
18 #define PORT         8888
19
20 // ===== Instantiations =====
21 WiFiClient client;
22 MPU9250 accelgyro;
23 I2Cdev I2C_M;
24 // ===== Variables for MPU9250 =====
25 #define sample_num_mdate 5000
26 volatile float mx_sample[3], my_sample[3], mz_sample[3];
27 static float mx_centre = 0, my_centre = 0, mz_centre = 0;
28 static float mx_max = 0, my_max = 0, mz_max = 0;
29 static float mx_min = 0, my_min = 0, mz_min = 0;
30 uint8_t buffer_m[6];
31 int16_t mx, my, mz;
32 float heading;
33 float Mxyz[3];
34 // ===== Human Interface and Controls =====
35 char command      = '0';
36 bool restart_program = true;
37 // ===== Pins =====
38 long pin_left_motor_pwm      = D1; // Pin to left motor pwm
39 long pin_righ_motor_pwm      = D2; // Pin to righ motor pwm
40 long pin_left_motor_pwm_direction = D3; // Pin to left motor direction
41 long pin_righ_motor_pwm_direction = D4; // Pin to righ motor direction
42 long pin_left_optical_sensor = D5; //Pin to optical sensor
43 long pin_righ_optical_sensor = D6; //Pin to optical sensor
44 long left_encoder           = D7; // Pin to encoder
45 long righ_encoder           = D8; // Pin to encoder
46 // ===== Timing =====
47 unsigned long tempo          = 0;
48 unsigned long global_start_time = 0;
49 unsigned long global_spended_time = 0;
50 // ===== Send Datas =====
51 byte left_light_boolean = 1, righ_light_boolean = 1;
52 byte left_light_boolean_inter = 1, righ_light_boolean_inter = 1;
53 // ===== Receive Datas =====
```

```

54     long size_map      = 0;
55     char s_package[256] ;
56     bool package_sent  = false;
57     String str_send    = "";
58
59 //TODO: Rever essas variáveis se estão postas de forma certa
60     int left_master_pwm = 0;
61     int righ_master_pwm = 0;
62     int left_master_frames = 0;
63     int righ_master_frames = 0;
64
65 // ===== Encoder Angular Velocity =====
66     long left_current_encoder_this_car = 0;
67     long righ_current_encoder_this_car = 0;
68     long left_last_encoder_this_car   = 0;
69     long righ_last_encoder_this_car   = 0;
70     long x_point                    = 0;
71     long y_point                    = 0;
72     int reference                  = 0;
73
74
75
76
77     float calculating_heading(void) {
78         heading = 180 * atan2(Mxyz[1], Mxyz[0]) / PI;
79         if (heading < 0) heading += 360;
80
81         return heading;
82     }
83
84     void getTiltHeading(void) {
85         float pitch = asin(-Axyz[0]);
86         float roll = asin(Axyz[1] / cos(pitch));
87
88         float xh = Mxyz[0] * cos(pitch) + Mxyz[2] * sin(pitch);
89         float yh = Mxyz[0] * sin(roll) * sin(pitch) + Mxyz[1] * cos(roll) - Mxyz[2] * sin(roll) *
90             cos(pitch);
91         float zh = -Mxyz[0] * cos(roll) * sin(pitch) + Mxyz[1] * sin(roll) + Mxyz[2] * cos(roll) *
92             cos(pitch);
93         tiltheading = 180 * atan2(yh, xh) / PI;
94         if (yh < 0)     tiltheading += 360;
95     }
96
97     void Mxyz_init_calibrated () {
98         Serial.print("\n[INFO] Sample starting.....");
99         Serial.print("\n[INFO] waiting .....");
100
101         get_calibration_Data ();
102
103         Serial.print("[INFO] Compass calibration parameter: ");
104         Serial.print(mx_centre); Serial.print(", ");
105         Serial.print(my_centre); Serial.print(", ");
106         Serial.print(mz_centre);
107
108
109     void get_calibration_Data () {
110
111         get_one_sample_date_mxyz();

```

```

112     mx_max = mx_sample[2];
113     my_max = mx_sample[2];
114     mz_max = mx_sample[2];
115     mx_min = mx_sample[2];
116     my_min = mx_sample[2];
117     mz_min = mx_sample[2];
118
119     for (int i = 0; i < sample_num_mdate; i++) {
120         get_one_sample_date_mxxyz();
121
122         Serial.print("\n");
123         Serial.print(i); Serial.print(":");
124         Serial.print(sample_num_mdate); Serial.print("   ");
125         Serial.print(mx_sample[2]); Serial.print(" ");
126         Serial.print(my_sample[2]); Serial.print(" ");
127         Serial.print(mz_sample[2]);
128
129         if (mx_sample[2] >= mx_max) mx_max = mx_sample[2];
130         if (my_sample[2] >= my_max) my_max = my_sample[2];
131         if (mz_sample[2] >= mz_max) mz_max = mz_sample[2];
132         if (mx_sample[2] <= mx_min) mx_min = mx_sample[2];
133         if (my_sample[2] <= my_min) my_min = my_sample[2];
134         if (mz_sample[2] <= mz_min) mz_min = mz_sample[2];
135
136     }
137     mx_centre = (mx_max + mx_min) / 2;
138     my_centre = (my_max + my_min) / 2;
139     mz_centre = (mz_max + mz_min) / 2;
140 }
141
142 void get_one_sample_date_mxxyz() {
143     getCompass_Data();
144     mx_sample[2] = Mxyz[0];
145     my_sample[2] = Mxyz[1];
146     mz_sample[2] = Mxyz[2];
147 }
148
149 void getCompass_Data(void) {
150     I2C_M.writeByte(MPU9150_RA_MAG_ADDRESS, 0x0A, 0x01); //enable the magnetometer
151     delay(10);
152     I2C_M.readBytes(MPU9150_RA_MAG_ADDRESS, MPU9150_RA_MAG_XOUT_L, 6, buffer_m);
153
154     mx = ((int16_t)(buffer_m[1]) << 8) | buffer_m[0] ;
155     my = ((int16_t)(buffer_m[3]) << 8) | buffer_m[2] ;
156     mz = ((int16_t)(buffer_m[5]) << 8) | buffer_m[4] ;
157
158     Mxyz[0] = (double) mx * 1200 / 4096;
159     Mxyz[1] = (double) my * 1200 / 4096;
160     Mxyz[2] = (double) mz * 1200 / 4096;
161 }
162
163 void getCompassDate_calibrated () {
164     getCompass_Data();
165     Mxyz[0] = Mxyz[0] - mx_centre;
166     Mxyz[1] = Mxyz[1] - my_centre;
167     Mxyz[2] = Mxyz[2] - mz_centre;
168 }
169
170
171 float get_heading() {

```

```

172     getCompassDate_calibrated();
173     return calculating_heading();
174 }
175
176
177 //      10-DOF
178 //
179 //      =====
180 //      PROCEDIMENTOS ENCODER
181
182
183 int difference_trick_encoder_left() {
184     int diff = left_current_encoder_this_car - left_last_encoder_this_car;
185
186     if (diff > 8)
187         diff = 0;
188
189     left_last_encoder_this_car = left_current_encoder_this_car;
190     return diff;
191 }
192 int difference_trick_encoder_righ() {
193     int diff = righ_current_encoder_this_car - righ_last_encoder_this_car;
194
195     if (diff > 8)
196         diff = 0;
197
198     righ_last_encoder_this_car = righ_current_encoder_this_car;
199     return diff;
200 }
201
202 float left_distance_centimeters() { return 2 * PI * RADIUS * (difference_trick_encoder_left()) /
203     NUMBER_FRAMES ; }
204 float righ_distance_centimeters() { return 2 * PI * RADIUS * (difference_trick_encoder_righ()) /
205     NUMBER_FRAMES ; }
206
207 float center_distance_centimeters() { return (righ_distance_centimeters() +
208     left_distance_centimeters()) / 2.0; }
209
210 float refresh_x_point() { x_point = x_point + center_distance_centimeters() * cos(reference);
211     return x_point; }
212 float refresh_y_point() { y_point = y_point + center_distance_centimeters() * sin(reference);
213     return y_point; }
214 float refresh_reference() { reference = righ_distance_centimeters() - left_distance_centimeters() /
215     LARGURE;     return reference; }
216
217 //      DEFINIÇÕES DE VARIÁVEIS
218 //
219 //      =====
220 //      CONEXÕES DE REDE
221
222 unsigned long millis_now() { return millis() - global_start_time; }
223 void left_add_encoder() { left_current_encoder_this_car++; }

```

```

223 void righ_add_encoder() { righ_current_encoder_this_car++; }
224
225 void left_interrupt_ligth() { left_light_boolean_inter = 1; }
226 void righ_interrupt_ligth() { righ_light_boolean_inter = 1; }
227
228 // Função para iniciar a Conexão com a rede WiFi
229 void start_connection_wifi() {
230     bool twice = false;
231     do {
232         if (SERIALDEBUG && ! twice) {
233             Serial.print("\n[INFO] Connecting in: ");
234             Serial.print(SSID);
235         }
236
237         WiFi.begin(SSID, SSID_PASSWORD); // conecta na Rede Wireless
238
239         while (WiFi.status() != WL_CONNECTED) {
240             if (SERIALDEBUG && ! twice)
241                 Serial.print("\n\tNão Conectado na Rede WiFi");
242             delay(200);
243             twice = true;
244         }
245
246         if (SERIALDEBUG) {
247             Serial.print("\n[INFO] Conectado na Rede: \n\t"); Serial.print(SSID);
248             Serial.print(" | IP ");
249             Serial.print(WiFi.localIP());
250         }
251
252     } while (WiFi.status() != WL_CONNECTED);
253 }
254
255 void master_connection_cloud() {
256     bool twice = false;
257     do {
258         if (SERIALDEBUG && ! twice) {
259             Serial.print("\n[INFO] Start Cloud Connecting");
260             Serial.print("\n\tAddress: ");
261             Serial.print(URL);
262             Serial.print(" : ");
263             Serial.println(PORT);
264
265             Serial.print("\n[INFO] Getting Connection with the cloud");
266         }
267
268         // configure traged server and url
269         if(client.connect(URL, PORT)) {
270             twice = false;
271             if (SERIALDEBUG && ! twice) {
272                 Serial.print("\n[INFO] Connection with the cloud stablished");
273                 Serial.print("\n[INFO] Sending the robot information Hi");
274             }
275
276             client.print("Hi From Master Robot\r\n");
277
278         } else {
279             if (SERIALDEBUG && ! twice)
280                 Serial.print("\n[ERRO] Failed!");
281             twice = true;
282         }

```

```

283     if (SERIALDEBUG && ! twice)
284         Serial.print("\n[INFO] Finnishing Cloud Connecting");
285
286     } while (! client.connected());
287 }
288
289
290 void verify_lost_connection_wifi() {
291     //if (SERIALDEBUG)
292     //  Serial.print("\n[INFO] Verifying if there is connection.");
293
294     if (WiFi.status() != WL_CONNECTED || ! client.connected()) {
295         Serial.print("\n[ERRO] Desconnected! Restarting Everything.");
296         motor_action(0, 0);
297
298         ESP.restart();
299     }
300 }
301
302 void slave_connection_cloud() {
303     bool twice = false;
304     do {
305         if (SERIALDEBUG && ! twice) {
306             Serial.print("\n[INFO] Start Cloud Connecting");
307             Serial.print("\n\tAddress: ");
308             Serial.print(URL);
309             Serial.print(" : ");
310             Serial.println(PORT);
311
312             Serial.print("\n[INFO] Getting Connection with the cloud");
313         }
314
315         // configure traged server and url
316         if(client.connect(URL, PORT)) {
317             twice = false;
318             if (SERIALDEBUG && ! twice) {
319                 Serial.print("\n[INFO] Connection with the cloud stablished");
320                 Serial.print("\n[INFO] Sending the robot information");
321             }
322
323             client.print("Hi From Slave Robot\r\n");
324
325         } else {
326             if (SERIALDEBUG && ! twice)
327                 Serial.print("\n[ERRO] Failed!");
328             twice = true;
329         }
330
331         if (SERIALDEBUG && ! twice)
332             Serial.print("\n[INFO] Finnishing Cloud Connecting");
333
334     } while (! client.connected());
335 }
336
337
338 //      CONEXÕES DE REDE
339 //
340 //      =====
341 //
342 //      =====

```

```

341 //      PACOTES DE ENVIO
342
343
344 void set_package_values(int &left_pwm, int &righ_pwm){
345     left_light_boolean = (int) digitalRead(pin_left_optical_sensor) || left_light_boolean_inter;
346     righ_light_boolean = (int) digitalRead(pin_righ_optical_sensor) || righ_light_boolean_inter;
347
348     left_light_boolean_inter = 0;
349     righ_light_boolean_inter = 0;
350
351 }
352
353 void create_string_values() {
354     s_package[0] = '\0';
355
356     str_send = "";
357     str_send = str_send + " " + millis_now() + " " + left_light_boolean + " " +
358     righ_light_boolean + " " + left_current_encoder_this_car + " " +
359     ↵ righ_current_encoder_this_car; // + " " + get_heading();
360 }
361
362 void prepare_and_send_new_datas(int &left_pwm, int &righ_pwm) {
363     set_package_values(left_pwm, righ_pwm);
364     create_string_values();
365
366     if (client.connected())
367         client.print(str_send);
368     else
369         ESP.restart();
370 }
371
372
373 //      PACOTES DE ENVIO
374 //
375 //      =====
376 //      PACOTES DE RECEBIMENTO
377
378
379
380 void get_new_values_master(int &left_pwm, int &righ_pwm) {
381     char * left_motor_pwm_token;
382     char * righ_motor_pwm_token;
383     const char search[2] = " ";
384
385     left_motor_pwm_token = strtok(s_package, search);
386     righ_motor_pwm_token = strtok(NULL, search);
387
388     left_pwm = atoi(left_motor_pwm_token);
389     righ_pwm = atoi(righ_motor_pwm_token);
390 }
391
392 void get_new_values_slave() {
393     char * tempo_token;
394     char * left_master_pwm_token;
395     char * righ_master_pwm_token;
396     char * left_master_frames_token;
397     char * righ_master_frames_token;

```

```

398 const char search[2] = " ";
399
400 tempo_token           = strtok(s_package, search);
401 left_master_pwm_token = strtok(NULL, search);
402 righ_master_pwm_token = strtok(NULL, search);
403 left_master_frames_token = strtok(NULL, search);
404 righ_master_frames_token = strtok(NULL, search);
405
406 tempo           = atoi(tempo_token);
407 left_master_pwm = atoi(left_master_pwm_token);
408 righ_master_pwm = atoi(righ_master_pwm_token);
409 left_master_frames = atoi(left_master_frames_token);
410 righ_master_frames = atoi(righ_master_frames_token);
411 }
412
413 void motor_action(long left_velocity, long righ_velocity) {
414     long max = 1023;
415
416     Serial.printf("\nL:%d %d\n", left_velocity, righ_velocity);
417
418     if (left_velocity > 0) {
419         digitalWrite(pin_left_motor_pwm_direction, HIGH);
420         if (left_velocity > max) {
421             left_velocity = max;
422         }
423
424     } else if (left_velocity < 0) {
425         digitalWrite(pin_left_motor_pwm_direction, LOW);
426         if (left_velocity < -max)
427             left_velocity = max;
428         else
429             left_velocity = 0 - left_velocity;
430     }
431
432     if (righ_velocity > 0) {
433         digitalWrite(pin_righ_motor_pwm_direction, HIGH);
434         if (righ_velocity > max) {
435             righ_velocity = max;
436         }
437
438     } else if (righ_velocity < 0) {
439         digitalWrite(pin_righ_motor_pwm_direction, LOW);
440         if (righ_velocity < -max)
441             righ_velocity = max;
442         else
443             righ_velocity = 0 - righ_velocity;
444     }
445
446 // Drive motors according to the calculated values for a turn
447 analogWrite(pin_left_motor_pwm, left_velocity);
448 analogWrite(pin_righ_motor_pwm, righ_velocity);
449 }
450
451
452
453 //          PACOTES DE RECEBIMENTO
454 //
455 //          =====

```

```

456  // 
457
458 void setup() {
459
460     // Start the i2c communication
461     //Wire.begin(D3, D4);
462
463     Serial.begin(115200);
464
465     if (SERIALDEBUG)
466         Serial.print("\n-----\n\n[INFO] Defining Pins");
467
468     pinMode(D0, INPUT);
469
470     // Set motors
471     pinMode(pin_left_motor_pwm, OUTPUT);
472     pinMode(pin_righ_motor_pwm, OUTPUT);
473     pinMode(pin_left_motor_pwm_direction, OUTPUT);
474     pinMode(pin_righ_motor_pwm_direction, OUTPUT);
475
476     digitalWrite(pin_righ_motor_pwm, LOW);
477     digitalWrite(pin_left_motor_pwm, LOW);
478     digitalWrite(pin_righ_motor_pwm_direction, HIGH);
479     digitalWrite(pin_left_motor_pwm_direction, HIGH);
480
481     restart_program = true;
482
483     pinMode(left_encoder, INPUT);
484     pinMode(righ_encoder, INPUT);
485 }
486
487
488 void restart_program_process() {
489     restart_program = false;
490
491     if (WiFi.status() != WL_CONNECTED)
492         start_connection_wifi();
493
494     if (digitalRead(D0) == HIGH) {
495         if (SERIALDEBUG)
496             Serial.print("\n[INFO] MASTER");
497
498         if (! client.connected())
499             master_connection_cloud();
500
501         // Light Sensor
502         pinMode(pin_left_optical_sensor, INPUT);
503         pinMode(pin_righ_optical_sensor, INPUT);
504
505         if (SERIALDEBUG)
506             Serial.print("\n[INFO] Waiting the Starting...");
```

```

516     if (command != 's') {
517         Serial.print("\n[INFO] Error on Start! Restarting.");
518         restart_program = true;
519
520         restart_program = true;
521     }
522
523     command = '0';
524
525     if (SERIALDEBUG)
526         Serial.print("\n[INFO] Starting!");
527
528     global_start_time = millis() - 5000;
529 }
530
531 else {
532     if (SERIALDEBUG)
533         Serial.print("\n[INFO] SLAVE");
534
535     if (! client.connected())
536         slave_connection_cloud();
537
538     if (restart_program == true)
539         ESP.restart();
540
541
542     if (SERIALDEBUG)
543         Serial.printf("\n[INFO] Waiting for Start");
544
545     while(! client.available() && client.connected()) ;
546     verify_lost_connection_wifi();
547
548     command = '0';
549
550     if (client.available())
551         command = client.read();
552
553
554     if (SERIALDEBUG)
555         Serial.printf("\nCommand Read!");
556
557     if (command != 's') {
558         Serial.print("\n[INFO] Error on Start! Restarting.");
559         restart_program = true;
560
561         ESP.restart();
562     }
563
564     command = '0';
565
566     global_start_time = millis();
567
568     while(millis_now() < 5000) {
569         delay(20);
570         if (SERIALDEBUG)
571             Serial.printf("\n%lu>%lu", millis_now(), 5000);
572     }
573
574     if (SERIALDEBUG) {
575         //Serial.printf("\n0:%d %lu %d %d", size_map, 5000,

```

```

576         //left_motor_pwm, righ_motor_pwm);
577         Serial.printf("\nStarting!");
578     }
579 }
580
581 detachInterrupt(D7);
582 detachInterrupt(D8);
583
584 left_current_encoder_this_car = righ_current_encoder_this_car = 0;
585 left_last_encoder_this_car = righ_last_encoder_this_car = 0;
586 attachInterrupt(D7, left_add_encoder, CHANGE);
587 attachInterrupt(D8, righ_add_encoder, CHANGE);
588 }
589
590
591 void loop() {
592     int left_pwm = 0, righ_pwm = 0;
593     long i, index = 0;
594     long spended_time_waiting;
595     String s_buffer;
596
597     if (restart_program)
598         restart_program_process();
599
600     if (SERIALDEBUG)
601         Serial.printf("\nIntro the loop");
602
603     if (digitalRead(D0) == HIGH) {
604         while (true) {
605             verify_lost_connection_wifi();
606
607             if (restart_program == true)
608                 ESP.restart();
609
610             package_sent = false;
611
612             do {
613                 prepare_and_send_new_datas(left_pwm, righ_pwm);
614
615                 while(! client.available() and restart_program == false)
616                     verify_lost_connection_wifi();
617
618                 if (restart_program == true)
619                     ESP.restart();
620
621                 if (client.available())
622                     package_sent = true;
623             } while (! package_sent);
624
625             if (restart_program == true)
626                 ESP.restart();
627
628             i = 0;
629             while (client.available())
630                 s_package[i++] = client.read();
631
632             s_package[i] = '\0';
633             get_new_values_master(left_pwm, righ_pwm);
634             motor_action(left_pwm, righ_pwm);
635

```

```

636     while (millis_now() - spended_time_waiting < DELTA_TEMPO) ;
637     spended_time_waiting = millis_now();
638 }
639 }
640
641 else {
642
643     index = 0;
644     while (! client.available() and restart_program == false)
645         verify_lost_connection_wifi();
646
647     if (restart_program == true)
648         ESP.restart();
649
650     s_buffer = client.readStringUntil('\n');
651     size_map = s_buffer.toInt();
652
653     i = 0;
654     while (i < size_map) {
655
656         while (! client.available() and restart_program == false)
657             verify_lost_connection_wifi();
658
659         if (restart_program == true)
660             ESP.restart();
661
662         s_buffer = client.readStringUntil('\0');
663         client.print("Next\n");
664         s_buffer.toCharArray(s_package, 127);
665         s_package[125] = '\n';
666         s_package[126] = '\0';
667
668         get_new_values_slave();
669
670         left_slave_error = calcule_proportion(left_current_encoder_this_car, left_master_frames);
671         righ_slave_error = calcule_proportion(righ_current_encoder_this_car, righ_master_frames);
672
673         calcule_spin(left_slave_error, righ_slave_error);
674
675         while (millis_now() < tempo) ;
676
677         motor_action(left_pwm, righ_pwm);
678
679         i++;
680     }
681     motor_action(0, 0);
682     if (SERIALDEBUG)
683         Serial.printf("\nDONE!");
684
685     delay(200);
686
687     ESP.restart();
688 }
689 }

```

---

## 12.2 Código em Linguagem Python

### 12.2.1 ploudy Server

---

```
1  from __future__ import print_function
2  import socket
3  import sys
4  import time
5  import function
6  import pickle
7
8  HOST = "" # Symbolic name, meaning all available interfaces
9  PORT = 8888 # Arbitrary non-privileged port
10
11 ROBOT_SENSORS = "Master Robot"
12 ROBOT_WITHOUT_SENSORS = "Slave Robot"
13
14 main_command = "0"
15 import threading
16
17 command = "0"
18
19 class Command_thread ( threading.Thread ):
20
21     def run ( self ):
22         print("Command Thread Started")
23
24         global command
25
26         print("(d) done\t(r) restart the robot")
27         command = sys.stdin.read(2)
28
29         print("Command Thread Finished")
30
31
32 DEBUG = True
33
34 max_rotation_motor = 1023
35 Kp = 1
36 reference = 2
37
38 def print_debug(s, new_line):
39     global DEBUG
40
41     if DEBUG:
42         if new_line:
43             print(s)
44         else:
45             print(s, end="")
46
47 def start_server(HOST, PORT):
48     try:
49         # create an AF_INET, STREAM socket (TCP)
50         s = socket.socket(socket.AF_INET, socket.SOCK_STREAM)
51     except socket.error, msg:
52         print("[INFO] Failed to create socket. Error code: " + str(msg[0]) + " , Error message : " +
53               msg[1])
54         sys.exit()
```

```

55     print("[INFO] Socket created")
56
57     # Bind socket to local host and port
58     try:
59         s.bind((HOST, PORT))
60     except socket.error as msg:
61         print("[INFO] Bind failed. Error Code : " + str(msg[0]) + " Message " + msg[1])
62         sys.exit()
63
64     print("[INFO] Socket bind complete")
65
66     # Start listening on socket for ten clients
67     s.listen(10)
68     print("[INFO] Socket now listening")
69
70     return s
71
72 def verify_parameters (client):
73     # now keep talking with the client
74     package = client.recv(1024)
75
76     list_package = package.split()
77
78     if len(list_package) == 5:
79         time = int(list_package[0])
80
81         led_esq = int(list_package[1])
82         led_dir = int(list_package[2])
83
84         left_motor = float(list_package[3])
85         righ_motor = float(list_package[4])
86         #acel_z = float(list_package[4])
87
88         #degree = int      (list_package[5])
89         #giro_y = float(list_package[6])
90
91     else:
92         print_debug("[ERRO] Erro na leitura do sensor", True)
93         return 0, 0, 0, 0, 0
94
95
96     print_debug("tme:" + '{:07d}'.format(time), False)
97     print_debug("  lig:" + str(led_esq) + ":" + str(led_dir), False)
98     print_debug("  enc:" + '{:04d}'.format(int(left_motor)) + ":" +
99                 '{:04d}'.format(int(righ_motor)), False)
100
101     return time, led_esq, led_dir, left_motor, righ_motor
102
103 def send_new_rotations(client, left_rotation_motor, righ_rotation_motor):
104     new_package = str(left_rotation_motor) + " " + str(righ_rotation_motor)
105
106     client.sendall(new_package)
107
108 def send_command(client, s_string):
109     client.sendall(s_string)
110
111 def calcule_proportion(current_encoder, reference):
112     position = current_encoder
113     perfect_position = reference

```

```

114     proportion = perfect_position - position
115
116     if reference == 0:
117         proportion *= 10
118
119     error_value = proportion * Kp
120
121     return error_value
122
123 def calcule_spin(left_rotation_motor, righ_rotation_motor, left_error_value, righ_error_value,
124   ↪   left_light, righ_light):
125     min = -0 #max_rotation_motor
126     max = max_rotation_motor
127
128     if left_light == 0:
129         left_rotation_motor += float((left_error_value * 35.0 * abs(max_rotation_motor -
130           ↪   left_rotation_motor)) / 100)
131     else:
132         left_rotation_motor += -max
133
134     if righ_light == 0:
135         righ_rotation_motor += float((righ_error_value * 35.0 * abs(max_rotation_motor -
136           ↪   righ_rotation_motor)) / 100)
137     else:
138         righ_rotation_motor += -max
139
140     if left_rotation_motor > max:
141         left_rotation_motor = max
142     elif left_rotation_motor < min:
143         left_rotation_motor = min
144
145     if righ_rotation_motor > max:
146         righ_rotation_motor = max
147     elif righ_rotation_motor < min:
148         righ_rotation_motor = min
149
150     print_debug("      PWM(L:R):" + '{:04d}'.format(int(left_rotation_motor)) + "  " +
151       ↪   '{:04d}'.format(int(righ_rotation_motor)), True)
152
153     return int(left_rotation_motor), int(righ_rotation_motor)
154
155 def sensors_robot_procedure(client_connection):
156     print_debug("[INFO] <time> <left_light> <light_righ> <left_motor> <right_motor>", True)
157
158     global command
159
160     command = "r"
161     l_data_base = []
162
163     while "r" in command:
164         print("(y) to start the robot")
165         command = sys.stdin.read(2)
166
167         send_command(client_connection, "s")
168
169         sys.stdin.flush()
170
171         Command_thread().start()

```

```

170     l_data_base = []
171     last_left_rotation_motor = 0
172     last_righ_rotation_motor = 0
173     left_rotation_motor = 0
174     righ_rotation_motor = 0
175     left_sum_sensor = 0
176     righ_sum_sensor = 0
177     time = 0
178
179     command = "a"
180
181     # now keep talking with the client
182     while "a" in command:
183         # Verify the parameters
184         time, left_light, righ_light, current_left_motor_frames, current_righ_motor_frames =
185             ↪ verify_parameters(client_connection)
186
187         factor_left_rotation_motor = float(current_left_motor_frames - last_left_rotation_motor)
188         factor_righ_rotation_motor = float(current_righ_motor_frames - last_righ_rotation_motor)
189
190         if factor_left_rotation_motor > 12:
191             factor_left_rotation_motor = 0
192
193         if factor_righ_rotation_motor > 12:
194             factor_righ_rotation_motor = 0
195
196         last_left_rotation_motor = current_left_motor_frames
197         last_righ_rotation_motor = current_righ_motor_frames
198
199         print_debug("    |  Dif_btwn_Encd:" + str(int(factor_left_rotation_motor)) + " " +
200             ↪ str(int(factor_righ_rotation_motor)), False)
201
202
203         if left_light == 0:
204             left_error_value = calcule_proportion(factor_left_rotation_motor, reference)
205         else:
206             left_error_value = calcule_proportion(factor_left_rotation_motor, 0)
207
208         if righ_light == 0:
209             righ_error_value = calcule_proportion(factor_righ_rotation_motor, reference)
210         else:
211             righ_error_value = calcule_proportion(factor_righ_rotation_motor, 0)
212
213
214         print_debug("      ERROR(L:R):" + '{: 03d}'.format(int(left_error_value)) + " " +
215             ↪ '{: 03d}'.format(int(righ_error_value)), False)
216
217         left_rotation_motor, righ_rotation_motor = calcule_spin(left_rotation_motor,
218                                         righ_rotation_motor,
219                                         ↪ left_error_value,
220                                         ↪ righ_error_value,
221                                         ↪ left_light, righ_light)
222
223         l_data_base.append([time, left_rotation_motor, righ_rotation_motor,
224             ↪ current_left_motor_frames, current_righ_motor_frames])
225
226         send_new_rotations(client_connection, left_rotation_motor, righ_rotation_motor)

```

```

223     print_debug("[INFO] Command pressed: " + command, True)
224
225     send_new_rotations(client_connection, 0, 0)
226     l_data_base.append([time + 50, 0, 0, 0, 0])
227
228     command = "0"
229
230     return l_data_base
231
232 # =====
233
234 def send_new_rotations_slave(client, list_rotations):
235     reference = 0
236     client.sendall(str(len(list_rotations)) + "\n")
237
238     for i in range(0, len(list_rotations) - 1):
239         time = list_rotations[i][0]
240
241         if list_rotations[i][1] < max_rotation_motor:
242             left_motor = int(list_rotations[i][1])
243         else:
244             left_motor = list_rotations[i][1]
245
246         if list_rotations[i][2] < max_rotation_motor:
247             right_motor = int(list_rotations[i][2])
248         else:
249             right_motor = list_rotations[i][2]
250
251         left_motor_frames = list_rotations[i][3]
252         right_motor_frames = list_rotations[i][4]
253
254         client.sendall(str(int(time)) + " " + str(int(left_motor)) + " " + str(int(right_motor)) + "
255         →   " +
256                         str(int(left_motor_frames)) + " " + str(int(right_motor_frames)) + "\n\0")
257
258         client.recv(128)
259
260 def slave_robot_procedure(client_connection, l_data_base):
261     print_debug("[INFO] <time> <left_motor> <right_motor>", True)
262
263     global command
264     print("(y) to start the robot")
265     command = sys.stdin.read(2)
266
267     send_command(client_connection, "s")
268
269     sys.stdin.flush()
270     command = '0'
271
272     Command_thread().start()
273
274     print_debug("[INFO] Sending the Map", True)
275     send_new_rotations_slave(client_connection, l_data_base)
276
277     print_debug("[INFO] Sending Done", True)
278
279     print_debug("[INFO] Closing Connection", True)
280
281

```

```

282 def main():
283     # Address Family : AF_INET (this is IP version 4 or IPv4)
284     # Type : SOCK_STREAM (this means connection oriented TCP protocol)
285
286     global main_command
287     function.print_debug("[INFO] Starting the Ploudy", True)
288     ploudy_server = function.start_server(HOST, PORT)
289
290     l_data_base = []
291
292     while not "x" in main_command:
293
294         function.print_debug("[INFO] Waiting a new client... Size of List Map: " +
295             " " + str(len(l_data_base)), True)
296
297         # wait to accept a connection - blocking call
298         # addr0 is the IP and and addr1 is the ports
299         client_connection, client_addr = ploudy_server.accept()
300         function.print_debug("\n[INFO] Connected with " + client_addr[0] + " : " +
301             str(client_addr[1]), True)
302
303         robot = client_connection.recv(1024)
304
305         function.print_debug("\n\tReceived: \" " + robot + " \"", True)
306
307         print "(y) to accept\t(n) to recuse"
308         main_command = sys.stdin.read(2)
309
310
311         try:
312             l_data_base = function.sensors_robot_procedure(client_connection)
313
314             with open("master.map", 'wb') as f:
315                 pickle.dump(l_data_base, f)
316         except:
317             function.print_debug("[ERROR] Client was desconnected by a error!", True)
318
319         elif ROBOT_SENSORS in robot and 'y' in main_command:
320
321             print "(y) read from file"
322             main_command = sys.stdin.read(2)
323
324             if "y" in main_command:
325                 with open("master.map", 'rb') as f:
326                     l_data_base = pickle.load(f)
327
328             main_command = '0'
329             if len(l_data_base) > 0:
330                 try:
331                     function.slave_robot_procedure(client_connection, l_data_base)
332                 except:
333                     function.print_debug("[ERROR] Client was desconnected by a error!", True)
334             else:
335                 function.print_debug("[INFO] Client was desconnected by a error! \tEmpty map!",
336                                 " " + True)
337
338         function.print_debug("[ERRO] Client was desconnected!", True)

```

```
339
340     client_connection.close()
341
342     ploudy_server.close()
343
344 main()
```

---