CENTRO FEDERAL DE EDUCAÇÃO TECNOLÓGICA DE MINAS GERAIS

TRINCABOTZ-CEFETMG

Paulo Henrique Santos Constantino

RELATÓRIO DE PROJETO SEGUIDOR DE LINHA

Versão 1.0

Belo Horizonte
2020

Sumário

1. INTRODUÇÃO

Este relatório tem como objetivo tratar sobre o desenvolvimento do projeto seguidor de linha da equipe Trincabotz CEFET-MG, trazendo um entendimento geral sobre o projeto e sobre a categoria, de modo que, se o leitor ler cuidadosamente este documento e ter o mínimo de conhecimento técnico necessário, poderá ser capaz de entender e até mesmo reproduzir o que já foi feito.

Será abordado aqui, as ideias, passos e fases que o projeto passou até alcançar seu estado atual.

Vale ressaltar que, este relatório não tem caráter formal e que o projeto continua em constante evolução, sendo assim, algumas informações aqui relatadas, **podem NÃO estar totalmente corretas ou até mesmo ultrapassadas**. É recomendado que tenha pensamento crítico ao ler cada informação, e ao tentar reproduzir e/ou implementar algo visto aqui, adapte idéias para a sua necessidade e aplicação.

2. CATEGORIA

2.1 Conceito e Regras

Robôs seguidores de linha são robôs que tem o objetivo de seguir um certo trajeto de maneira autônoma.

No Brasil, o trajeto é uma linha branca contínua sobre uma superfície preta, ganha quem completar o percurso em menor tempo. Há marcações do lado esquerdo e direito do traçado, sendo a marcação direita a indicação de início e fim do trajeto e a marcação esquerda indica mudanças na curvatura na trajetória.

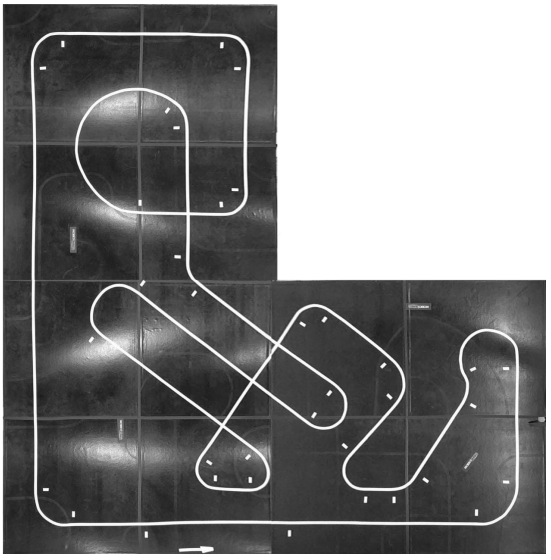


Figura 1: Trajeto do Winter Challenge XIV

Em robôs de combate a limitação da categoria é feita por peso, já os seguidores de linha são limitados por tamanho 25x25x20cm. Não é permitido que as rodas do robô sejam grudentas, também não é permitido o uso de ventoinhas como dispositivos de sucção.

Os competidores podem fazer testes na pista antes da tomada de tempo oficial, e em geral se tem mais de duas oportunidades de se realizar a tomada de tempo durante a competição.

Cada tomada de tempo consiste em 3 tentativas de passagens do seguidor na pista, uma volta é validada quando o trajeto é completado sem que o robô saia

completamente de cima da linha branca. O seguidor deve parar completamente de maneira autônoma dentro da área de chegada, sem que o operador interfira.

Essas regras são as usadas atualmente pelas competições de seguidores de linha organizadas pela Robocore (responsável pelas maiores competições de robótica no Brasil). Em outros eventos, as regras podem ser diferentes.

Todas as regras e especificações necessárias a serem seguidas tanto para o robô quanto para a pista, podem ser encontradas no site da Robocore : https://www.robocore.net/upload/attachments/robocore__regras_seguidor_de_linha_108.pdf

2.2 Projetos da equipe

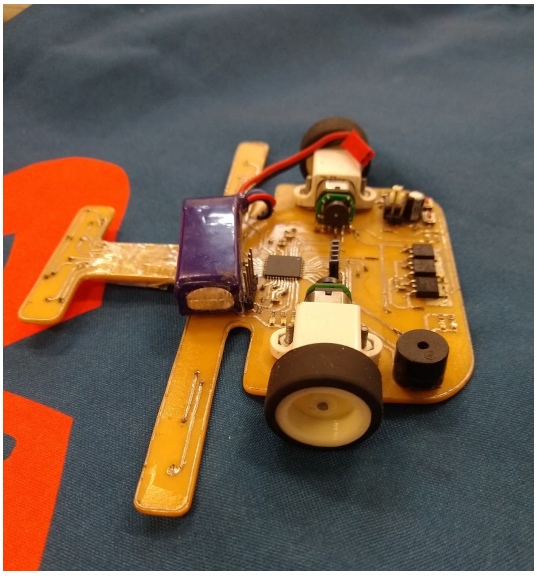
A equipe Trincabotz desenvolve seu projeto de seguidor de linha desde o ano de 2014, desde então, segue em constante aprimoramento do mesmo.

Neste relatório será tratado o desenvolvimento de projetos de seguidores de linha com base na experiência obtida na execução dos projetos Ártemis e Nêmesis.

2.2.1 Ártemis

Ártemis foi criada no início de 2016 e tem sofrido mudanças constantemente. Sua versão de 2019 conta com seis sensores de infravermelho alinhados lado a lado que fazem a leitura da pista, dois sensores laterais no lado esquerdo e um sensor no lado direito para leitura de marcações.

O projeto conta também com encoders magnéticos, que fazem a leitura de velocidade e distância percorrida de cada roda do robô, LEDs, botões e um buzzer piezoelétrico para uso geral.



Peso : 78g

Motores: Micro Metal Gearmotors 10:1 6v dual-shaft com encoder magnético pololu

Microcontrolador: PIC32mx170f256d

Bateria: LiPo 300mAh 2S 35~70C

Interface: 5x LEDs, 2x botões, 1x buzzer, Bluetooth

Rodas: Cubo de impressão 3D, pneus Kyosho mini-z 2-20

Motor Driver: DRV 8871

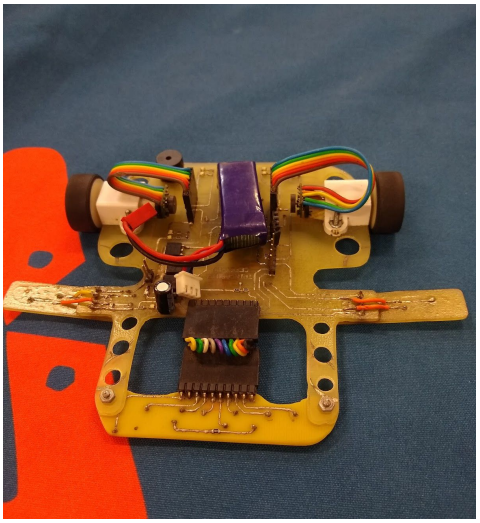
Figura 2 : Ártemis 2019 em fenolite

2.2.2 Nêmesis

Projetado em 2017, Nêmesis estreou em 2018 no Winter Challenge XIV.

O projeto de 2019 conta com quatro placas independentes: o array com sete sensores infravermelho, duas placas de sensores laterais, cada uma com dois sensores para detectar marcações, e a placa de controle.

O robô tem componentes muito similares a Ártemis, o que muda é seu microcontrolador com memória reduzida, placas independentes que permitem a regulagem de altura dos sensores e componentes de interface.



Peso : 104g

Motores: Micro Metal Gearmotors 10:1 6v dual- shaft com encoder magnético pololu

Microcontrolador: PIC32mx110f016d

Bateria: LiPo 300mAh 2S 35~70C

Interface: 2x LEDs, 1x buzzer, Bluetooth

Rodas: Cubo de impressão 3D, pneus Kyosho mini-z 2-20

Motor Driver: DRV 8871

Figura 3: Nemesis 2019 em fenolite

3. MECÂNICA

3.1 Dinâmica

O projeto mecânico foi pensado e construído analisando aspectos como: **centro de gravidade**, **torque** e **motores**. As figuras a seguir, foram adaptadas e traduzidas de uma apresentação de NG beng kiat (link na seção de referências), e demonstram aspectos da dinâmica de um seguidor de linha.

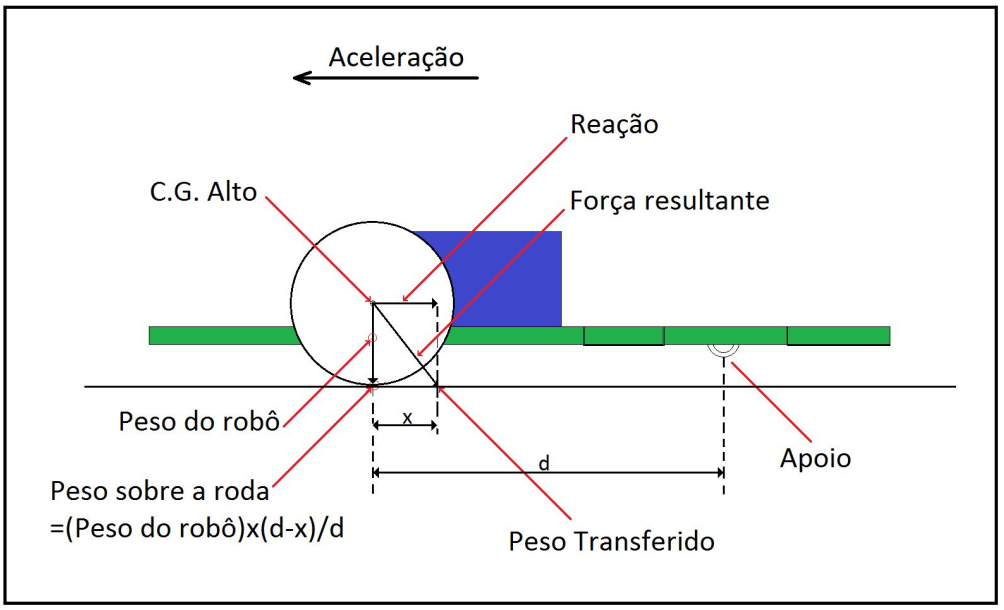


Figura 4: Peso transferido com centro de massa alto

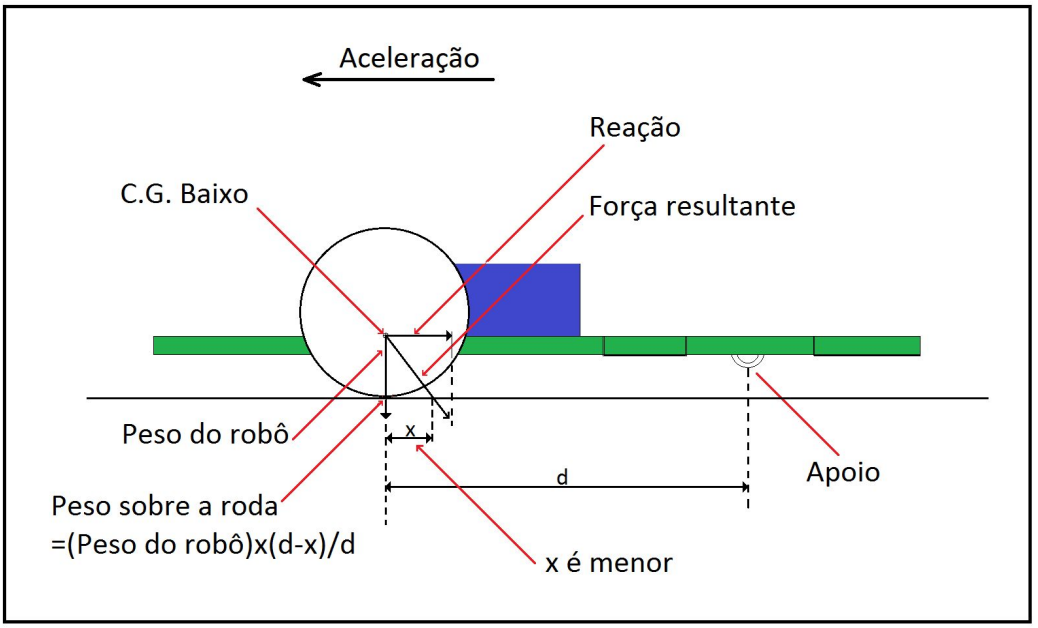


Figura 5: Peso transferido com centro de massa baixo

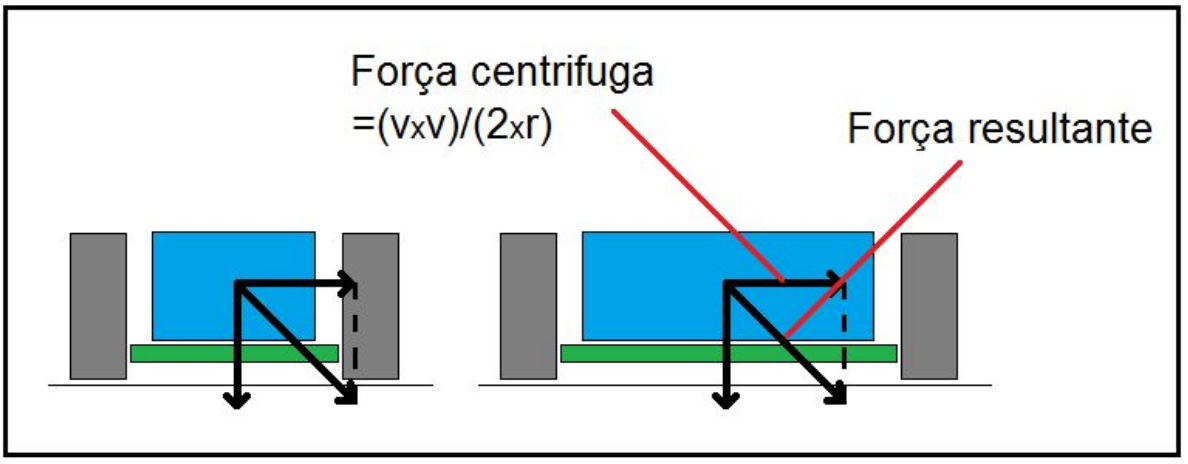


Figura 6: Força centrífuga resultante em diferentes distâncias entre rodas.

Nota-se visualmente a importância de um projeto com centro de gravidade baixo e rodas não muito próximas uma das outras, devido a transferência de peso e força centrífuga, causados quando o robô sofre uma aceleração (translacional ou rotacional). Estes são **alguns** dos motivos que fazem Ártemis e Nêmesis terem rodas e baterias relativamente pequenas. Além disso, foi colocado alguns calços embaixo do motor, com o objetivo de deixar o corpo do robô o mais rente do chão possível.

Todos os componentes, com exceção do bluetooth que por sua vez não é utilizado no momento em que o projeto está “tomando” tempo, foram soldados direto na placa de circuito impresso, evitando assim, peso de módulos e componentes que fariam o robô mais pesado e com centro de gravidade mais alto.

3.2 Motores

Um dos pontos mais importantes a serem observados na construção de um seguidor de linha são os motores. Saber dimensionar adequadamente os motores tornam o projeto mais competitivo, eficiente e com melhor custo-benefício.

Para um a escolha de um motor adequado para o projeto, deve-se levar em conta peso do robô, tamanho da roda, velocidade e aceleração máxima desejada. Para fins de comparação os projetos com melhor desempenho no *All Japan Robotrace Contest* (a maior e mais competitiva competição de seguidores de linha no mundo) tem em média um peso de 100g, rodas de 25mm de diâmetro, velocidade de até 6m/s e aceleração máxima de até 20m/s². No entanto, para sustentar essas características, são necessários motores de alta performance, que em sua grande maioria são caros.

Ártemis, em sua primeira versão de 2016, pesava cerca de 150g e utilizou motores Micro Metal Gearmotors 30:1 HPCB 12V. Este conjunto (motor e redução), utilizando a tensão nominal de 12V, tem potência máxima de 1.1W, *stall* torque de 39 mNm e rotação de no máximo 1100 rpm. Esta configuração deixou o robô com bom torque, fato que o deixa mais fácil de se controlar, mas por outro lado a velocidade máxima é reduzida.

Em 2017 o motor utilizado foi o Micro Metal Gearmotors 10:1 HPCB 6V, que por sua vez tem potência máxima de 1.3W, *stall* torque de 17 mNm e rotação de no máximo 3100 rpm. Tal mudança visou um aumento de velocidade do seguidor, e utilização de uma bateria LiPO 2s (8.4V) mais leve e comercialmente mais fácil de se encontrar. Utilizando esta configuração, é notável que o sistema de controle deve ser aprimorado e que o peso do robô seja reduzido.A falta de torque no conjunto faz com que seja mais difícil de se controlar o robô utilizando um PID de malha simples (mais comumente utilizado pelos competidores no Brasil).

3.3 Estruturas

A estrutura mecânica do robô é um ponto chave a ser observado, pois é nela que se aplica os conceitos vistos na seção 3.1.

Os projetos da equipe Trincabotz usam a própria placa de circuito impresso como corpo/base dos robôs. Desta forma, evita-se que seja necessário uma estrutura e uma placa de controle por cima, reduzindo consideravelmente o peso do protótipo.

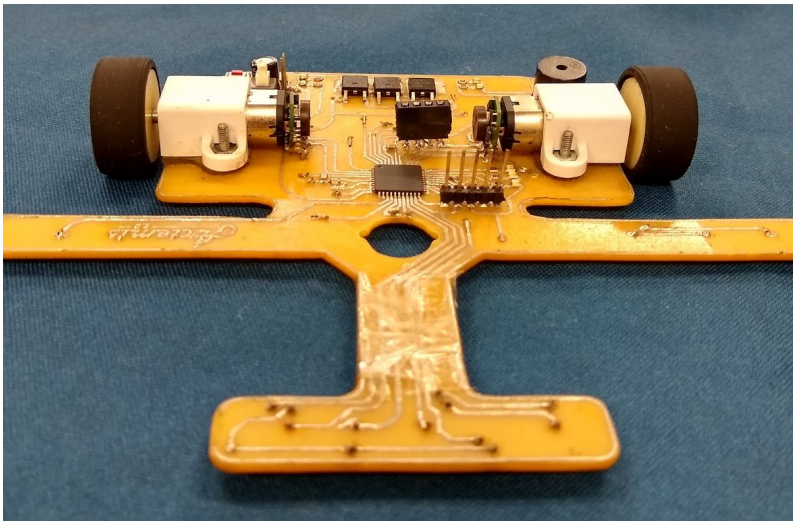


Figura 7: Estrutura da Ártemis 2019

3.3.1 Rodas

É importante que se utilize rodas com boa aderência com a pista, para que o robô consiga fazer curvas com maior velocidade e sem deslizar.

Em sua primeira versão, Ártemis utilizou rodas *Pololu Wheel 32×7mm.*

Figura 8: Roda Pololu

Estas são rodas de ótima qualidade para projetos iniciantes/intermediários pois são muito leves e com “pneus” de borracha que adere bem à pista.

Foram feitas também tentativas de fabricação de rodas e pneus próprios. Utilizou-se cubos de rodas e moldes (para a fabricação dos pneus) de impressão 3D. Os pneus foram confeccionados utilizando-se uma mistura de silicone, amido de milho e corante.

Infelizmente, estes pneus caseiros se desgastavam muito rápido e, com poucas voltas na pista, perdiam a aderência devido ao acúmulo de sujeira. Sendo assim, optou-se por utilizar as rodas da pololu de 32mm.

Atualmente os dois seguidores de linha da Trincabotz utilizam cubos de roda feitos de impressão 3D projetados pela própria equipe e pneus de borracha da KYOSHO do modelo mzw 2-20. Este pneu tem uma dureza de 20 shore (segundo o catálogo da Kyosho) e demonstrou empiricamente um desempenho superior às anteriores.

Figura 9: Roda kyosho

Observações :

O coeficiente de atrito estático entre o pneu e a pista é um ponto importante a se observar uma vez que, quanto maior o valor deste coeficiente, mais rápido o robô consegue executar uma curva sem deslizar.

Considerando-se que o robô não tenha mecanismo para aumentar a sua normal e que esteja sobre um plano não inclinado, é possível chegar na seguinte equação : $V máx = \sqrt{R.g.\mu}$

Onde: *Vmáx* é a velocidade máxima na curva sem que haja deslizamento;

R é o raio da curva;

g é a aceleração da gravidade ;

μ é o coeficiente de atrito estático;

Esta fórmula de velocidade máxima em uma curva é importante quando se deseja obter o desempenho máximo do protótipo, de forma com que o robô corra nas curvas (de raio conhecido) sem deslizar. Sendo assim, é importante que se tenha uma estimativa do módulo de μ.

É possível obter-se o valor do coeficiente de atrito estático entre a pista e as rodas, através de um experimento simples. Basta utilizar a borracha da pista como uma base plana que possa ser inclinada e colocar sobre a pista o material que queira medir o coeficiente de atrito estático, como mostrado na figura.

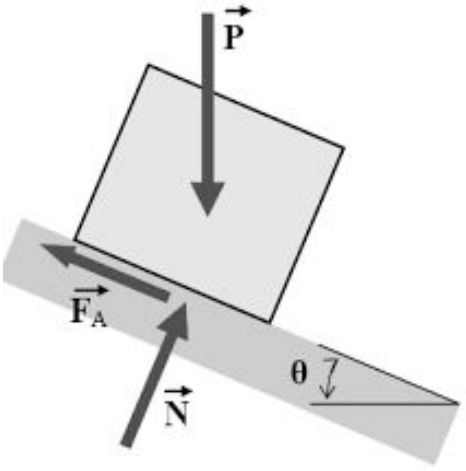


Figura 10: Diagrama de forças

Em seguida, deve-se aumentar o ângulo Θ até o exato momento em que o material sobre a pista comece a deslizar. Desta forma, o coeficiente de atrito μ pode ser dado pela seguinte equação:

$$\mu = tan\ \Theta$$

Onde Θ é o ângulo da “pista” ,em relação ao chão, em que o material que está sobre a superfície começa a deslizar.

4. ELETRÔNICA

O dimensionamento correto de todos os componentes eletrônicos do robô, evitam “dores de cabeça” desnecessárias que atrasam a evolução do software, por isso é sempre bom optar pelo seguro quando se trata da eletrônica em geral.

Neste tópico, será tratado o funcionamento básico e o dimensionamento dos principais componentes eletrônicos, divididos em potência e controle, presentes em nosso projeto.

4.1 Potência

- DRV8871

Este componente é responsável por controlar a velocidade e o sentido de giro de um motor DC (Micro Metal Pololu). Este driver opera com a faixa de tensão de 6,5v até 45v, e suporta uma corrente de pico de até 3,6A (mais que o dobro da stall current de 1.6A do motor pololu 6v).

Basicamente o componente opera recebendo 2 sinais PWM, e de acordo com essa combinação, é possível definir o sentido de giro e a velocidade para o motor (baseado na tabela verdade presente no datasheet do CI).

Observação: Os motores de Ártemis e Nêmesis são alimentados com aproximadamente 8v (overvoltage), com a alimentação de 6v (nominal do motor) este componente **não** funciona.

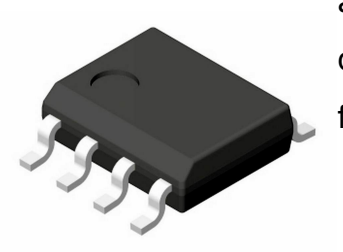


Figura 11: DRV8871

- LM7805 e LM7833

Ambos LMs 7805 e 7833 são reguladores lineares de tensão que alteram a tensão de entrada de 8.2v da bateria para tensões de saída de 5v e 3.3v constantes (utilizados na maioria dos componentes da placa) e permitem a passagem de 1,5A cada.

Este dispositivo regula a tensão através de dissipação de potência em forma de calor, o que o torna menos eficiente que um regulador chaveado. Por outro lado, é um CI que exige pouquíssimos componentes externos, tornando-o muito simples de ser utilizado.

Figura 12: 7805 smd

Observação: Deve-se calcular a quanto de corrente será exigida de cada regulador, evitando assim, mal funcionamento e sobreaquecimento.

Como medida de segurança, nossa placa utiliza 2 7805 em paralelo para permitir uma maior passagem de corrente e melhorar a dissipação de potência.

- BATERIA

A escolha da bateria foi feita levando em conta a tensão máxima necessária, corrente máxima e peso. Baterias Lithium Polymer (Li-Po) são geralmente as mais usadas, pelo fato de terem uma grande densidade de energia e serem recarregáveis.

Foi estimado então que a corrente máxima possível que o projeto exige é de aproximadamente 6 amperes.

A bateria escolhida tem carga de 300mah e taxa de descarga de 35C ~ 70C e supre os requisitos necessários:

$$Imax < Taxa\ de\ descarga \times Carga\ da\ bateria$$

$$6A < 35c \times 0,3A \Rightarrow 6A < 10,5A$$

Para calcular o tempo em que a bateria irá se descarregar, foi considerado que o robô irá consumir uma corrente média de 1,5A, e então, utilizando a seguinte equação :

$$Tdescarga = \frac{Carga}{Corrente} \times 60$$

$$Tdescarga = \frac{0,300Ah}{1,5A} \times 60 = 12min$$

Com um tempo de descarga de 12min, Ártemis e Nêmesis conseguem rodar muitas vezes em pistas não muito longas.

4.2 Controle

- QRE1113

Para identificação da linha do trajeto, foi utilizado o sensor QRE1113. Este sensor é composto por um LED emissor de luz infravermelha e um transistor de base aberta que funciona como um receptor.

Figura 13: QRE1113

O circuito dimensionado para a utilização do circuito foi o seguinte:

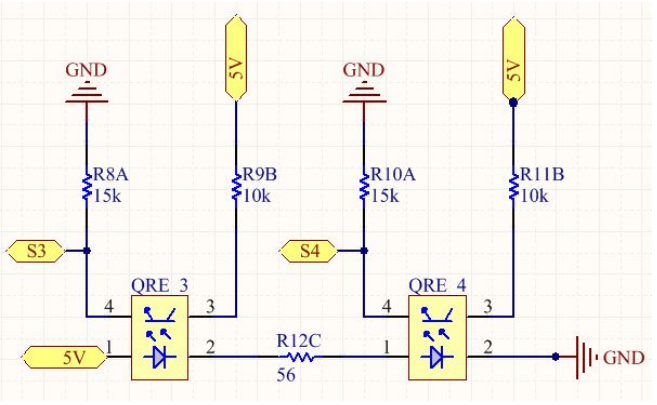


Figura14: Circuito para utilização do sensor

Dessa forma, a corrente I_L que passa pelos LEDs infravermelhos segue a seguinte equação :

$$I_L = \frac{Vcc - Vled - Vled}{R} = \frac{5 - 1{,}4 - 1{,}4}{56} = \frac{2{,}2}{56}$$

I_L = 39,29mA

Desta forma, a corrente calculada I_L , está próxima à corrente de funcionamento típica do sensor que é de 40mA.

Para se obter a resposta do sensor, foi necessário implementar um divisor de tensão, pelo fato de que o microcontrolador utilizado aceitar apenas tensões abaixo de 3,3v. Sendo assim, a tensão V_{out} sobre o resistor R_e do emissor do transistor deve ser menor ou igual a 3,3v, pois esta será usada como a resposta do sensor. A tensão máxima V_{out} da resposta do sensor ocorre quando o transistor está em sua maior saturação.Então, V_{out} pode ser calculada pela seguinte equação :

$$V_{out} = \frac{Vcc - Vbe}{Re + Rc} \times Re = \frac{5 - 0{,}3}{10000 + 15000} = \frac{4{,}7}{25000} \times 15000$$

V_{out} = 2,82v

- Encoder magnético

Para o monitoramento da velocidade e a distância percorrida pelo robô durante seu trajeto, são utilizados dois encoders magnéticos de quadratura da Pololu, um para cada motor.

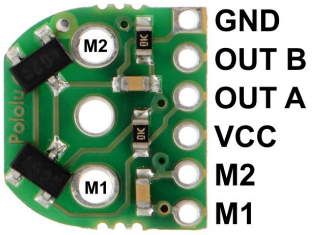


Figura 15: Placa do encoder magnético

Figura 16: Encoder instalado no motor

Como pode ser observado na figura 16, esta placa vai acoplada diretamente nos dois terminais do motor. As duas respostas do encoder, OUT A e OUT B, são defasadas de 90°, o que permite saber o sentido de giro do motor. A resolução deste encoder é de 12 pulsos por revolução, mas o valor que irá aparecer nas saídas é multiplicado pelo fator de redução, que em nosso caso é de 10:1, logo obtemos 120 pulsos por revolução.

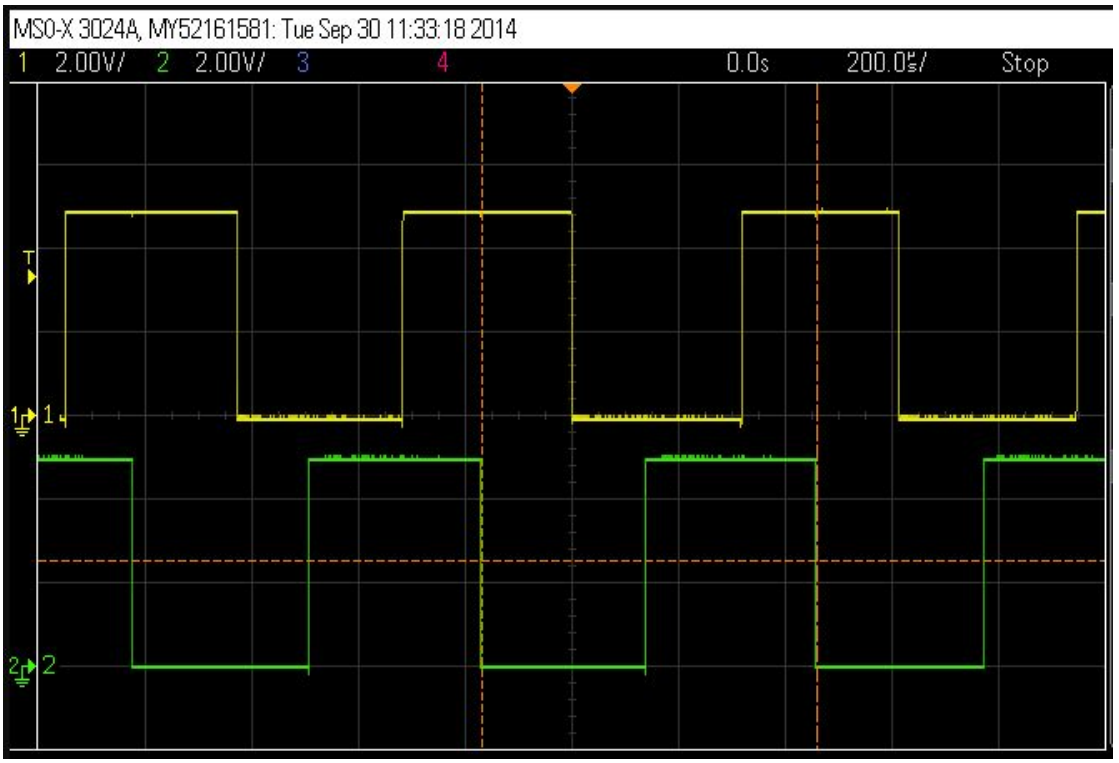


Figura 17: Sinais de saída do encoder observado por osciloscópio

- HC05

Com o intuito de realizar transferências de dados entre o robô e um computador/celular, o módulo bluetooth HC05 foi escolhido. A transferência de dados é feita de maneira serial através dos pinos 2 (RX) e 3 (TX). Pode ser configurado um Baud Rate de 4800 até 1382400 e sua tensão de alimentação é de 3,3v, podendo variar de 2,7v até 4,2v.

Figura 18: Módulo HC05

Para a utilização deste módulo, foi adicionado um conector *header* fêmea de 4 entradas na placa de circuito impresso. Estes conectores são responsáveis por receber os quatro pinos centrais do módulo, RX,TX,GND e VCC (apenas estes quatro pinos já são suficientes para o funcionamento do HC05).

- Microcontrolador

O cérebro do projeto é um microcontrolador PIC de 32 bits. O modelo pode ser o pic32mx110f016d ou o pic32mx170f256d, suas únicas diferenças são os espaços de memória de dados e de programa. As principais características dos microcontroladores podem ser visualizadas na tabela x.

Nome	Valor	
Componente	32MX110f016D	32MX170f256d
Max CPU Speed MHz	40	50
Program Memory Size(KB)	16	256
SRAM(KB)	4	64
SPI	2	2
I2C	2	2
ADC Input	13	13
ADC Resolution (Bits)	10	10
16-bit Timer	5	5
PWM	5	5
I/O Pins	34	34

Estes componentes tem a velocidade de processamento e periféricos suficientes para executar todas tarefas exigidas pelo projeto com desempenho satisfatório.

Atualmente o projeto roda utilizando clock interno para gerar uma frequência de 40MHz, e tem como tarefas simultâneas: Conversão de 7 canais analógicos, utilização de 4 saídas PWM, comunicação UART com módulo Bluetooth, utilização de 4 timers sendo 1 como fonte de interrupção, 4 canais gerando interrupção externa para leitura do encoder.

Observação:

O projeto Ártemis no seu início em 2016, utilizou apenas o microcontrolador do modelo PIC32MX110F016D por ser mais barato e atender as especificações do projeto. Foi a primeira vez que a equipe implementou um microcontrolador de 32bits em um projeto. Com o desenvolvimento do código do seguidor de linha, a memória de programa de 16KB não estava sendo o suficiente, por este motivo optou-se por um modelo com mais memória e com a mesma pinagem.

5. LÓGICA

Para um desempenho satisfatório do projeto, é necessário uma utilização correta dos periféricos e técnicas de controle para uma minimização de erros.

Neste tópico, será tratado como é feita a leitura dos sensores de linha e encoder, aplicação da técnica PID, a obtenção de dados da pista e técnica para se percorrer a pista de maneira mais eficiente.

5.1 Sensores

- Sensor de linha e marcações

A leitura dos sensores infravermelhos que são utilizados para ler a linha branca é feita utilizando-se um conversor analógico digital de 10 bits (periférico presente no microcontrolador utilizado). Optou-se por não tratar a leitura do sensor como binário (na linha ou não) para se obter uma melhor precisão de quão distante os sensores estão da linha.

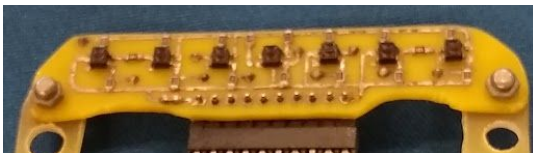


Figura 19: Placa de sensores

Para a leitura das marcações da pista, a leitura dos sensores é feita de forma binária. A leitura é feita de forma periódica a cada 100ms, e para se reconhecer a marcação o algoritmo espera uma borda de descida (sensor sai do branco e vai para parte preta da pista). Para se evitar a leitura errada de marcação, como por exemplo um cruzamento de linhas, antes de se validar a marcação é verificado se o sensor oposto não está sobre a linha, por exemplo: Quando o sensor direito acusa uma borda de descida a próxima linha de código verifica se o sensor esquerdo está no preto.

- Encoder

Para a utilização dos encoders de quadratura, é utilizado quatro interrupções por mudança de estado (change notice), uma para cada canal dos encoders. Desta forma, o algoritmo consegue ler e tratar todas as bordas de subida e descida que os dois encoders fornecem.

Ao ocorrer a interrupção, o primeiro passo é conferir em que sentido o motor está girando através de uma condição que verifica uma combinação entre o estado anterior e o estado atual dos dois canais de cada encoder. Caso ocorra a interrupção e o sentido verificado for correto, uma variável para distância é incrementada e a velocidade é calculada.

Para o cálculo de distância e velocidade, é necessário ter ciência de alguns dados como resolução do encoder e perímetro da roda utilizada. O encoder da pololu fornece 120 pulsos por rotação e a roda utilizada na Ártemis e Nêmesis tem aproximadamente 75,39mm de perímetro, desta forma sabe-se que cada pulso do encoder representa $\frac{120}{75,39}$ = 0,628mm rodados. Desta forma, sabe-se a cada interrupção do encoder a roda percorreu 0,628mm, tendo conhecimento do número de interrupções geradas em cada trecho é possível ter uma boa aproximação do tamanho real de cada reta e curva da pista.

Para estimar a velocidade real do robô, utiliza-se dois timers (um para cada motor). Como se tem conhecimento da distância percorrida a cada interrupção, utiliza-se o timer para contar o tempo entre cada interrupção. Sendo assim, basta usar a fórmula $\frac{0,628\ (Distancia\ [mm])}{Timer\ (Tempo\ [s])}$.

5.2 PID

O seguidor de linha tem o objetivo de se manter corretamente sobre o seu trajeto (a linha), mas para que isso aconteça é necessário um controle para minimizar os erros.

O sistema descrito aqui, tem o objetivo de controlar as velocidades **rotacional** e **translacional** do robô. A velocidade rotacional é a variação da posição do robô em torno do seu eixo central e a velocidade translacional o movimento que faz o robô se

locomover para trás ou para frente. Estas duas velocidades são controladas de forma independente pelo sistema de controle PID.

Controlador proporcional integral derivativo ou simplesmente PID, é uma técnica de controle de processos que une as ações derivativa, integral e proporcional, fazendo assim com que o sinal de erro seja minimizado pela ação proporcional, zerado pela ação integral e obtido com uma velocidade antecipativa pela ação derivativa.

Neste controlador são ajustados dois setpoints, a velocidade translacional (speedX) e a velocidade rotacional (speedW), sendo:

speedX = VelocidadeRodaDireita + VelocidadeRodaEsquerda

speedR = VelocidadeRodaDireita - VelocidadeRodaEsquerda

O setpoint para speedX é a velocidade que se deseja alcançar translacionalmente e é puramente decidida pelo usuário, já o setpoint para speedR é decidido pelos sensores de linha que definem qual deve ser a velocidade rotacional para que o seguidor se mantenha na linha. Esta velocidade pode ser obtida facilmente fazendo-se uma média ponderada com os valores lidos pelos sensores de linha.

As variáveis manipuladas são as razões cíclicas (duty cycle) dos dois motores. Em uma reta o speedW deve ser zero, enquanto que em curvas speedX e speedW são constantes diferentes de zero, e para realizar uma curva em torno do próprio eixo do robô (curva pivot) speedX deverá ser igual a zero.

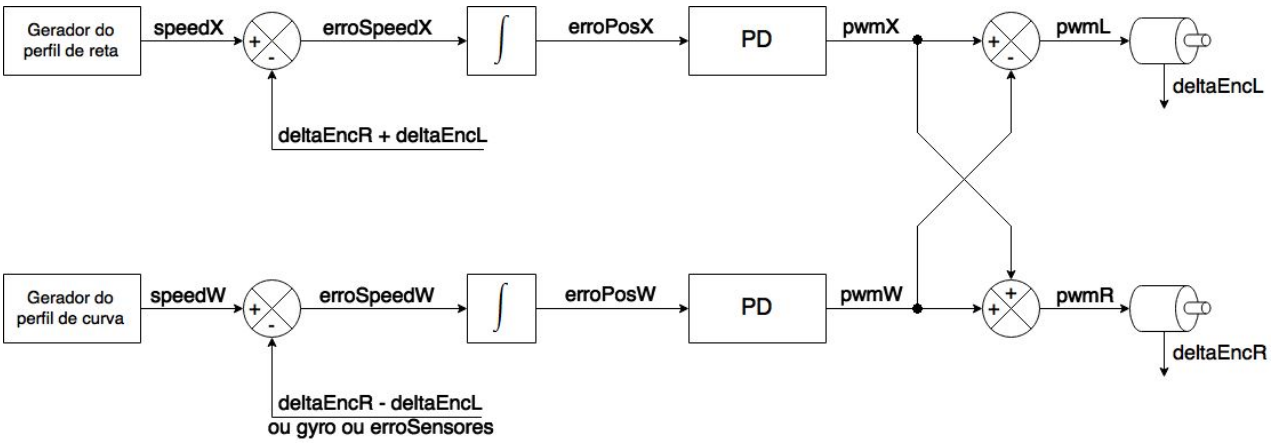


Figura 20: Malha de controle PD

Código de exemplo que implementa este controlador:

```
// Controlador de velocidade para movimento translacional + rotacional
void controlador(void)
```

```
{
    // Calcula as realimentações dos encoders, sensores e giroscópio
    // K_GYRO e K_SENSORES são constantes para ajustar as escalas
    feedbackEncW = deltaEncR - deltaEncL;
    feedbackSensores = erroSensores / K_SENSORES;
    feedbackGyro = gyro / K_GYRO;

    // Calcula os PVs (process variable) do controlador
    pvX = deltaEncR + deltaEncL;
    pvW = 0;
    if(usarEncoders) pvW += feedbackEncW;
    if(usarSensores) pvW += feedbackSensores;
    if(usarGyro) pvW += feedbackGyro;

    // Calcula os erros (erro = Setpoint - PV)
    // O "+=" representa a operação integrativa da velocidade = posição
    erroPosX += speedX - pvX;
    erroPosW += speedW - pvW;

    // Realiza o controlador PD (Proporcional Derivativo)
    // KP_X, KD_X, KP_W e KD_W são as constantes do controlador PD
    posPwmX = KP_X * erroPosX + KD_X * (erroPosX - erroAnteriorPosX);
    posPwmW = KP_W * erroPosW + KD_W * (erroPosW - erroAnteriorPosW);

    // Registra os erros anteriores para a ação derivativa
    erroAnteriorPosX = erroPosX;
    erroAnteriorPosW = erroPosW;

    // Calcula os PWMs dos motores da esquerda e da direita
    pwmL = pwmX - pwmW;
    pwmR = pwmX + pwmW;

    // Realiza o comando para atualizar os PWMs dos motores
    setPwmL(pwmL);
    setPwmR(pwmR);
}
```

Observação: esta função deve ser chamada em um intervalo fixo de tempo, por exemplo: 1ms ou 10ms.

Uma das técnicas utilizadas para gerar os setpoints das velocidades é a de perfil de velocidade (speed profiler), a qual gera um perfil trapezoidal de velocidade em função do tempo, conforme a figura a seguir, e com isso, a distância percorrida pelo robô é obtida a partir da área do trapézio. As rampas entre [t0; t1] e [t2; t3] são necessárias, pois não é possível acelerar ou desacelerar instantaneamente o robô.

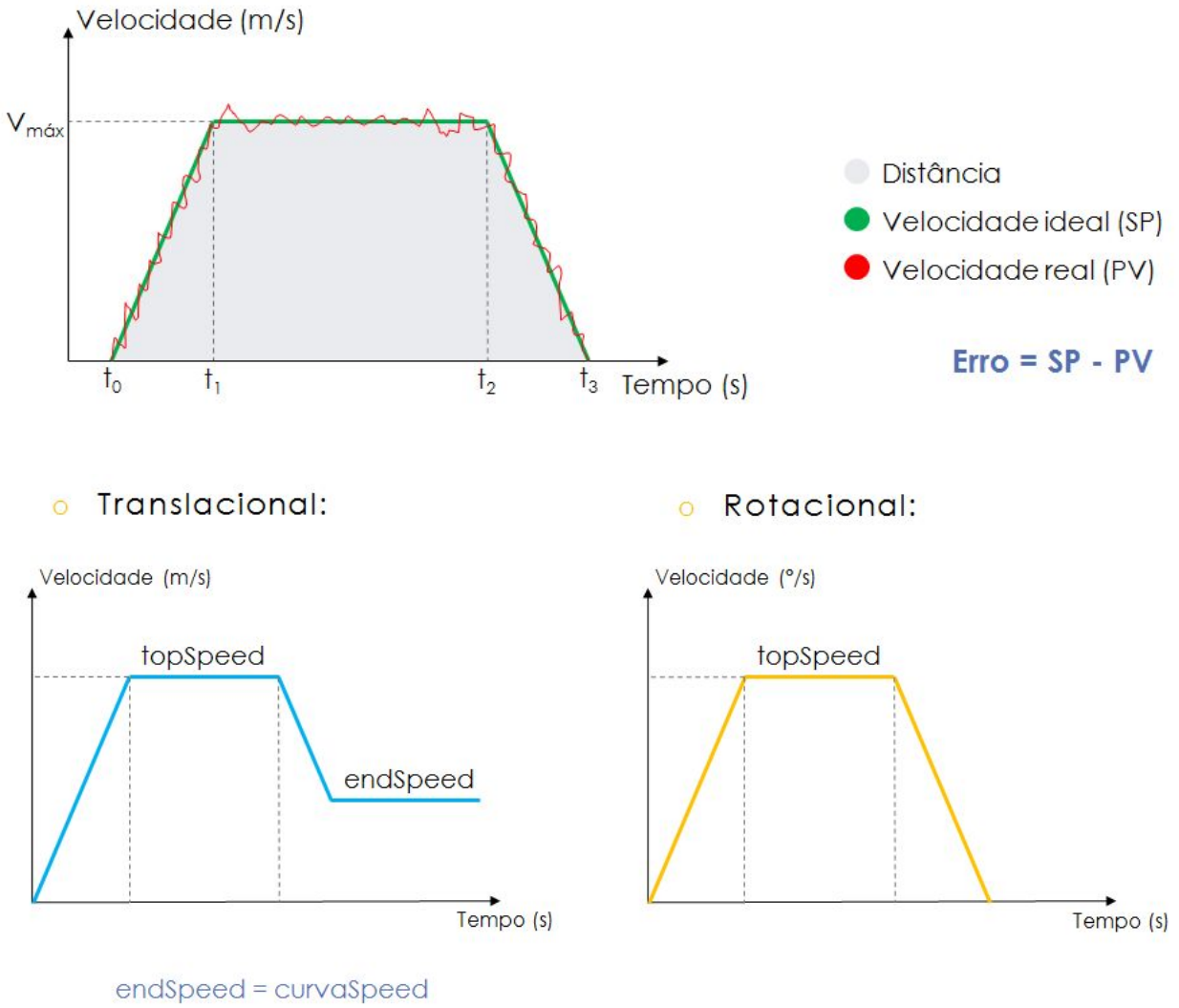


Figura 21: Perfil de velocidade rotacional e translacional ao longo do tempo.

A geração de perfil de velocidades, será tratado na seção 5.4 RUN.

Observação : Para um melhor entendimento do assunto tratado nesta seção é recomendado ler e ver o conteúdo da aula 3 do project futura.

5.3 Dados da pista

É desejável que o seguidor de linha corra o mais rápido possível durante todo o trajeto, mas diferentes trechos da pista exigem diferente velocidades. Em uma reta, teoricamente, é possível alcançar a velocidade máxima do projeto mas em uma curva esta velocidade deve ser controlada em função do raio desta curvatura. Por esta causa, a obtenção de dados da pista é muito importante quando se quer melhorar ao máximo a performance do robô.

Utilizando-se encoders em cada motor, é possível ter o conhecimento da distância que cada roda percorreu, como visto na parte *Encoder* da seção 5.1 Sensores . Com este pretexto, foi desenvolvido um código básico de mapeamento da pista, seu funcionamento pode ser descrito da seguinte forma:

- A pista inteira é dividida em trechos, sendo que cada trecho é o intervalo entre uma marcação e outra. Isto inclui as marcações da direita (início e fim de percurso).
- De início o robô percorre a pista de maneira mais lenta e cautelosa obtendo os dados.
- E então, para se obter o valor que cada roda percorre em cada trecho, o número de pulsos é acrescentado e salvo em uma variável 'x' a cada interrupção do encoder. Quando ocorre alguma marcação direita ou esquerda, o valor salvo dentro de x é salvo em uma matriz de dados e em seguida o valor de x é zerado para que se possa salvar o valor do próximo trecho.
- Posteriormente, quando ocorre a segunda marcação direita e todos os valores de encoder já estão salvos na matriz, estes dados são enviados, via bluetooth, para uma interface computadorizada que executa o tratamento das informações.

Após ser feito o mapeamento do percurso, utilizando o algoritmo descrito acima, é possível se obter informações de distância e raio de cada curva utilizando alguns fatos e aproximações.

Primeiramente, todos os trechos da pista são considerados como curvas e o que estabelecerá de fato a diferença entre um trecho curvo e um trecho reto será o valor calculado de raio. Uma curva com raio muito grande, será considerado como uma reta.

Para se fazer o cálculo de raio (R) de cada trecho é necessário conhecer as seguintes variáveis : Distância entre as rodas do robô (D), distância percorrida pela roda direita (Dd) e esquerda (De).

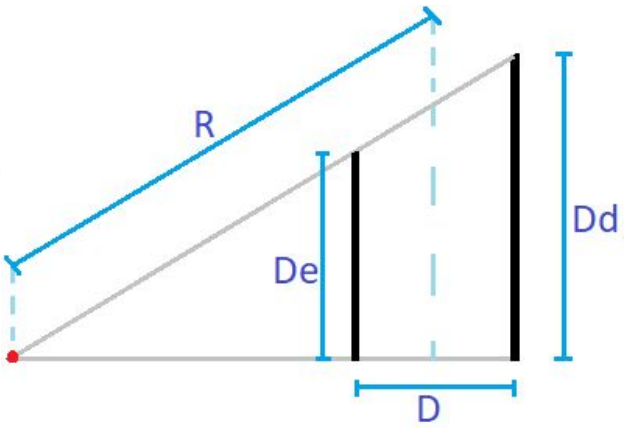


Figura 22: Distâncias em uma curva

A figura ilustra a distância percorrida por cada roda em uma curva feita no sentido anti-horário. Utilizando semelhança de triângulos é possível determinar a valor de R que é a distância entre o ponto vermelho (centro da curva) e o ponto médio de D. Com as devidas deduções feitas, temos :

$$R = \left|(\frac{D}{2})x\frac{Dd+De}{Dd-De}\right| \mid Dd \neq De$$

Com esta equação podemos definir a amplitude do raio para todos os trechos do percurso, tomando o devido cuidado de condicionar Dd≠De. Caso Dd seja igual De, é definido que o raio é infinito, ou seja, o trecho é uma reta perfeita.

E então, com os valores de raios obtidos, é possível calcular a velocidade máxima possível em cada trecho, utilizando a equação abaixo, vista anteriormente na seção 3.3.1 Rodas :

$$V\,máx = \sqrt{R.g.\mu}$$

Onde: *Vmáx* é a velocidade máxima na curva sem que haja deslizamento;

R é p raio da curva;

g é a aceleração da gravidade ;

μ é o coeficiente de atrito estático;

Desta forma, é possível se obter o tamanho, em metros, e velocidade máxima, em metros por segundos, de cada trecho. No entanto, dependendo to tamanho e número de trechos de cada percurso, se torna inviável fazer estes cálculos um a um. Por este motivo, a equipe Trincabotz desenvolveu um software simples que auxilia no mapeamento e cálculos do percurso.

5.3.1 Interface

Esta interface foi desenvolvida utilizando o software Visual Studio e foi aprimorada de acordo com cada necessidade do projeto.

Figura 23: Tela principal da interface desenvolvida

Sua principal função consiste em obter os dados da pista que o seguidor de linha transfere via bluetooth, interpretar as informações, calcular as velocidades máximas em cada trecho e converter isto em linhas de código de maneira que o operador possa apenas gerar, copiar e colar estas linhas no código principal.

Além disso, outras funcionalidades também foram adicionadas como o intuito de se realizar um debug do sistema, como:

- **Velocidade real**

Esta função tem a finalidade de receber dados referente a velocidade real do robô em um certo intervalo de tempo. Utilizando estas informações, é traçado um gráfico de velocidade[m/s] x tempo [s], onde é possível se observar valores de aceleração e erros, e então pode-se fazer um melhor ajuste das malhas de controle PID.

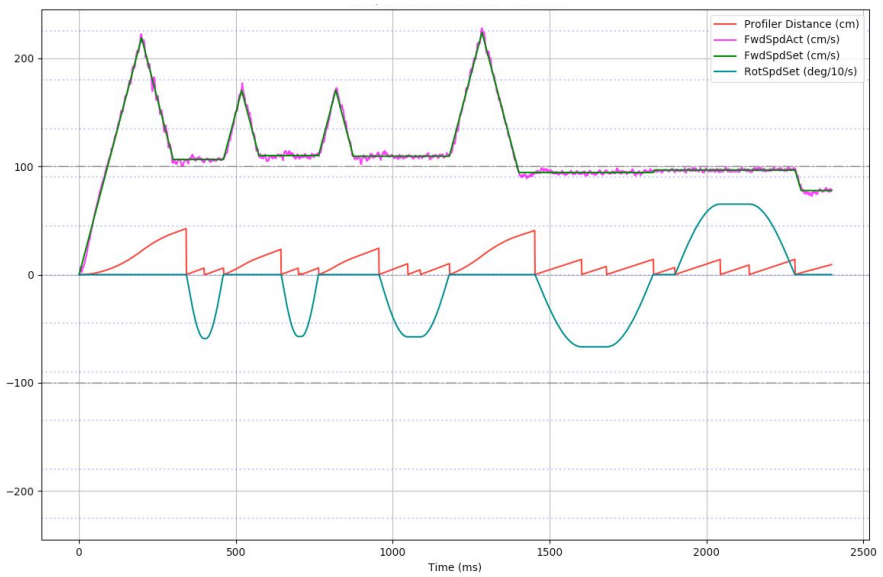


Figura 24: Gráfico de dados obtidos.

- **Giroscópio e acelerômetro**

  De maneira análoga a função velocidade real, esta opção foi criada no intuito de receber informações referentes a um sensor de movimentos. Desta forma, pode-se verificar o funcionamento do sensor e analisar os dados referentes à aceleração e velocidade angular.

- **Tomada de tempo**

  Esta função mostra o tempo que foi necessário para o robô percorrer toda a pista. Seu funcionamento básico é receber a informação de tempo que foi contabilizada no microcontrolador do próprio seguidor de linha e mostrar na interface. Foi escolhido que o próprio microcontrolador fizesse a cronometragem de tempo para que a medida fosse mais precisa.

Observações:

Para um bom funcionamento de um sistema como este, é necessário que haja uma compatibilidade entre o algoritmo gravado no robô e o algoritmo por trás da interface computadorizada. À medida que as funções vão aumentando a complexidade de ambos os algoritmos também aumenta, tornando o projeto mais suscetível a alguns erros difíceis de serem detectados.

Outro ponto a se observar é a transferência de dados entre o robô e a interface. Foram enfrentados alguns problemas com perdas de dados, fato que fazia com que as

informações fossem interpretadas de maneira incorreta. Ainda não se sabe ao certo se estes problemas são causados pelo modo com que são enviadas os dados, o modo com que a interface capta estes dados ou na transferência em si realizada pelo módulo bluetooth.

Para minimizar estes problemas, todas as funções da interface têm o seguinte funcionamento: O robô executa o comando e salva todas as informações em sua própria memória, após os dados serem armazenados, só ocorre a transferência de dados com o comando do operador. Desta forma, evita-se problemas de má conexão devido a distâncias.

Outra medida tomada foi o espaço de tempo entre o envio de uma informação e outra. Foi acrescentado um *delay* de alguns milissegundos entre envio de dados, fato que diminuiu consideravelmente as falhas que ocorriam quando se era necessário enviar muitas informações.

5.4 RUN

Com todas as informações minimamente necessárias, é possível otimizar o modo em que o seguidor corre o percurso. A idéia principal é que o robô utilize a velocidade máxima possível em cada trecho e tenha a capacidade de acelerar e desacelerar a tempo de alcançar a velocidade dos próximos trechos.

Desta forma, Ártemis e Nêmesis utilizam um algoritmo baseado em PID e na equação de Torricelli mostrada abaixo:

$$d = \frac{v^2 - vo^2}{2a}$$

Onde: v é a velocidade atual

vo é a velocidade final que se deseja alcançar

a é a aceleração desenvolvida

d é a distância necessária para atingir a velocidade vo

Basicamente o algoritmo funciona da seguinte forma:

- É definido um valor de aceleração ‘a’. Este valor deve bater com valores reais, de forma com que esta aceleração não faça as rodas deslizarem. De modo bem geral, caso o robô não utilize elementos que aumentem a normal, o valor de aceleração máximo pode ser calculado como :

$a = \mu.g$ Onde: μ é o coeficiente de atrito estático

g é a aceleração da gravidade

Que fique claro que esta etapa não necessariamente precisa ser executada a cada rotina do PID;

- Os valores de velocidade atual '***v***', velocidade final '***v0***', velocidade máxima permitida no trecho '***TopSpeed***', distância já percorrida '***DistAtual***' e distância total do trecho '***DistTotal***' são atualizados;
- É feito o cálculo da distância '***DistFaltando***' que falta para o robô atingir o próximo trecho fazendo a subtração das variáveis ***DistTotal*** e ***DistAtual;***
- É feito o cálculo da distância necessária '***DistNecessaria***' para que o robô consiga atingir a velocidade final ***vo*** com a determinada aceleração ou desaceleração utilizando-se a equação de Torricelli :

$$\textbf{DistNecessaria} = \frac{(v.v)-(v0.v0)}{2.a}$$

Se o valor de *DistNecessaria* for **maior** que o valor de *DistFaltando,* significa que ainda não é necessário que o robô desacelere, e então é definida a velocidade TopSpeed é preparada para o controlador.

Caso o valor de *DistNecessaria* for **menor** que o valor de *DistFaltando,* é definida a velocidade *v0* preparada para o controlador.

Segue abaixo um código de exemplo do algoritmo acima:

```
void run () {
        //Atualiza velocidade máxima do trecho
        TopSpeed = Velocidades[Trecho];

        //Atualiza velocidade final
        v0 = Velocidades[Trecho+1];

        //Atualiza velocidade atual
        v = getSpeed();

        //Atualiza distância total e distância faltando
        DistTotal = getEncoderDist();
        DistFaltando = DistTotal - DistFaltando;
```

```
            //Calcula-se a distância necessária e atualiza variável Speed
            DistNecessaria = (((v*v) - (v0*v0))/(2*a));
             if(DistNecessaria<0){DistNecessaria=0;} //Evita que a dist de uma valor negativo

             if(DistNecessaria<DistFaltando){
                    Speed = TopSpeed;
             }
             else{
                    Speed = v0;
             }
            //Evitar que velocidade setada seja maior que a permitida no trecho
            if(Speed>TopSpeed)Speed = TopSpeed;
}           //Fim void run
```

Com o código de exemplo acima e os ajustes necessário, Ártemis e Nêmesis se tornaram multi campeões nacionais utilizando o mesmo hardware da grande maioria. É necessário pensar sempre além e buscar constantemente aprimoramento de código e hardware para se obter o projeto mais competitivo possível. Não fique limitado a apenas às informações contidas neste relatório relatório!

Figura 25: Algumas das premiações conseguidas até o ano de 2019.

6. REFERÊNCIAS

https://kleberufu.wixsite.com/micromousebrasil/single-post/2015/07/27/Controlador
Micromouse Brasil

http://micromouseusa.com/?page_id=1342
Project Futura

https://docs.google.com/viewer?a=v&pid=sites&srcid=ZGVmYXVsdGRvbWFpbnxuZ2JlbmdraWF0ZG9jfGd4OjM0YzM0Y2EyYThmNjI5MTc
Artigo Ng Biang Kiat