UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ
DEPARTAMENTO ACADÊMICO DE INFORMÁTICA
CURSO DE ENGENHARIA DE COMPUTAÇÃO

WILLIAN AMERICANO LOPES

PROJETO E IMPLEMENTAÇÃO DE ROBÔ AUTÔNOMO SEGUIDOR DE LINHA

TRABALHO DE CONCLUSÃO DE CURSO

PATO BRANCO
2017

WILLIAN AMERICANO LOPES

PROJETO E IMPLEMENTAÇÃO DE ROBÔ AUTÔNOMO SEGUIDOR DE LINHA

Trabalho de Conclusão de Curso de graduação, apresentado à disciplina de Trabalho de Conclusão de Curso, do Curso Superior de Engenharia de Computação do Departamento Acadêmico de Informática - DAINF - da Universidade Tecnológica Federal do Paraná - UTFPR, como requisito parcial para obtenção do título de "Engenheiro de Computação ".

Orientador: Prof. Dr. Fábio Favarim
Co-orientador: Prof. Dr. Eng. César Rafael Claure Torrico

PATO BRANCO
2017

Ministério da Educação
Universidade Tecnológica Federal do Paraná
Câmpus Pato Branco
Departamento Acadêmico de Informática
Curso de Engenharia de Computação

UTFPR
UNIVERSIDADE TECNOLÓGICA FEDERAL DO PARANÁ

TERMO DE APROVAÇÃO

Às 10 horas e 20 minutos do dia 05 de dezembro de 2017, na sala V003, da Universidade Tecnológica Federal do Paraná, Câmpus Pato Branco, reuniu-se a banca examinadora composta pelos professores Fábio Favarim (orientador), Cesar Rafael Claure Torrico (coorientador), Everton Luiz de Aguiar e Kathya Silvia Collazos Linares para avaliar o trabalho de conclusão de curso com o título **Projeto e implementação de robô autônomo seguidor de linha**, do aluno **Willian Americano Lopes**, matrícula 01260871, do curso de Engenharia de Computação. Após a apresentação o candidato foi arguido pela banca examinadora. Em seguida foi realizada a deliberação pela banca examinadora que considerou o trabalho aprovado.

Fábio Favarim Orientador (UTFPR)	Cesar Rafael Claure Torrico Coorientador(UTFPR)
Everton Luiz de Aguiar (UTFPR)	Kathya Silvia Collazos Linares (UTFPR)
Profa. Beatriz Terezinha Borsoi Coordenador de TCC	Prof. Pablo Gauterio Cavalcanti Coordenador do Curso de Engenharia de Computação

A Folha de Aprovação assinada encontra-se na Coordenação do Curso.

Dedico este trabalho à meus pais, Antonio D. Lopes e Rosani A. A. Lopes, e ao meu irmão, Diego A. Lopes, os quais sem eles, não seria possível o desenvolvimento deste projeto, bem de como todas as conquistas que já tive.

AGRADECIMENTOS

Agradeço aos meus pais e ao meu irmão, que me deram suporte por tantos anos, suporte esse que foi muito mais do que financeiro, mas o de afeto, carinho e atenção, tendo sido a minha sustenção nessa trajetória.

Gostaria de manifestar a minha imensa gratidão aos professores orientadores deste trabalho, os professores Dr. Fábio Favarim e Dr. César Rafael Claure Torrico, que sempre tiveram paciência e estiveram a disposição para sanar as minhas dúvidas. Sem vocês, este trabalho também não poderia ser concluído.

Aos membros da banca avaliadora, os professores Dra. Kathya Silvia Collazos Linares e Me. Everton Luiz de Aguiar, que pelas críticas e conselhos, contribuíram para que este trabalho se tornasse o que ele é hoje.

Agradeço a Universidade Tecnológica Federal do Paraná, por todo o conhecimento adquirido nesses anos de graduação, por me prover educação gratuita e de qualidade, em que tive muitas dificuldades mas também muitos aprendizados, além dos cursos que obtive. Também agradeço a Fundação Araucária, pela bolsa que recebi por um projeto de extensão, o que me permitiu crescer tanto academicamente quanto profissionalmente.

Agradeço a outros dois profissionais da UTFPR que foram essenciais para que este trabalho pudesse ser desenvolvido: A professora Beatriz Terezinha Borsoi, do DAINF, e o técnico laboratorista do DAELE, Célio Antônio Degaraes.

Agradeço a meus amigos e colegas que conheci em Pato Branco, vocês foram a minha família nesta cidade. Muito obrigado pelas conversas, pelas risadas, mas também pelas dificuldades que vocês me ajudaram a superar. Espero que nossa amizade continue depois da graduação.

Agradeço a todos os desenvolvedores de tecnologias *open-source*, os quais facilitaram o desenvolvimento deste trabalho, assim como muitos outros projetos que já desenvolvi ou que ainda farei (agradecimento especial a Leslie Lamport, por essa ferramenta sensacional chamada LaTeX).

Por último e não menos importante, agradeço a Deus, que sempre está me acompanhando, por toda a sabedoria e conhecimento que me permitiram passar por todos os obstáculos e desafios e aprender com eles.

"Se enxerguei mais longe, foi porque
me apoiei nos ombros de gigantes"

Sir Isaac Newton

RESUMO

LOPES, Willian Americano. Projeto e Implementação de robô autônomo seguidor de linha. 114 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

Esse trabalho tem por objetivo o projeto e a implementação de um robô autônomo seguidor de linha, utilizando controle híbrido, composto por um controlador de tempo contínuo e por um orientado a eventos discretos. Após feita uma revisão do estado da arte, a qual cobriu mapeamento de pista, estrutura de robôs móveis, métodos de controle contínuo, como o PD (Proporcional-Derivativo), que foi implementado nesse trabalho, bem como de Sistemas a Eventos Discretos, o SED, o qual foi projetado utilizando autômato de Moore. No controlador de SED, de acordo com o respectivo estado, um vetor, responsável pelo mapeamento da pista, recebe informações como o começo e o fim de curva, bem como da pista. Como resultados, foi obtida a planta do veículo, foram feitos testes da resposta de sensores de refletância e testes de desempenho dos robôs desenvolvidos, o que aconteceu em competições e na pista disponível na universidade, a qual está de acordo com as normas estabelecidas pela Robocore.

Palavras-chave: Robôs móveis. Robô seguidor de linha. Controlador híbrido. Sistemas microcontrolados.

ABSTRACT

LOPES, Willian Americano. Project and implementation of a line follower autonomous robot. 114 f. Trabalho de Conclusão de Curso – Curso de Engenharia de Computação, Universidade Tecnológica Federal do Paraná. Pato Branco, 2017.

The goal of this work was to design and prototype an autonomous line follower robot using hybrid control, which is composed by of a continuous time controller and a discrete time controller. After a review of the state of art, which had covered path mapping, mobile robot structure, continuous control methods such as PD (Proportional-Derivative), which was implemented in this work, as well as Discrete Event Systems, DES, which was designed by Moore automaton. In the DES controller, according to the respective state, a vector responsible for mapping the path receives informations about the beginning and the ending of a curve, as well as the path. As results, the vehicle's plant was obtained, performance tests were performed on the response of reflectance sensors and on the developed robots, that happened in competitions and on the runway available at the university, which is in accordance with the standards established by Robocore.

Keywords: Mobile robots. Line follower robot. Hybrid controller. Microcontrolled systems.

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS E ACRÔNIMOS

ACK	*Acknowledgement* (Confirmação de reconhecimento)
AFD	Autômato Finito Determinístico
ARM	*Advanced RISC Machine* (Máquina *RISC* Avançada)
BJT	*Bipolar Junction Transistor* (Transistor de Junção Bipolar)
CA	Corrente Alternada
CC	Corrente Contínua
DMA	*Direct Memory Access* ou Acesso Direto à Memória
DMIPS	*Dhrystone Million Instructions Per Second* (Milhões de Instruções por Segundo Dhrystone)
EDA	*Electronic Design Automation* (Desenho Eletrônico Automatizado)
FPU	*Floating-Point Unit* (Unidade de Ponto Flutuante)
HPCB	*High-Power Carbon Brush* (Escova de Carbono de Alta Potência)
IEEE	*Institute of Electrical and Electronics Engineers* (Instituto dos Engenheiros Eletricistas e Eletrônicos)
LED	*Light Emitting Diode* (Diodo Emissor de Luz)
Li-Po	Lítio-Polímero
LS3	*Legged Squad Support Systems* (Sistema de Suporte a Esquadrão com Pernas)
MDF	*Medium-Density Fiberboard* (Painel de Fibras de Média Densidade)
MOSFET	*Metal Oxide Semiconductor Field Effect Transistor* (Transistor de Efeito de Campo de Semicondutor Óxido Metal)
NASA	*National Aeronautics and Space Administration* (Administração Nacional Aeronáutica e Espacial)
NiCad	Níquel-Cádmio
NiMH	Hidreto Metálico de Níquel
P	Proporcional
PCI	Placa de Circuito Impresso
PD	Proporcional-Derivativo
PI	Proporcional-Integral
PID	Proporcional-Integral-Derivativo
PWM	*Pulse Width Modulation* (Modulação por Largura de Pulso)
RIA	*Robotic Industries Association* (Associação das Indústrias de Robótica)
RISC	*Reduced Instruction Computer Set* (Computador com Conjunto Reduzido de Instruções)
RPM	*Revolutions Per Minute* (Revoluções por minuto)
RUR	*Rossum's Universal Robots* (Robôs Universais de Rossum)
SED	Sistema a Eventos Discretos
SRAM	*Static Random Access Memory* (Memória Estática de Acesso Aleatório)

TCC	Trabalho de Conclusão de Curso
T_s	Tempo de assentamento
UART	*Universal Asynchronous Receiver Transmitter* (Transmissor-Receptor Assíncrono Universal)
UFRJ	Universidade Federal do Rio de Janeiro
USB	*Universal Serial Bus* (Barramento Serial Universal)
UTFPR	Universidade Tecnológica Federal do Paraná

SUMÁRIO

1 INTRODUÇÃO

Este capítulo está dividido da seguinte forma: A Seção 1.1 apresenta uma visão geral do tema abordado, a Seção 1.2 trata da justificativa desta pesquisa, a Seção 1.3 trata as limitações do projeto e a Seção 1.4 apresenta os objetivos a serem alcançados neste trabalho.

1.1 CONSIDERAÇÕES INICIAIS

A robótica é uma das áreas mais promissoras da engenharia, tendo aplicabilidade em várias áreas: de médicas a aeroespaciais. Atualmente é difícil encontrar atividades industriais que não possuam um sistema robótico ou automatizado, seja total ou parcial. Uma das aplicações da robótica, os robôs móveis vem se destacando na atualidade, dotando a estes capacidades[1] de autonomia[2] e mobilidade, características necessárias para que que o veículo possa se deslocar nos mais diversos ambientes de operação.

Devido à ampla aplicabilidade e utilidade que os robôs apresentam foram criadas competições para robôs móveis, as quais visam estimular e contribuir com a pesquisa na robótica, tais como:

- A Robogames (2016) que também é conhecida como "Olimpíada dos robôs", em que são disputadas mais de cinquenta categorias;
- A VEX Worlds (2016), a maior competição de robótica do mundo, que contou com 1075 times e mais de 15.000 participantes em sua última edição (RECORDS, 2016);
- A RoboCup (2016), que em 2016 foi sediada em Leipzig, Alemanha;
- A Robocore (2016a), com eventos realizados no Brasil.

A WinterChallenge, realizada anualmente pela Robocore, em São Paulo, é uma das maiores competições de robótica móvel da América Latina, contando com a participação de vários países e teve mais de mil competidores e cerca de quinhentos robôs na edição de 2016 (MAUÁ, 2016).

Uma das categorias disputadas é a dos seguidores de linha, na qual os robôs devem seguir, de maneira autônoma, um trajeto que é determinado por uma linha. Nessa categoria,

[1] Capacidade, no contexto deste documento, se refere à habilidade ou aptidão do veículo.

[2] Autonomia é a distância na qual um dispositivo pode percorrer de forma independente, sem auxílio humano.

se destacam os competidores do Japão, na competição Robogames, e do México, na Robocore, estes obtendo os três primeiros lugares na competição WinterChallenge na categoria Seguidor de Linha - Pro (ROBOCORE, 2016b). Competindo nesta mesma categoria, a equipe Patobots, da Universidade Tecnológica Federal do Paraná UTFPR - Câmpus Pato Branco, conquistou o 5º e 6º lugar, com os robôs *Alpha project* e *Robbie 3.0*, respectivamente.

Com base nesse contexto, este trabalho propõe a construção de um protótipo de robô seguidor de linha e o seu respectivo controle, visando melhorar o seu desempenho para participar de competições de robótica, tais como a Robocore.

O desenvolvimento deste trabalho contribuirá com a pesquisa que é feita na Universidade Tecnológica Federal do Paraná - Câmpus Pato Branco, na área de robótica móvel, em que já foram produzidos os trabalhos de Guadagnin (2014) e Petry (2016), os quais servem de base para este trabalho.

1.2 JUSTIFICATIVA

O trabalho de Guadagnin (2014), utilizou-se de controle híbrido, o qual integrou dinâmicas de controle a eventos discretos para detecção de marcas laterais e controle de tempo contínuo, tal qual o controlador Proporcional-Integral-Derivativo (PID) para controle de posição sobre a linha do percurso. Segundo Guadagnin (2014), o robô funcionou de acordo com o esperado para um percurso dentro das normas da Robocore, tendo alguns problemas relacionados à detecção das marcas laterais quando a pista estava com uma inclinação maior que 5º (graus).

O trabalho de Petry (2016) desenvolveu um robô híbrido, realizando um estudo sobre os resultados obtidos por controladores PID e *Fuzzy* (lógica difusa), sendo que o controlador PID apresentou melhor desempenho. Devido às dificuldades encontradas, a comparação dos métodos foi realizada no robô 3pi, da Pololu[3], em que o autor conseguiu uma velocidade de 1 m/s em retas.

Com base nos trabalhos de Guadagnin (2014) e Petry (2016), é proposta a modelagem de um novo *hardware*, utilizando-se de técnicas de controle híbrido, o qual combina dinâmicas orientadas a eventos discretos e orientadas a tempo (CASSANDRAS; LAFORTUNE, 2008), e a implementação de um mapeamento de pista.Também propõe-se a utilização de um microcontrolador com Unidade de Ponto Flutuante (FPU), que pode facilitar a implementação de técnicas mais complexas e que exigem maior poder computacional e memória.

[3]Pololu é uma empresa norte-americana que fabrica e revende produtos eletrônicos na área de robótica.

1.3 LIMITAÇÕES DO TRABALHO

Este trabalho apresenta as seguintes limitações:

1. Devido à complexidade que é o projeto e a implementação de um robô, este trabalho não se preocupará com o desenvolvimento mecânico do dispositivo, mas se atentará a parte eletrônica e de *software*;
2. O projeto do robô seguidor de linha se aterá ao funcionamento em pistas que seguem as normas da Robocore, podendo apresentar restrições de comportamento, e até mesmo não funcionar, caso a pista não esteja no padrão estabelecido.

1.4 OBJETIVOS

1.4.1 OBJETIVO GERAL

Projetar e implementar um protótipo de um robô seguidor de linha, que seja autônomo, através da utilização de controle híbrido, aperfeiçoando as técnicas desenvolvidas por Petry (2016).

1.4.2 OBJETIVOS ESPECÍFICOS

- Projetar o condicionamento de sinais necessário para os dispositivos a serem utilizados, permitindo uma boa precisão na leitura dos sensores;
- Projetar e confeccionar a estrutura do protótipo, visando atender as dimensões especificadas pela Robocore;
- Implementar um controlador de Sistemas a Eventos Discretos, para que seja possível tratar de maneira precisa as marcações laterais da pista;
- Modelar a função de transferência do robô;
- Implementar um controlador para manter o robô sobre a linha na pista;
- Realizar testes com o protótipo em pistas que sigam as normas da Robocore;
- Fazer o mapeamento do percurso com um *encoder*;
- Comparar os resultados obtidos com o de Petry (2016).

2 REFERENCIAL TEÓRICO

Neste capítulo são apresentados os principais conceitos teóricos relacionados ao desenvolvimento deste trabalho.

2.1 ROBÓTICA

A *Robotic Industries Association* (RIA), entidade norte-americana responsável pela indústria de robótica, define um robô industrial como,

> Um manipulador multipropósito reprogramável, controlado automaticamente, programado em três ou mais eixos, os quais podem ser fixos em um lugar ou móveis para aplicações em automação industrial.(ANSI/RIA.R15.06-1999, 2010).

Secchi (2012) classifica os robôs em três tipos:

- Industriais: São formados por estruturas mecânicas articuladas, as quais se movem conforme as ordens de um sistema de controle, normalmente um microcontrolador;
- Médicos: Também conhecidos como de cooperação ou reabilitação, são os utilizados em cirurgias de alta complexidade e precisão, assim como as próteses inteligentes, que visam manter a aparência e funcionalidade do membro de pessoas com deficiência.;
- Móveis: São plataformas mecânicas, que se locomovem em um certo ambiente e apresentam certa autonomia. São empregados principalmente em tarefas em que se tem risco à vida humana, como na manutenção de reatores nucleares ou exploração de minérios, mas também podem ser aplicados na agricultura e no transporte de cargas. O veículo que será desenvolvido neste trabalho será desta categoria.

2.1.1 ROBÓTICA MÓVEL

Mesmo que os robôs industriais apresentem alta precisão e velocidade, estes possuem uma grande desvantagem, que é a falta de mobilidade, já que são fixos e não são capazes de se locomoverem pelo ambiente. Algumas atividades não seriam realizadas sem a utilização dos robôs móveis, como é o caso da *Mars Pathfinder*, missão exploratória da NASA para o

reconhecimento da atmosfera de Marte. O veículo Sojourner (Figura 1), que foi utilizado nesta missão, explorou o território marciano por oitenta e três dias, tirando fotografias e realizando medições do ambiente (NASA, 1997b).

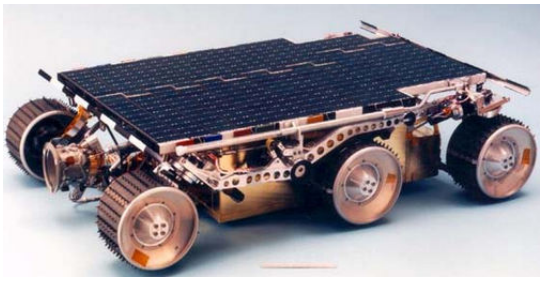


Figura 1 – Veículo exploratório Sojourner
Fonte: (NASA, 1997a)

A robótica móvel lida com o controle de veículos autônomos e semi-autônomos, tendo ênfase em problemas relacionados com o espaço em larga escala, que são regiões com espaços consideravelmente maiores que as observáveis pelo ponto de visão do robô. O espaço em larga escala é de extrema importância para um robô móvel, visto que afeta o seu movimento, compreensão e raciocínio nesta área, sendo estes três subproblemas essenciais para este campo de pesquisa (DUDEK; JENKIN, 2010).

Siegwart et al. (2011) classifica os robôs móveis em duas categorias relacionadas à locomoção:

- Robôs terrestres (*legged robots*): Tem como vantagem a manipulação de objetos e a locomoção em terrenos acidentados, mas, tem alta complexidade mecânica e energética. A Figura 2 mostra o *Legged Squad Support Systems* (LS3) da Boston Dynamics, projetado para atuar nos mesmos terrenos acidentados utilizados por *marines*[4] e soldados norte-americanos, ajudando a carregar equipamentos (DYNAMICS, 2016).

Figura 2 – Robô terrestre LS3
Fonte: (DYNAMICS, 2016)

[4]Os *marines* são fuzileiros navais das forças armadas dos Estados Unidos da América

- Robôs com rodas (*wheeled robots*): É o tipo de locomoção mais utilizado em robôs móveis e veículos. Normalmente o equilíbrio não é levado em consideração, visto que na maior parte dos projetos as rodas são consideradas em contato com o solo o tempo todo. O robô Sojourner da Figura 1 é um exemplo de robô com rodas.

2.2 SISTEMAS DE CONTROLE PARA ROBÔS MÓVEIS

Para que o robô seja autônomo, é necessário que este apresente uma resposta desejada para as mais diversas situações. Para tanto se utiliza de um sistema de controle, que consiste em subsistemas e processos, conhecidos como plantas, dos quais se obtém uma saída com desempenho desejado para uma dada entrada (NISE, 2012). Na Figura 3 é mostrado o diagrama de blocos[5] de um sistema de controle de tempo contínuo, o qual é composto pela planta e pelo controlador, que são os processos do sistema, e pelos sinais de Resposta Desejada, Sinal de Controle e Saída do Sistema (ARAÚJO, 2007).

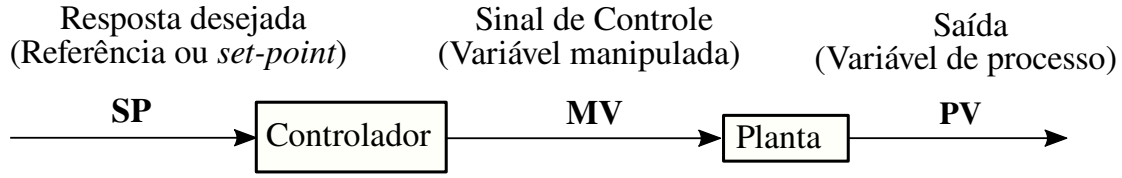


Figura 3 – Descrição de um sistema de controle a malha aberta
Fonte: Adaptado de (ARAÚJO, 2007, p.2).

Para Nise (2012) os sistemas de controle podem ser classificados em duas configurações principais:

- Controle por malha aberta (*Open loop control*): É a configuração mais simples, na qual o sinal de saída não exerce nenhuma ação de controle no sistema (OGATA, 2010). Desta forma, um sistema de controle de malha aberta não é medido e nem comparado com a entrada, como pode ser visto na Figura 3, não sendo capaz de compensar distúrbios que possam ser adicionados ao sistema (NISE, 2012).

- Controle por Malha fechada (*Closed loop control*): Nesta configuração, a saída ou a resposta influencia a entrada do sistema (ARAÚJO, 2007). Logo após a Resposta Desejada, na Figura 4, tem-se um ponto de soma, em que será feita a soma algébrica dos sinais associados. Segundo Ogata (2010), o sinal de erro atuante, que é a diferença

[5]Diagrama de blocos é uma representação gráfica do sistema que mostra o fluxo de sinais que ocorre entre os componentes deste.

entre o sinal de entrada e o sinal de realimentação (o qual é calculado logo após o ponto de soma, como pode ser visto na Figura 4), realimenta o controlador, o qual tende a minimizar o erro e deixar a saída com o valor desejado.

2.2.1 AÇÕES BÁSICAS DE CONTROLE

Conforme Ogata (2010), um controlador automático compara o valor de saída da planta com a entrada do sistema, determinando o desvio e produzindo um sinal de controle que reduzirá este desvio a um valor pequeno. É chamada de ação de controle a maneira pela qual o controlador produz o sinal de controle. Na Figura 4 é mostrado o diagrama de blocos de um sistema de controle, composto por um controlador automático, um atuador, uma planta e um sensor, o qual é o elemento de medição. Os atuadores são dispositivos capazes de alterar o estado do sistema controlado com base em sinais de controle. Os motores elétricos são um dos principais atuadores encontrados. O sensor é um elemento que converte a variável de saída em uma outra variável pertinente, que possa ser manipulada pelo sistema.

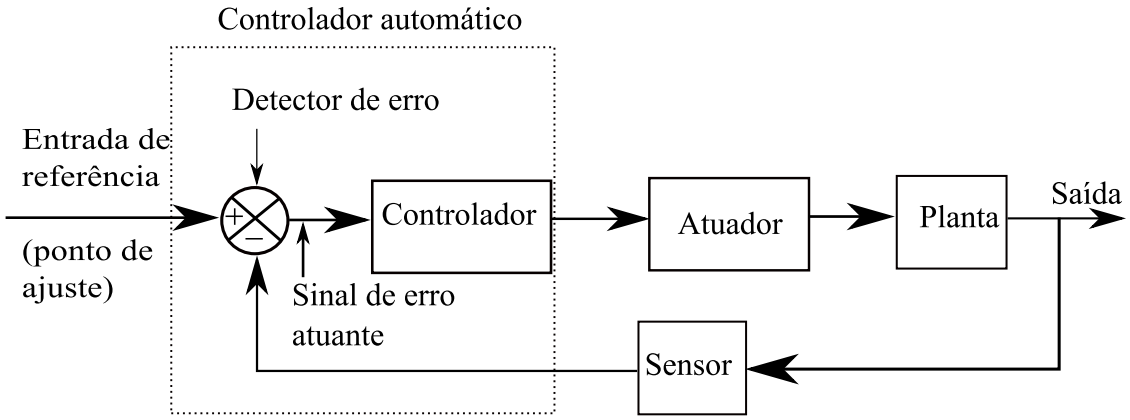


Figura 4 – Diagrama de blocos de um sistema de controle em malha fechada
Fonte: Adaptado de (OGATA, 2010, p.18).

Ogata (2010) classifica os controladores de acordo com as suas ações de controle:

- Controlador de duas posições ou *on-off*;
- Controlador Proporcional (P);
- Controlador Integral;
- Controlador Proporcional-Integral (PI);
- Controlador Proporcional-Derivativo (PD);
- Controlador Proporcional-Integral-Derivativo (PID).

2.2.1.1 CONTROLADOR *ON-OFF*

Em controladores de duas posições ou *on-off*, o elemento atuante tem somente duas posições, as quais são fixas, geralmente sendo *on* (ligado) e *off* (desligado) (OGATA, 2010). Tem grande aplicabilidade em sistemas de controle industriais e domésticos, devido ao baixo custo e simplicidade de implementação.

Considerando-se o sinal de saída do controlador $u(t)$ e o sinal de erro atuante $e(t)$, o sinal $u(t)$ apresenta um valor máximo ou mínimo, caso o erro atuante seja negativo ou positivo, respectivamente. Assim, tem-se que:

$$u(t) = U_1, \quad para\ e(t) > 0, \tag{1}$$

$$u(t) = U_2, \quad para\ e(t) < 0, \tag{2}$$

em que U_1 e U_2 são constantes.

Também pode ser encontrado o controlador *on-off* com histerese (*differential gap* ou lacuna de diferença), conforme visto na Figura 5, que acarreta a saída $u(t)$ a manter o seu presente valor até que o erro atuante tenha mantido o seu valor ligeiramente acima de zero (OGATA, 2010). A histerese muitas vezes evita um número excessivo de chaveamentos, os quais podem reduzir a vida útil dos componentes a serem controlados, tais como motores.
Em problemas de controle mais complexos, se faz necessária a utilização de outros controladores, devido ao *on-off* apresentar muitas oscilações e *offset*[6].

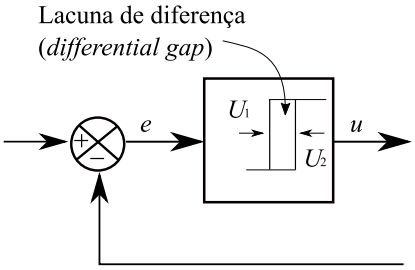


Figura 5 – Diagrama de blocos de um controlador *on-off* com histerese
Fonte: Adaptado de (OGATA, 2010, p.23).

2.2.1.2 CONTROLADOR PROPORCIONAL

Na ação de controle proporcional, a relação entre a saída do controlador $u(t)$ e o sinal de erro $e(t)$ é apenas um ganho:

$$u(t) = K_p e(t), \tag{3}$$

[6] *Offset* é um erro entre o valor desejado e o valor obtido.

em que K_p é o ganho do controlador proporcional. Para Araújo (2007) quanto maior o ganho, menor será o erro $e(t)$. No entanto o sistema pode perder estabilidade devido à diminuição do tempo de acomodação[7].

2.2.1.3 CONTROLADOR INTEGRAL

Segundo Ogata (2010), na ação de controle integral, a saída do controlador é diretamente proporcional à integral do sinal do erro o valor da saída $u(t)$ é diretamente proporcional à integral do sinal de erro atuante $e(t)$:

$$u(t) = K_i \int_0^t e(t)dt, \tag{4}$$

em que K_i é a constante de ganho integral.

2.2.1.4 CONTROLADOR PROPORCIONAL-INTEGRAL (PI)

Para Nise (2012), um controlador Proporcional-Integral pode ser chamado de compensador integral ideal, visto que as ações de controle Proporcional e Integral alimentam o erro e a integral do erro para a planta, respectivamente, desta forma tendem a eliminar o erro em regime permanente[8]. A ação de controle é definida por:

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt, \tag{5}$$

em que T_i é chamado de tempo integrativo.

2.2.1.5 CONTROLADOR PROPORCIONAL-DERIVATIVO (PD)

Segundo Araújo (2007), as ações proporcional e derivativa deste controlador contribuem com o regime transitório, tendendo a aumentar a estabilidade do sistema e reduzir o tempo de acomodação. A ação de controle deste controlador é definida por:

$$u(t) = K_p e(t) + K_p T_d \frac{de(t)}{dt} \tag{6}$$

em que T_d é chamado de tempo derivativo (OGATA, 2010).

[7] Tempo de acomodação (*settling time*) T_s é o tempo para que a curva de resposta alcance valores em uma faixa (2% a 5%) em relação ao valor final, permanecendo nesta faixa interminavelmente.

[8] A resposta temporal de um sistema consiste em duas partes: da resposta transitória e da resposta estacionária ou em regime permanente

2.2.1.6 CONTROLADOR PROPORCIONAL-INTEGRAL-DERIVATIVO (PID)

O controlador Proporcional-Integral-Derivativo reúne três ações de controle: A Proporcional, a Integral e Derivativa, atuando na melhoria do regime permanente e da resposta transitória. A equação do controlador é dada por:

$$u(t) = K_p e(t) + \frac{K_p}{T_i} \int_0^t e(t)dt + K_p T_d \frac{de(t)}{dt} \tag{7}$$

em que K_p é o ganho proporcional, T_i é o tempo integrativo e T_d é o tempo derivativo (OGATA, 2010).

2.3 SISTEMAS A EVENTOS DISCRETOS

Para Cassandras e Lafortune (2008), os sistemas podem ser classificados em duas categorias quanto à natureza do espaço de estados selecionado:

- Estado contínuo: O espaço de estados X é uma continuidade, compreendendo todos os vetores n-dimensionais de números reais ou complexos, eventualmente;
- Estado discreto: O espaço de estados é um conjunto discreto. As variáveis de estado deslocam-se em pontos discretos no tempo, de um valor estado discreto para outro.

Teixeira (2013) diz que a modelagem computacional de um sistema pode ser estruturada em dois fundamentos:

- Estado, que determina o *status* do sistema em determinada situação;
- Transição de estados, o qual caracteriza o crescimento do sistema.

Alguns sistemas são mapeados continuamente e suas estruturas de transição são regidas pelo tempo, enquanto que em outros sistemas seus estados podem não ser contínuos e as transições não dependem do tempo, mas de eventos instantâneos e assíncronos, como o processamento de um dado em um equipamento computacional, em que o conjunto de estados é discreto e as transições não são feitas por tempo, mas por eventos como *clock* e interrupções (TEIXEIRA, 2013).

Torrico (2003) define um Sistema a Eventos Discretos (SED) como um sistema dinâmico a estado discreto que evolui à proporção assíncrona de eventos. Deste modo, os SEDs tem como características o espaço de estados ser discreto e o mecanismo de transição de estados ser dirigido por eventos.

Alguns exemplos dos Sistemas a Eventos Discretos, conforme Cassandras e Lafortune (2008), são os Sistemas de Filas, os Sistemas de Computador, Sistemas de Comunicação e Sistemas de Manufatura.

2.3.1 LINGUAGENS E AUTÔMATOS

Uma das maneiras formais de se estudar o comportamento lógico de um SED é baseado nas teorias de linguagens e autômatos, que se dá pelo fato de qualquer Sistema a Eventos Discretos ter um conjunto implícito Σ associado a ele. Cassandras e Lafortune (2008) chama esse conjunto Σ de eventos como "alfabeto"e sequências com esses eventos são denominados de "palavras". Para demonstrar estes conceitos, o autor fala do exemplo de uma máquina que é ligada uma ou duas vezes ao dia, como um carro ou um computador pessoal, e deseja-se projetar um sistema que realize uma simples tarefa: Quando a máquina estiver ligada, esta primeiramente deve indicar que está ligada e depois reportar este fato (como "checar óleo", caso seja um carro). Neste caso, cada um destes sinais define um evento, enquanto todos os sinais possíveis que a máquina possa emitir são definidos como uma sequência de eventos, ou cadeias com elementos do alfabeto.

Cassandras e Lafortune (2008) define um autômato como um dispositivo apropriado para representar a linguagem de acordo com regras bem definidas. Caso o conjunto de estados do autômato seja finito e caso este seja determinístico, ou seja, que este não possua duas ou mais transições com o mesmo nome saindo de um estado, o autômato é chamado de Autômato Finito Determinístico (AFD).

Hopcroft et. al (2003) diz que em um autômato finito , os estados são representados por círculos, e as entradas, as quais são representadas por arcos, caracteriza influências externas sob o sistema. Duas extensões dos autômatos são os Autômatos de Mealy e de Moore, as quais são detalhadas a seguir.

2.3.1.1 AUTÔMATO DE MOORE

O modelo de Moore são autômatos com saídas atribuídas aos estados, em que há uma função de saída a qual especifica uma saída para cada estado (CASSANDRAS; LAFORTUNE, 2008). As ações são produzidas nos estados e a saída depende somente do estado atual. Um exemplo do Autômato de Moore pode ser visto na Figura 6, a qual consiste em um diagrama do funcionamento de uma válvula e seu sensor de fluxo, sendo que a saída associada a cada estado é mostrada em negrito. O primeiro estado (seta que aponta para um estado sem ter origem em outro) é o de *válvula fechada*, que apresenta a saída *SEM FLUXO*; ocorrendo a transição *Abre válvula_um giro* o estado muda para *Válvula parcialmente aberta* causando a ação de

FLUXO PARCIAL. O próximo estado, *válvula aberta*, tem a ação de *MÁXIMO FLUXO*, o que em caso de ocorrer uma saída de emergência (transição *emergência_desligar*) retorna para o estado inicial, no qual não terá mais fluxo da válvula (pela ação *SEM FLUXO*).

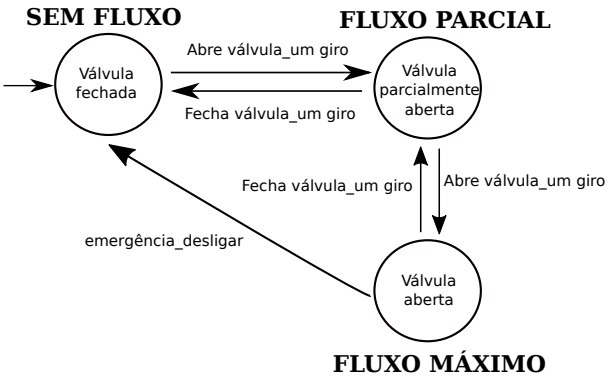


Figura 6 – Autômato de Moore
Fonte: Adaptado de (CASSANDRAS; LAFORTUNE, 2008, p.73)

2.3.1.2 AUTÔMATO DE MEALY

Diferentemente de Moore, no autômato de Mealy as ações de saída são atribuídas às transições de estados. Este modelo funciona da seguinte maneira: para uma transição e_i/e_o do estado x para o estado y: Quando o sistema estiver no estado x, o autômato receberá o evento de entrada e_i e expedirá o evento de saída e_o durante a transição para y (CASSANDRAS; LAFORTUNE, 2008). Conforme mostra a Figura 7, as funções de saída estão nas transições. Este exemplo mostra o fluxo de informações em uma rede de comunicação do tipo *half-duplex*, em que é possível enviar ou receber dados, mas não simultaneamente. O primeiro estado é o *Enviando pacote com rótulo 0*, em que podem acontecer as ações *reenviar pacote* e *espera* caso ocorram as transições *timeout* (*timeout* é quando expirou o tempo de entrega do pacote) e *recebe* ACK (ACK ou *Acknowledgement* significa a confirmação de recebimento ou envio do pacote), respectivamente. O comportamento da máquina de Mealy é o mesmo que de Moore, no entanto estes tem uma implementação diferente.

2.4 CONTROLE HÍBRIDO

Cassandras e Lafortune (2008) diz que um sistema híbrido é formado por dinâmicas discretas e contínuas. Branicky et al. (1998) diz que quase todos os sistemas de controle atuais, ao mesmo tempo em que são executados por linhas de código em um computador, também

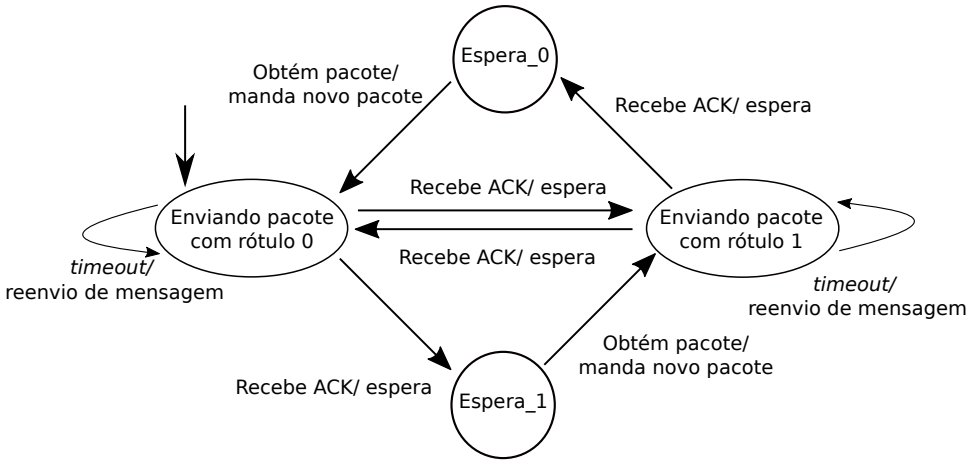


Figura 7 – Autômato de Mealy
Fonte: Adaptado de (CASSANDRAS; LAFORTUNE, 2008, p.73)

respondem comandos de variáveis contínuas. Desta forma, problemas modernos de controle provavelmente serão modelados sobre um sistema híbrido.

Dudek e Jenkin (2010) diz que o enquanto o controle em baixo nível precisa de resposta em tempo real a eventos externos, tendo natureza contínua, o controle de alto-nível geralmente é implementado como um conjunto de operações discretas ou tarefas que precisam ser cumpridas. Neste contexto, o controle em alto nível geralmente é chamado de controle de tarefa ou controle estratégico, enquanto o controle em baixo nível é conhecido como controle reativo ou controle de execução (DUDEK; JENKIN, 2010).

Na Figura 8 é mostrado o diagrama de blocos de uma arquitetura de controle híbrida. Em *Interface* acontece a "abstração"do sistema de controle contínuo, de baixo nível, para um SED, de alto nível. Isso acontece na *Interface* devido à informação dos sensores e das variáveis contínuas serem retransmitidas para o controlador supervisório, o qual ocorre na forma de eventos, na medida que os comandos do supervisório são transmitidos de volta para a *Interface* para gerar o sinal apropriado para acionar os atuadores (CASSANDRAS; LAFORTUNE, 2008).

2.5 ESTRUTURA DE UM ROBÔ MÓVEL

Dudek e Jenkin (2010) classifica os componentes físicos (*hardware*) dos robôs móveis em quatro partes:

- Locomoção: É como o robô se move pelo ambiente;

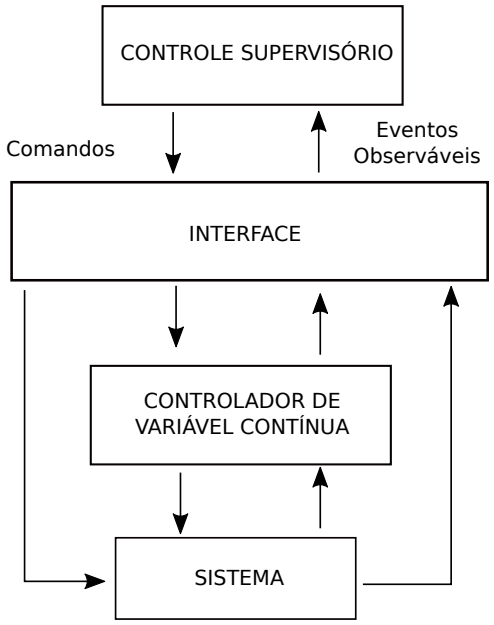


Figura 8 – Arquitetura de um sistema de controle híbrido
Fonte: Adaptado de (CASSANDRAS; LAFORTUNE, 2008, p.43)

- Percepção: É como o robô percebe propriedades do ambiente e dele mesmo;
- Raciocínio: É como o robô transforma a suas medidas em ações;
- Comunicação: É como que o robô se comunica com um agente externo.

Com suporte nessa divisão e no restante da literatura, tem-se os seguintes elementos necessários para o projeto de um robô móvel:

Bateria: É um componente essencial do sistema, visto que o robô móvel precisa se locomover pelo espaço e não pode ser dependente de uma fonte externa de energia elétrica. Dudek e Jenkin (2010) diz que baterias com bom desempenho e baixo custo são as baseadas em células de gel (*gell cell*), mas por serem pesadas não são utilizadas com veículos leves. Para estes, são utilizadas tecnologias baseadas em Íons de Lítio (*lithium-ion*), Níquel-Cádmio (NiCad) e as de Hidreto Metálico de Níquel (NiMH), as quais são originalmente projetadas para serem utilizadas em dispositivos sem-fio, como *laptops* e *smartphones*.

Motores: São os responsáveis por fazer com que o veículo "ande". Muitos robôs utilizam a bateria para acionar os motores de passo ou servomotores, os quais convertem energia elétrica em mecânica (Dudek e Jenkin (2010)). No entanto existem outros dispositivos, capazes de gerar energia mecânica em movimento, como os piezoelétricos e os pneumáticos.

- O motor de passo (*stepper motor*) tem a posição de seu eixo regulada pelo número de pulsos elétricos fornecidos às suas bobinas. Assim, ao se conhecer o número de pulsos nas chaves, é possível conhecer a posição instantânea do eixo do motor;

- O servomotor (*servo motor*) combina um motor elétrico de corrente contínua (CC) ou corrente alternada (CA) com um sensor de orientação de eixo (DUDEK; JENKIN, 2010). O controle dos servomotores é mais complexo que os motores de passo, sendo necessário utilizar um controlador de motor mais complexo, pois estes trabalham com parâmetros como posição, velocidade ou movimento de saída (DUDEK; JENKIN, 2010).

Controle de um servo motor: Podem ser utilizadas as abordagens de malha aberta e malha fechada. No controle de malha aberta, o valor desejado do eixo é utilizado para projetar um controlador oportuno, no entanto fica suscetível a distúrbios, pois não apresenta realimentação. No controle por malha fechada, a realimentação do sistema é utilizada para ajustar o movimento do dispositivo, sendo que um controlador amplamente utilizado é o PID (DUDEK; JENKIN, 2010).

Dispositivo de comunicação: Devido à necessidade em interagir com um operador humano, como no caso de reportar que uma tarefa foi concluída, Dudek e Jenkin (2010) relata que os robôs utilizam de um meio físico para a comunicação, como por cabo, ou por comunicação sem-fio (*wireless*), como o *bluetooth* (padrão sem-fio de curto alcance), 802.11 (padrão da IEEE para redes locais sem-fio) ou infravermelho.

Processamento: O processamento é um componente principal dos robôs móveis, tanto que Dudek e Jenkin (2010) discute em que lugar o processamento do veículo deve estar. O autor chega na conclusão que a melhor forma do processamento ser feito é a de separar as tarefas em *on-board* (dentro do veículo) e *off-board* (fora do veículo), responsáveis, respectivamente, pelas tarefas de tempo crítico e as que não são de tempo crítico. Este modelo tem como vantagens a redução no consumo de energia do dispositivo.

Sensores: Secchi (2012) diz que os robôs devem ser capazes de realizar três tarefas essenciais: estimar a sua posição e orientação (pose), manter o mapa do ambiente atualizado e identificar os possíveis obstáculos deste. Desta forma, para que o robô conheça o seu ambiente de trabalho e possa se adaptar a este, é necessário coletar informações e adequá-las ao sistema de controle. Assim, o controle de malha fechada, o qual foi abordado anteriormente, necessariamente precisa do sinal de sensores para realizar o *feedback* (realimentação) do sistema.

Dudek e Jenkin (2010) classifica os sensores em duas variantes: visuais e não-visuais. A seguir serão mostrados os principais sensores não-visuais dos robôs móveis. Os sensores visuais, devido à não terem aplicação neste trabalho, não serão abordados. Para mais informações sobre estes, consultar (DUDEK; JENKIN, 2010), (SIEGWART et al., 2011) e (SECCHI, 2012).

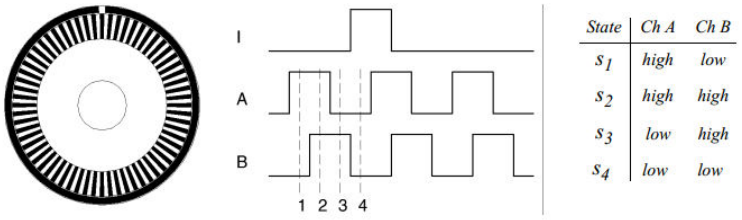
Sensores inerciais: São sensores externos que fazem referência com o mundo externo, ou seja, medem variáveis de posição do robô, tais como:

- Encoder óptico: Estes dispositivos medem a velocidade e a posição angular em um *driver* de motor ou no eixo de uma roda. Este sensor, segundo Siegwart et al. (2011), é um interruptor de luz, que produz uma quantidade de pulsos para cada revolução do eixo. Os pulsos gerados pela diferença de fase entre os canais A e B são utilizados para determinar a direção de rotação, como pode ser visto na Figura 9a.

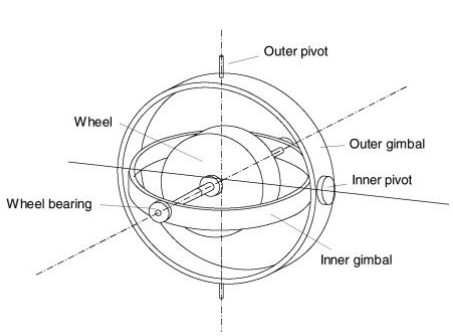
- Giroscópios: São sensores que determinam a orientação em relação a um referencial fixo, com base nas forças que são aplicadas sobre ele. Podem ser mecânicos ou ópticos. Na Figura 9b pode ser visto um giroscópio de dois eixos.

- Acelerômetros: Siegwart et al. (2011) diz que estes sensores são capazes de medir todas as forças externas que estejam agindo sobre ele, inclusive a gravidade. São baseados em sistemas pendulares, em que a primeira integração das acelerações proporciona a velocidade e a segunda, a posição (SECCHI, 2012). A precisão deste dispositivo é de grande importância, visto que pequenos erros podem influenciar na posição estimada, devido à dupla integração das acelerações. A Figura 9c mostra um circuito integrado de um acelerômetro.

Sensores de infravermelho: São sensores de proximidade rápidos e baratos, que basicamente consistem na emissão de um pulso infravermelho e detecção do sinal refletido, com a distância aproximada calculada através da intensidade do sinal refletido (DUDEK; JENKIN, 2010).



(a)



(b) **(c)**

Figura 9 – Sensores inerciais: (a) Sensor óptico; (b) Giroscópio; (c) Acelerômetro.
Fonte: (SIEGWART et al., 2011)

2.6 CONDICIONAMENTO DE SINAIS

Fraden (2006) diz que um sensor é um dispositivo que recebe um estímulo (uma quantidade, condição ou propriedade que é medida) e responde com um sinal elétrico. Como foi mencionado na seção anterior, os sensores são componentes fundamentais na construção de um robô móvel, visto que este precisa reconhecer o ambiente em que está inserido.

Muitos sensores têm uma resposta não-linear, são sensíveis à temperatura e apresentam outros fatores que podem comprometer a medição. Assim, é necessário corrigir estas adversidades para a correta utilização destes dispositivos.

Dunn (2006) diz que o condicionamento de sinais são as modificações ou mudanças necessárias para corrigir as características de entrada e saída de um sensor. Para Dunn (2006), os seguintes aspectos devem ser levados em conta para o correto condicionamento de sinais de um sensor genérico:

- Condicionamento do *offset* (alteração na amplitude de um valor) e *span* (alcance da amplitude, sendo a diferença entre o valor máximo e o mínimo em uma faixa de medição): Fraden (2006) diz que o *span* é um intervalo dinâmico de estímulos, os quais podem ser convertidos por um sensor. Também é chamado de fundo de escala de entrada (*input full scale*) e representa o maior valor de entrada possível que pode ser aplicado a um sensor para que não ocorra uma imprecisão que exceda os limites toleráveis.

- Linearização de circuitos analógicos: Um sistema linear, segundo Nise (2012), possui duas propriedades: a de superposição, em que a saída do sistema para a soma de suas entradas, é a soma das entradas individuais do sistema; e de homogeneidade, em que a multiplicação de uma entrada por um escalar acarreta em uma resposta que é multiplicada pelo mesmo escalar. Como muitos sensores não têm uma relação linear entre as variáveis físicas e o sinal de saída, as características não-lineares do sensor precisam ser corrigidas ou condicionadas para a uma adequada aplicação (DUNN, 2006). O autor diz que a linearização é complexa de ser feita, a não ser que se tenha uma equação simples que descreva as características do sensor. Fraden (2006) diz que a não-linearidade de um sensor pode ser expressa por sua sensibilidade ou *span* (é expresso em porcentagem) ou em termos da variável medida, como em amperes, no caso de medição de corrente elétrica.

- Correção de temperatura: Sensores são dispositivos sensíveis a temperatura, com o seu valor de saída esperado alterado à medida em que a temperatura muda, sendo que muitas vezes essa mudança não é linear (DUNN, 2006). Para que possa ser feita a correção

dos efeitos da temperatura no dispositivo, é necessário utilizar um elemento sensível a temperatura para monitorar as alterações no sensor. O autor diz que a compensação depende das características dos sensores, porque estas variam de um componente para outro, mas geralmente são utilizados circuitos em ponte.

- Correção de ruído: São utilizados filtros para a correção dos ruídos e em alguns casos, amplificadores, quando se tem um sinal de baixo nível em um ambiente com alto ruído.

Para tipos específicos de sensores, o condicionamento de sinais pode ser diferente. Sendo assim, estes não serão abordados nesta seção. Mais informações podem ser obtidas em Dunn (2006) e Fraden (2006).

No presente trabalho, o condicionamento de sinais dos sensores será simples, já que para circuitos mais complexos serão adquiridas placas que contêm os dispositivos já condicionados. Isso se deve ao fato de que o veículo necessita de desempenho, o que pode ser ampliado se forem utilizados componentes com tamanho e peso reduzido.

2.7 MAPEAMENTO DO AMBIENTE DE NAVEGAÇÃO

Em alguns casos é interessante para um robô saber como que é o ambiente em que ele se encontra.

Para Siegwart et al. (2011), são necessárias quatro etapas para que ocorra a navegação de um robô móvel: Percepção (interpretação que o robô tem sobre as informações de seus sensores); localização (o veículo precisa determinar a sua posição no ambiente); cognição (como a máquina deve agir para que os seus objetivos sejam alcançados); e controle de movimento (o controle que o robô tem sobre os seus motores para que seja possível realizar a trajetória desejada). Dentro destas categorias, a localização é uma área que vem tendo destaque. Um dos desafios para a robótica móvel é a localização em mapas.

Na localização em mapas, o robô tenta se localizar através das informações provenientes de seus sensores, depois atualiza sua percepção em relação ao mapa do ambiente em que está inserido (SIEGWART et al., 2011). Segundo o (SIEGWART et al., 2011), o trabalho de desenvolver um mapa resulta em uma arquitetura capaz de mapear e navegar com sucesso em vários ambientes, fazendo com que o esforço inicial seja recuperado ao longo do tempo. A Figura 10 mostra uma arquitetura robótica com navegação baseada em mapas.

Dudek e Jenkin (2010) diz que duas representações de mapas são importantes para os robôs móveis:

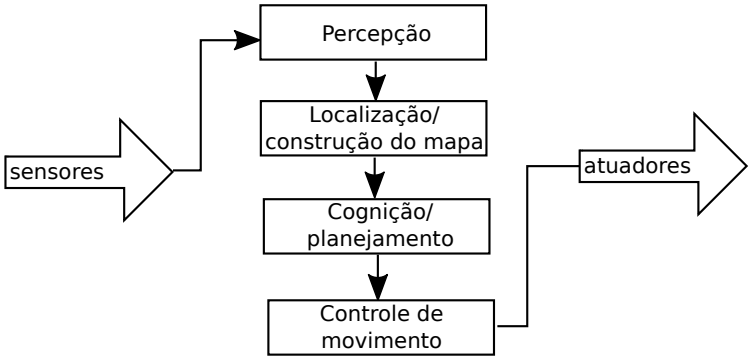


Figura 10 – Arquitetura de navegação baseada em mapas
Fonte: Adaptado de (SIEGWART et al., 2011, p.277).

- Mapa métrico: é baseado na referência absoluta e na estimativa numérica de onde os objetos estão no ambiente. É o mais intuitivo, sendo semelhantes a escala de mapas turísticos de uma cidade;
- Mapa topológico (também conhecido como mapa relacional): Representa somente a conectividade da informação e geralmente é implementado como grafo [9]. Esta representação é parecida com um mapa de metropolitano (metrô).

Neste trabalho será desenvolvido um mapa com representação topológica, pois é a que melhor condiz com as necessidades de projeto.

2.8 REGRAS DA ROBOCORE PARA ROBÔS SEGUIDORES DE LINHA

Na Seção 2.8.1 e Seção 2.8.2 são apresentadas as regras relacionadas à especificação dos robôs e do percurso, respectivamente, para a categoria robô seguidor de linha Pro, em eventos realizados pela Robocore (2016c).

2.8.1 ESPECIFICAÇÃO DOS ROBÔS

Para competir na categoria seguidor de linha, os robôs devem ser totalmente autônomos, não podendo ser controlados externamente por fio ou por rádio, exceto durante a sua inicialização. Todos os componentes devem ser embarcados. A dimensão máxima permitida é de 250mm de comprimento, 250mm de largura e 200mm de altura. Não é permitido alterar

[9] Um grafo é a relação entre objetos, os quais contém um vértice e uma aresta, de um mesmo conjunto, em que cada arco está associado a dois vértices, sendo a ponta com a seta o destino e a outra ponta a origem. Mais informações podem ser encontradas em (SIEGWART et al., 2011).

as dimensões do robô durante a partida, assim como alterar o *hardware* ou *software* durante a tomada de tempo. Também não é permitida a utilização de mecanismo de sucção, que vise aumentar a força normal do robô em relação ao solo.

2.8.2 ESPECIFICAÇÕES DO PERCURSO

A pista é feita de uma ou mais placas de MDF revestidas com uma manta de borracha preta. Assim, eventualmente são necessárias emendas para compor a área do percurso. Os robôs, no entanto, devem ser capazes de superar os desníveis decorrentes das emendas, que são de aproximadamente 1mm. Uma linha branca, de 19$\pm$1mm, indica o percurso. Esta linha pode cruzar sobre ela mesma, tendo, neste caso, um ângulo de interseção de 90$\pm5^{o}$ (graus), com os 250mm antes e depois do cruzamento sendo retas (conforme pode ser visto na Figura 11). O circuito é idealmente plano, porém podem ocorrer inclinações de até 5^{o}.

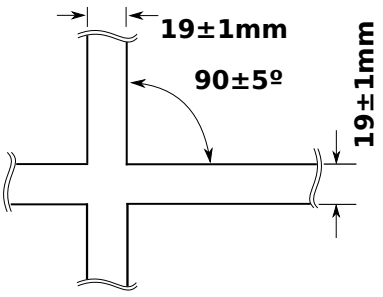


Figura 11 – Interseções no percurso
Fonte: Adaptado de (ROBOCORE, 2016c, p.4).

A área que se estende entre o ponto de partida e o ponto de chegada, considerando 200mm a direita da linha e 200mm a esquerda da linha é denominada "área de partida-chegada", conforme pode ser visto na Figura 12. Os retângulos representam as marcações laterais encontradas na pista.

Quando houver um arco (interseção entre a faixa branca), o raio deste é de pelo menos 100mm. Quando houver uma alteração na curvatura do percurso, deve haver uma marcação no lado esquerdo da linha, como pode ser visto na Figura 13. As linhas de partida e de chegada estarão localizadas em uma reta no circuito, sendo que esta será localizada um metro antes da linha de partida.

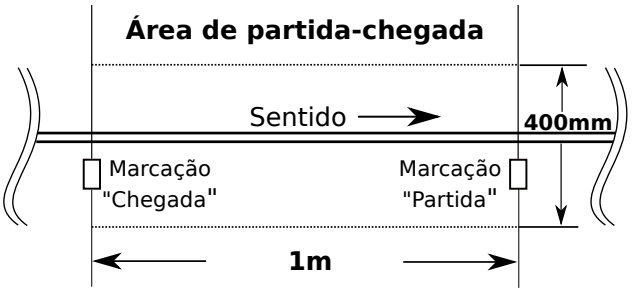


Figura 12 – Área de partida-chegada
Fonte: Adaptado de (ROBOCORE, 2016c, p.4).

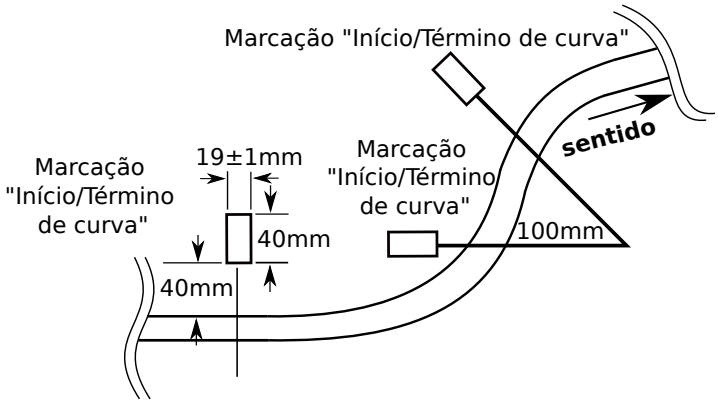


Figura 13 – Marcações de sinalização de curvatura
Fonte: Adaptado de (ROBOCORE, 2016c, p.4).

2.9 TRABALHOS RELACIONADOS

Na revisão bibliográfica a seguir são mostradas técnicas desenvolvidas para o sistema de navegação de robôs móveis autônomos.

O objetivo do trabalho de Freitas (2016) foi o desenvolvimento de um sistema robótico autônomo em uma arquitetura híbrida de três camadas, composta pela implementação das camadas Planejador e Executivo. O estudo de caso foi feito pelo sistema robótico DORIS, um robô guiado por trilhos, desenvolvido pela COPPE/UFRJ, em parceria com a Petrobras e a Statoil, para a inspeção de plataformas de petróleo. O veículo tem como funcionalidades a detecção de anomalias por vídeo, áudio, vibração, temperatura e por câmera de infravermelho e o mapeamento 3D do ambiente.

Heinen (2002) também propôs uma arquitetura de controle híbrida, integrando as técnicas de controle deliberativo e reativo em uma abordagem de três camadas: Deliberativa, a qual determina um trajeto até o objetivo e evita colisões com obstáculos já conhecidos; Vital, que utiliza de comportamentos reativos para guiar o robô ao seu objetivo e evitar colisões com obstáculos (sejam estes estáticos ou dinâmicos); e Funcional, que faz a integração das camadas vital e deliberativa, fornecendo parâmetros reativos para a camada vital. O sistema de controle

proposto foi implementado no simulador SimRob3D, o qual permite a utilização de modelos de ambiente tridimensionais, assim como sensoriais e cinemáticos, permitindo que estes possam ser alterados em tempo real.

A pesquisa de Pessin (2013) se concentra na busca por soluções inteligentes aplicadas em robôs móveis autônomos visando a operação destes em ambientes dinâmicos. Através da aplicação de aprendizado de máquina, buscou-se uma nova visão sobre a operação destes dispositivos em três desafios da área: navegação, localização e operações com grupos de robôs. No estudo das operações com grupos de robôs, foi utilizada uma aplicação de combate a incêndios, com a avaliação por quatro técnicas: Algoritmos Genéticos, Otimização por Enxame de Partículas, *Hill Climbing* e *Simulated Annealing*. Na investigação sobre a navegação, é apresentado o desenvolvimento de um veículo autônomo de grande porte, funcional para ambientes externos. Relacionado com a localização, é mostrado um método que provê informações de localização para os robôs com base em dados obtidos por redes sem fio.

A seguir serão abordados os trabalhos relacionados aos robôs seguidores de linha.

2.9.1 ROBÔS SEGUIDORES DE LINHA

Nesta Seção serão abordados os trabalhos de (GUADAGNIN, 2014), (PETRY, 2016) e (HIRAI, 2014):

- O trabalho de Guadagnin (2014) foi pioneiro na Universidade Tecnológica Federal do Paraná - câmpus Pato Branco, sendo o primeiro a ser feito sobre robôs seguidores de linha na instituição. Em seu trabalho foi implementado um sistema de controle híbrido, no qual o autor define como sendo um sistema no qual ocorre interação entre variáveis contínuas e a eventos discretos. Para o controle contínuo, que era o responsável pela posição do veículo, foi utilizado um controlador P (ação de controle proporcional), devido à planta já apresentar um integrador. O controle discreto, encarregado da detecção de marcas na pista, foi implementado por um autômato de Moore, composto por dez estados e quinze transições. Este controlador definia a velocidade do robô em 100%, para retas, em 75%, para curvas, e 0%, quando parado (em seu estado inicial). Com os controladores contínuos e discretos projetados, estes foram reunidos e implementados no microcontrolador. A detecção da pista foi feita por sensores de refletância.

  Com o protótipo implementado, foram feitos testes em uma pista desenvolvida por Guadagnin (2014), a qual foi baseada em pistas da competição Robocore. O percurso consistia de quatro curvas e dois cruzamentos. Primeiramente foram feitos cinco testes, sendo que três destes o robô não reconheceu as marcas laterais devido à inclinação da

pista não estar conforme a recomendada. Com a inclinação da pista corrigida, foram feitos oito testes, sendo que destes em somente um as marcas não foram detectadas. Também foi alterada a velocidade em curvas para 82%, sendo que nesta situação o veículo não conseguiu realizar a curva. O robô desenvolvido por Guadagnin (2014) pode ser visto na Figura 14a.

- O trabalho de Petry (2016) teve como base o de Guadagnin (2014). Logo nota-se a presença de várias semelhanças entre ambos, como o controle híbrido. Foi proposto para este projeto um *hardware* mais eficiente, com um novo sistema de sensoriamento, e a implementação da lógica *fuzzy*, também conhecida como lógica difusa, a qual foi integrada ao controlador discreto.

  O sistema de controle discreto foi modelado por autômato de Moore, utilizando autômatos determinísticos de estados finitos. Este controlador é muito semelhante ao projetado por Guadagnin (2014), com alteração na velocidade em curvas, sendo utilizado 80%. O controlador *fuzzy* foi projetado com base no valor dos sensores de refletância, comandando a velocidade dos motores de acordo com a base de regras desenvolvida. O controlador contínuo escolhido foi o PID, que assim como o *fuzzy* também utiliza as informações dos sensores de refletância para gerar a ação de controle.

  A estrutura do robô de Petry (2016) foi feita utilizando uma placa de fenolite, na qual os componentes são soldados. Para a propulsão foram utilizados dois motores CC de 6V e 1000 RPM, da Pololu, duas rodas de pneus de silicone e um *driver* para motor CC TB6612FNG da Toshiba. A alimentação foi feita por uma bateria LiPo (Lítio-Potássio) de 7,4V de tensão e 32,5A de corrente, com capacidade energética de 1300mAh. Foram utilizados os módulos reguladores de tensão XL6009 e MP2259, sendo *step-up* (aumenta a tensão) e *step-down* (diminui a tensão), respectivamente. O microcontrolador utilizado por Petry (2016) foi o MSP430G2553, o qual possui um ADC de oito canais, trabalha com *clock* de 16MHz, memória *flash* de 16KB e 0,5KB de SRAM. Os sensores de refletância utilizados foram os QTR-1A e QTR-8A (o qual é composto por oito unidades de QTR-1A), este empregado no protótipo.

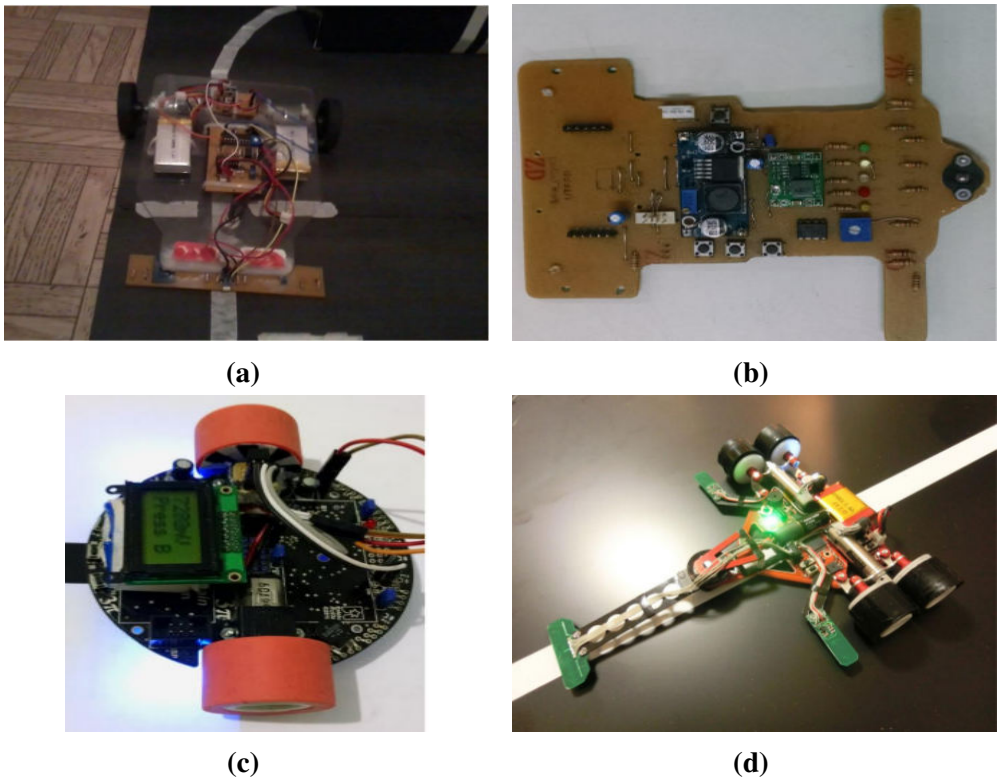
  Petry (2016) desenvolveu três protótipos, sendo que na Figura 14b pode ser visto o terceiro protótipo, o qual não funcionou devido a problemas com o circuito. Testes foram feitos com o segundo protótipo e de acordo com o autor, todas as marcações de curvas, cruzamentos e sinais (início e fim) foram reconhecidos. Também foi utilizado o robô 3pi, da Pololu, o qual pode ser visto na Figura 14c, para fazer a comparação entre os controladores PID e *fuzzy*. Este veículo foi modificado, com a substituição das rodas e dos motores originais pelos mencionados anteriormente. A bateria também foi

alterada, sendo utilizadas duas baterias em série, fornecendo 7,4V de tensão de 12A de corrente. Comparando os sistemas de controle desenvolvidos verificou-se que o *fuzzy* necessita de um processador bem mais eficiente comparado ao PID, com memória suficiente para implementar as variáveis de controle, além de maior poder computacional para processamento matemático, possibilitando o sistema de controle compensar o erro em tempo hábil. O controlador PID conseguiu realizar 680 leituras e atuações nos motores por segundo, enquanto que o *fuzzy* conseguiu somente 385. Com a configuração citada, o veículo não foi capaz de seguir a linha, devido à instabilidade. Então foram adicionadas quatro regras, as quais deixaram o robô mais estável mas com frequência de varredura muito baixa.

A conclusão inferida por Petry (2016), entretanto, não é totalmente concreta pois a comparação dos métodos foi feita em robôs diferentes (o *fuzzy* foi testado no 3pi e o PID testado no protótipo 2). Desta forma, não se teve um bom parâmetro comparativo entre os controladores, sendo que o controle de eventos discretos não foi utilizado na confrontação das técnicas, devido à limitações no *hardware*.

- Hirai (2014) é um dos principais competidores na categoria dos robôs seguidores de linha, tendo sido campeão em várias competições na Ásia, especialmente no Japão, onde em 2015 conquistou a terceira vitória seguida na Robotrace (HIRAI, 2016). O Cartisx04, o qual pode ser visto na Figura 14d, é um protótipo desenvolvido com base no Cartis03, sendo esse o seu último robô desenvolvido. Não se tem acesso ao *software* desenvolvido por Hirai (2014), no entanto é disponível o *hardware* utilizado. Logo é interessante conhecer como é feito o projeto de competidor de nível mundial.

  Segundo o Hirai, o CartisX04 é um robô leve de quatro rodas. São utilizados dois motores CC Maxon DCX10L e um motor de direção Maxon RE10. Os sensores utilizados são o giroscópio Invensense ISZ-650 e o sensor de refletância GP2S700, da Sharp. O microcontrolador utilizado foi o STM32F103RE, da STMicroelectronics, o qual é equipado com processador ARM-Cortex M3 de 32 bits, 72MHz de *clock*, 256KB de memória *flash* e 64KB de SRAM. As dimensões do veículo são 175mm de altura, 153mm de comprimento e 7mm de centro de gravidade. O robô pesa 98g, 32g a menos do que o protótipo anterior, o que para o autor propiciou aumento de desempenho.



(a) **(b)**

(c) **(d)**

Figura 14 – Robôs seguidores de linha: (a) Protótipo de Guadagnin (2014); (b) Robô *Alpha Project* (PETRY, 2016); (c) Pololu 3pi modificado (PETRY, 2016); (d) Cartisx04 de Hirai (2014).

3 MATERIAIS

Neste capítulo são apresentados os materiais utilizados neste projeto.

3.1 MICROCONTROLADOR

Foi utilizado o *Kit* de desenvolvimento NUCLEO-F303K8, da ST Microelectronics, o qual pode ser visto na Figura 15. Esta placa utiliza o microcontrolador STM32F303K8, que possui como processador ARM Cortex-M4 de 32 *bits*. Este processador dispõe de uma FPU, a qual facilita o processamento dos cálculos em ponto flutuante que são necessários para o controle do veículo. Esse *kit* de desenvolvimento possui trinta e dois pinos de conexão. A alimentação pode ser por USB ou por outra fonte externa e contém ainda três LEDs, sendo um deles acionado pelo usuário (STMICROELECTRONICS, 2016). As principais informações deste microcontrolador estão na Tabela 1.

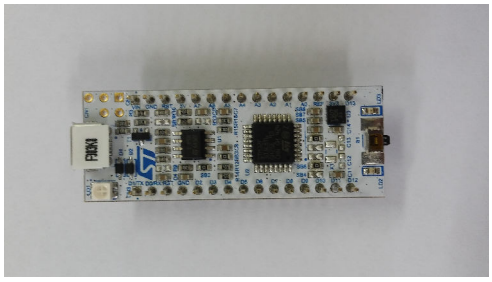


Figura 15 – *Kit* de desenvolvimento STM32-F303K8
Fonte: Autoria própria

.

Tabela 1 – Especificações do microcontrolador STM32F303K8

Característica	Descrição
Frequência de operação	72MHz
Desempenho	90 DMIPS[10]
Flash	64KB
SRAM	16KB
Quantidade de temporizadores (*timers*)	11
Quantidade de canais do ADC	21
Resolução do ADC	12 *bits*

Fonte: (STMICROELECTRONICS, 2015)

[10]DMIPS é uma medida de desempenho para avaliar um sistema embarcado.

3.2 MOTORES CC

Foi utilizado o motor CC modelo 3041 da Pololu, o qual pode ser visto na Figura 16. Estes motores são classificados pela fabricante como *High-Power Carbon Brush* (HPCB), os quais são motores escovados *brushed*[11]. Este modelo possui o eixo estendido, possibilitando o acoplamento de um *encoder*, a partir do qual é possível fazer o mapeamento da pista. As principais especificações técnicas deste dispositivo estão na Tabela 2.

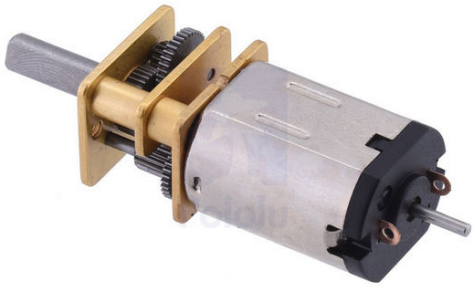


Figura 16 – Motor HPCB 3041
Fonte: (POLOLU, 2016b)

.

Tabela 2 – Especificações do motor 3041 da Pololu

Característica	Descrição
Tensão de alimentação	12V
Corrente de alimentação (sem carga)	100mA
Corrente máxima de alimentação	800mA
Caixa de velocidade	10:1
Rotação máxima	3000 RPM
Torque máximo	0,3kg-cm

Fonte: (POLOLU, 2016b)

3.3 PONTE H

Para o acionamento dos motores, foi utilizada uma Ponte H, que é um circuito que tem esse nome devido à sua configuração, que se parece com um 'H', como pode ser visto na Figura 17. A Ponte H controla não só a velocidade entregue aos motores, mas também o seu sentido. O modelo utilizado foi o TB6612FNG, da Toshiba, que é capaz de controlar até dois motores CC.

O dispositivo é baseado em MOSFET, que possui maior velocidade de chaveamento que os circuitos baseados em BJT, sendo adequado para o controle de motores (SEDRA; SMITH,

[11] Um motor escovado realiza a troca de fase do rotor através de escovas.

2010). A velocidade do motor é controlada por Modulação por Largura de Pulso (PWM, do inglês *Pulse Width Modulation*). As especificações deste dispositivo estão na Tabela 3

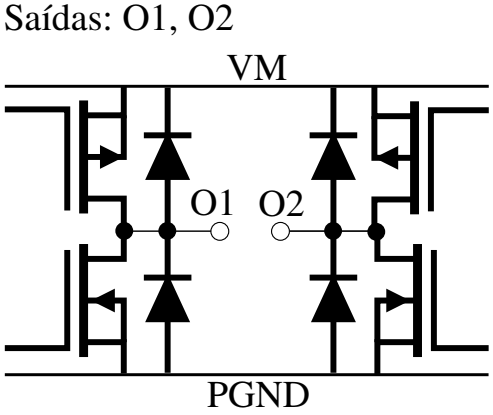


Figura 17 – Diagrama de uma Ponte H
Fonte: Adaptado de (TOSHIBA, 2016)

.

Tabela 3 – Especificações do *driver* TB6612FNG da Toshiba

Característica	Descrição
Tensão de alimentação	15V
Corrente de saída	1,2A
Corrente de saída (pico)	3,2A

Fonte: Adaptado de (TOSHIBA, 2016)

3.4 ENCODER MAGNÉTICO

Um *encoder* magnético é um transdutor de movimento, que converte movimentos em informações elétricas, sendo possível obter dados como posição e velocidade. Neste trabalho foi utilizado o modelo 3081 da Pololu, o qual pode ser visto na Figura 18. Este dispositivo utiliza um disco magnético e sensor de efeito *hall*, provendo doze contagens por revolução do eixo do motor (POLOLU, 2016a).

3.5 SENSORES DE REFLETÂNCIA

O sensor de refletância é um dispositivo eletrônico que consiste de um *Light Emitting Diode* (LED) e um fototransistor, medindo assim a refletância de uma superfície. O componente funciona da seguinte maneira: o LED emite um sinal luminoso invisível, o qual é refletido pela superfície; este sinal varia de acordo com cada superfície e assim pode-se saber sob qual a cor

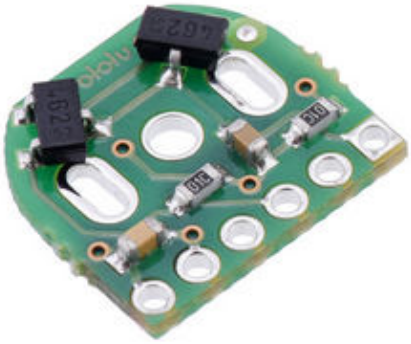


Figura 18 – Encoder magnético 3081
Fonte: (POLOLU, 2016a)

.

da superfície o sensor está.

Este circuito pode ser visto na Figura 19, em que foi utilizado para detectar a linha do percurso. O modelo utilizado nesse trabalho foi o QRE1113P, da Fairchild Semiconductor. As principais especificações deste dispositivo estão na Tabela 4.

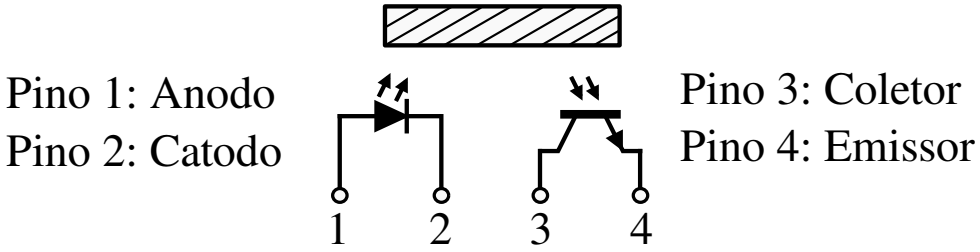


Figura 19 – Princípio de funcionamento de um sensor de refletância
Fonte: Adaptado de (SEMICONDUCTOR, 2016)

.

Tabela 4 – Especificações do sensor de refletância QRE1113P

Característica	Descrição
Tensão de alimentação	3,3V a 5V
Corrente de saída	1,2A
Corrente de saída (pico)	3,2A

Fonte: (SEMICONDUCTOR, 2016).

3.6 PLACA DE CIRCUITO IMPRESSO

O *chassi* do robô, ou seja, a estrutura deste, foi confeccionada em uma placa de circuito impresso (PCB) de fenolite.

3.7 MÓDULO BLUETOOTH

Foi utilizado o módulo *bluetooth* HC-05 para a telemetria. Este módulo possuiu a configuração mestre-escravo e comunicação *Universal Asynchronous Receiver Transmitter* (UART).

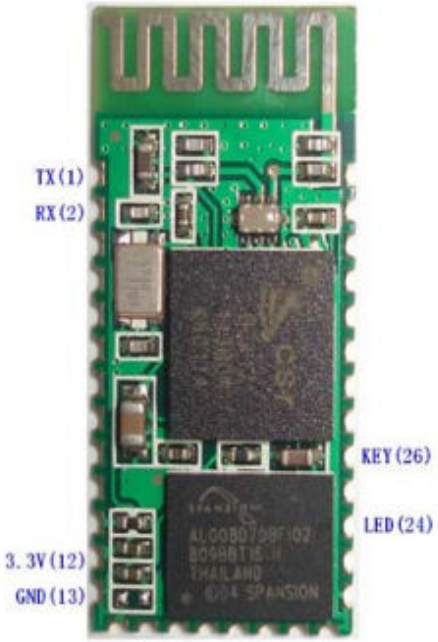


Figura 20 – Módulo *bluetooth* HC-05
Fonte: (TECHNOLOGY, 2016)

.

3.8 RODAS E PNEUS

Foram utilizadas rodas com pneus de silicone. As rodas foram adquiridas do México, com recursos próprios.

3.9 BATERIA LIPO

A alimentação do robô foi feita por uma bateria do tipo Lítio-Polímero (Li-Po) de três células, de 11,1V de tensão, 300mAh de energia e 7,5A de corrente máxima de descarga, pois possui alta capacidade de corrente e densidade de carga.

3.10 CIRCUITO *STEP-UP*

O *step-up*, também conhecido como *boost*, é um conversor CC-CC utilizado para elevar uma tensão de entrada a um valor desejado, desde que esteja no limite do circuito. Nesse trabalho foi utilizada uma placa com o circuito *step-up* já condicionado, o qual pode ser visto na Figura 21. O componente é um circuito *boost* capaz de fornecer até 60 V e 4 A de corrente, com frequência de chaveamento de 400 KHz, sendo conhecido por sua alta eficiência, de até 94% (XLSEMI, 2016).

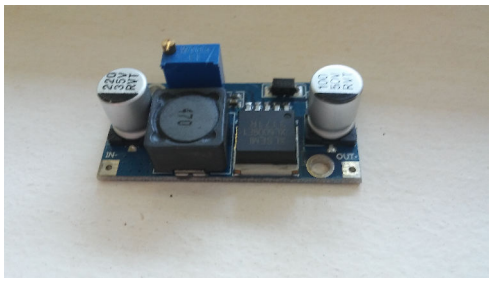


Figura 21 – Circuito *step-up*
Fonte: Autoria própria

.

3.11 *SOFTWARE*

Os principais *softwares* que foram utilizados neste trabalho estão na Tabela 5. A UTFPR possui algumas licenças para uso do *software* proprietário Matlab, na versão 2013b.

Tabela 5 – *Softwares* utilizados

Aplicação	***Software***
Desenho de diagramas	Inkscape
Gráficos	Matlab
Modelagem do controle contínuo	Matlab
Modelagem do controle discreto	Supremica e Deslab
Ambiente de desenvolvimento	System Workbench
Inicialização do microcontrolador	STM32 CubeMX
Desenho das PCIs (EDA)	Kicad

4 METODOLOGIA

Neste capítulo são descritas as etapas do trabalho proposto e como as quais foram executadas. As etapas foram definidas da seguinte forma:

1. Definição e aquisição dos componentes: Nesta etapa, foram escolhidos os componentes que fizeram com que os objetivos de projeto possam ser executados;

2. Condicionamento de sinais: São utilizados sensores para que grandezas físicas pertinentes possam ser aferidas pelo sistema, tais como velocidade e tensão elétrica. No entanto, para que o dispositivo de aquisição de sinais possa medir o valor dos sensores eficaz e corretamente, se faz necessário fazer o condicionamento de sinais destes dispositivos. Este processo foi realizado através das descrições contidas nos *datasheets* (folha de especificação) dos componentes.

3. Projeto da estrutura mecânica e *software* embarcado: Esta parte consiste em projetar o *chassi* do robô (a sua estrutura física) com todos os componentes necessários para o seu funcionamento. Foi utilizado um *software* do tipo *Electronic Design Automation* (EDA) para a criação da placa de circuito impresso (PCI). Nesta etapa também foram feitos os códigos no microcontrolador, que é o dispositivo de processamento do sistema.

4. Projeto do controlador: Para o projeto do controlador PID, foi utilizada a modelagem experimental para obter um modelo matemático, visto que neste caso fica mais simples do que a fenomenológica, conforme dito por Guadagnin (2014). Para este foram serão utilizados *softwares* matemáticos para a simulação do modelo obtido. Para o projeto do controlador de Sistemas a Eventos Discretos, foi utilizado o modelo de Moore, pois é de fácil implementação e é oportuno ao que se deseja implementar. Foi utilizado o Supremica, ferramenta computacional de controle supervisório, para esta modelagem.

5. Integração do sistema e implementação do protótipo: Com as etapas anteriores finalizadas, é necessário integrá-las. A integração foi feita com os sensores e atuadores sendo soldados ou acoplados na PCI e os controladores e a aquisição de sinais implementados no microcontrolador.

6. Testes de desempenho: Foram realizados testes de desempenho de modo a verificar o que pode ou o que precisa ser melhorado no protótipo. Podem ser feitos ajustes tanto de

hardware (como a substituição de componentes) quanto de *software* (alterações no código do microcontrolador). Pode ser necessário retornar a etapas anteriores para realizar estas correções.

7. Implementação do projeto final: Com os testes de desempenho e as alterações necessárias feitas, é implementado o projeto final de forma a validar este trabalho. Na versão final do veículo espera-se atingir os objetivos que foram propostos na Seção 1.4.

5 DESENHO E IMPLEMENTAÇÃO

5.1 PROJETO DO *HARDWARE*

A Figura 22 mostra o diagrama de blocos referente ao funcionamento do robô de maneira resumida. Com base nele que o projeto de *hardware* foi desenvolvido.

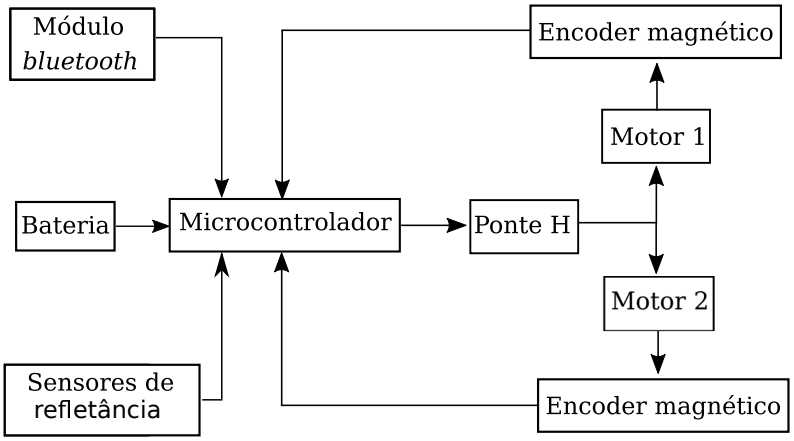


Figura 22 – Diagrama de funcionamento do *hardware* do veículo.
Fonte: Autoria própria.

5.1.1 SENSORES E CONDICIONAMENTO DE SINAIS

***Driver* de acionamento TB6612FNG:** Para este dispositivo, não foi feito nenhum tipo de condicionamento de sinal, já que foi adquirida uma placa que contém o circuito integrado já condicionado (TOSHIBA, 2016). A Figura 23 mostra o esquemático da placa deste dispositivo.

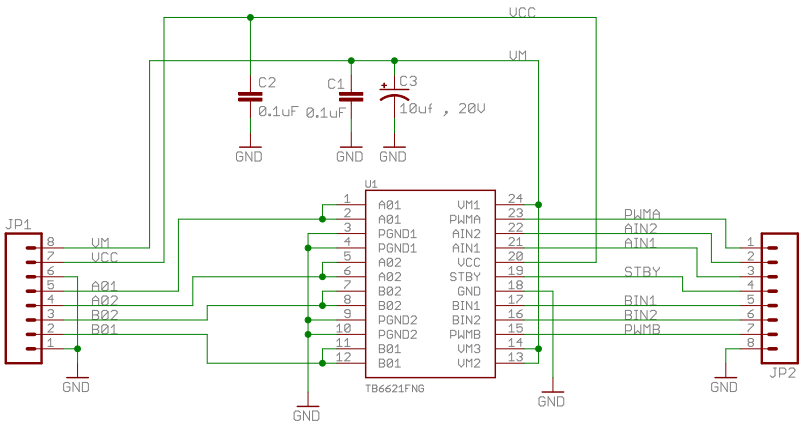


Figura 23 – Esquemático do *driver* de acionamento TB6612FNG.
Fonte: (SPARKFUN, 2017).

A Tabela 6 mostra os modos de operação da Ponte H TB6612FNG. São cinco possíveis modos de utilização: *Short Brake* (Pequena Parada), CW (*Clockwise* ou Horário), CCW (*Counterclockwise* ou Anti-horário), *Stop* (Parado) e *Standby* (Aguardando), em que os modos CW e CCW correspondem ao sentido em que os motores serão acionados e nos modos restantes não ocorre acionamento. São possíveis três estados para o dispositivo: H (*High* ou Alto), L (*Low* ou Baixo) e Alta Impedância (desconectado do resto do circuito). Para evitar que circule corrente reversa pelo circuito, é necessário colocar o dispositivo em um modo intermediário (Parado ou Aguardando) quando alternar entre os modos CW ou CCW e Parada pequena, e vice versa (SPARKFUN, 2017).

Tabela 6 – Modos de operação do *driver* de acionamento TB6612FNG
Fonte: Adaptado de (SPARKFUN, 2017)

Entrada				**Saída**		
EN1	EN2	PWM	STBY	SAÍDA 1	SAÍDA 2	Modo
H	H	H/L	H	L	L	Pequena parada
L	H	H	H	L	H	CCW
		L	H	L	L	Pequena parada
H	L	H	H	H	L	CW
		L	H	L	L	Pequena parada
L	L	H	H	Alta impedância	Parado	
H/L	H/L	H/L	L	Alta impedância	Aguardando	

Sensor de refletância QRE1113: No *datasheet* do QRE1113GR encontram-se os valores típicos e os valores máximos de operação do componente. Desta forma, visando o correto funcionamento do dispositivo, foram definidos dois resistores para cada sensor. O valor da alimentação dos sensores, V_{cc}, foi considerado como 3,3V.

O primeiro resistor, R_1, é conectado entre a alimentação do sensor, V_{cc}, e o anôdo do LED do QRE1113GR, como pode ser visto na Figura 24. De acordo com a folha de dados do componente, o LED trabalha com uma corrente típica de 20mA e sua queda de tensão é de cerca de 1,2V. Desta forma, como pode ser visto nas Equações 8 e 9, R_1 deve possuir uma resistência maior do que 105Ω; logo, foi escolhido o resistor de 120Ω, o qual é um valor comercial e satisfaz os critérios de projeto.

$$20mA \leqslant \frac{V_{cc} - 1,2}{R_1} \tag{8}$$

$$R_1 \geqslant 105\Omega \tag{9}$$

O outro resistor, R_2, é colocado entre V_{cc} e a entrada do coletor do fototransitor, como pode ser visto na Figura 24. De acordo com a folha de dados do QRE1113GR, o valor tipico

de corrente que passa pelo coletor é de 0,4mA. Desta forma, segundo as equações 10 e 11, R_2 deve ter resistência maior do que $8,2K\Omega$; então, foi escolhido o resistor de $10K\Omega$, o qual possui valor comercial e se enquadra nas características desejadas de projeto.

$$0,4mA \leqslant \frac{V_{cc}}{R_2}; \tag{10}$$

$$R_2 \geqslant 8,2K\Omega \tag{11}$$

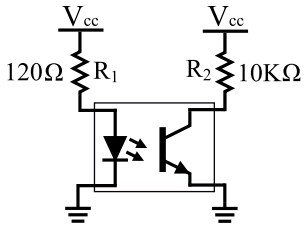


Figura 24 – Configuração do sensor de refletância.
Fonte: Autoria própria.

Regulador de tensão AMS1117: O AMS1117 pertence a uma família de reguladores de tensão fixa ou variável, sendo projetados para fornecer até 1A de corrente (SYSTEM, 2017). Neste trabalho, foi utilizado o modelo de tensão fixa de 3,3V. A tensão de V_{cc}, resultante deste componente é a tensão que alimentará os sensores de refletância, o módulo *bluetooth* e os *encoders* magnéticos.

De acordo com a folha de dados deste componente, é necessária a utilização de um capacitor na saída do circuito, para a compensação de frequência do dispositivo. Assim, com base no *datasheet* do componente, foram inseridos dois capacitores de tântalo, de $15\mu F$, na saída e na entrada do circuito, nos pinos 2 e 3, respectivamente. O pino 4 não é utilizado. A Figura 25 mostra a configuração do emprego do regulador de tensão neste trabalho.

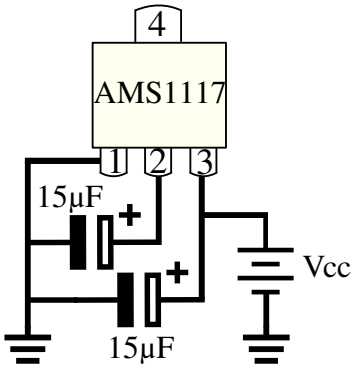


Figura 25 – Configuração do regulador de tensão AMS1117.
Fonte: Autoria própria.

5.2 PROJETO DO CONTROLADOR DE SED

Como abordado anteriormente, o controlador de Sistemas a Eventos Discretos é o responsável pelas ações a serem tomadas pelo sistema com base nos sinais obtidos do botão e pelos sensores laterais.

Este controlador foi graficamente modelado pelo *software* Supremica e pode ser visto na Figura 5.2. O código referente à este controlador foi obtido pela utilização do programa Deslab (TORRICO, 2015). O controlador foi modelado utilizando o autômato de Moore, no qual a ação correspondente a determinado estado é executada no próprio estado.

Diferentemente dos trabalhos anteriores, de (PETRY, 2016) e (GUADAGNIN, 2014), em que o SED continha os comandos para o controle das ações do robô em tempo real, neste trabalho ele será utilizado para mapear a pista, ou seja, obter a posição de cada início e fim de curva e de pista, com base nos *encoders*.

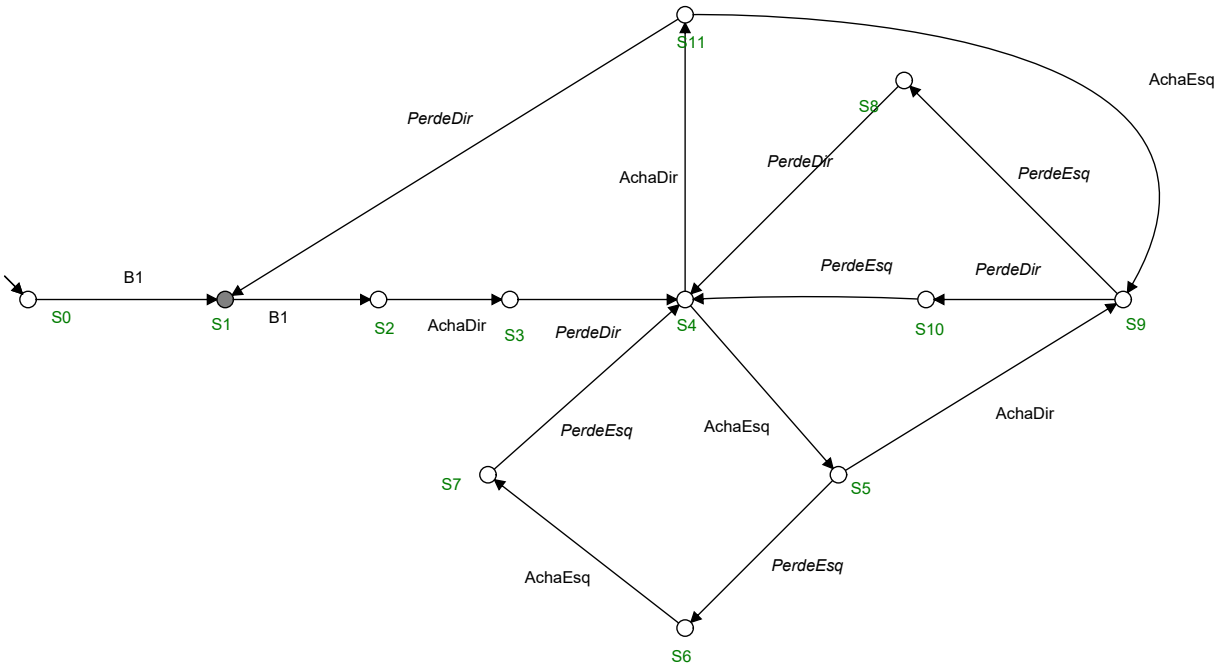


Figura 26 – Controlador SED
Fonte: Autoria própria.

Cinco variáveis são mapeadas como eventos não-controláveis [12] desse controlador, sendo:

- B1 (botão), utilizado para as funções iniciais e de configuração do SED;
- AchaDir (marcação lateral), utilizado quando se encontra uma marcação do lado direito, referente ao início e fim de pista;

[12] Eventos não-controláveis ocorrem de maneira espontânea e não podem ser evitados, logo sempre estão habilitados.

- PerdeDir (marcação lateral), utilizado quando se perde uma marcação do lado direito, referente ao início e fim de pista;
- AchaEsq (marcação lateral), utilizado quando se encontra uma marcação do lado esquerdo, referente à início e fim de curva;
- AchaEsq (marcação lateral), utilizado quando se perde uma marcação do lado esquerdo, referente à início e fim de curva.

Para obter um melhor desempenho, são utilizados dois padrões de corrida: o de reta e o de curva. Para cada um está associada uma constante proporcional, uma constante derivativa e uma velocidade. Isso se faz necessário, visto que os parâmetros do controlador contínuo precisam ser alterados em decorrência da velocidade (*duty cycle* aplicado). O veículo também pode estar parado, tendo velocidade nula.

As sequências de cada estado (S_x, em que x é o número do respectivo estado) do protótipo, são as seguintes:

- S_0: Estado inicial, em que o robô fica parado, esperando que o botão B1 seja apertado e gere um evento. Esse estado é obrigatório mesmo com o mapeamento, visto que o mapeamento poderá ser acionado;
- S_1: Neste estado é feito o calibramento dos sensores do veículo. Na terceira versão do robô, CrazyFrog 3, foi implementado um sistema automático de calibramento, permitindo que os sensores da barra frontal sejam calibrados sem intervenção de um observador humano (somente é necessário apertar o botão B1 para que este estado seja atingido). Nesta etapa também são habilitadas as interrupções dos sensores laterais. Caso a variável *qMap* seja diferente de zero, ou seja, o mapeamento das curvas já acabou, os motores são desligados, a posição dos *encoders* e a quantidade de curvas é armazenada no vetor, o qual logo é programado na memória *Flash*;
- S_2: Começa a corrida, com o robô estando com os parâmetros de curva (velocidade e constantes do PID);
- S_3: Foi encontrada uma marcação do lado direito, a qual representa o início da corrida;
- S_4: O sistema nesse estado pode ter se originado de duas maneiras: foi detectado o sinal de perda da marcação direita, tendo sido precedido pelo Estado S_3; ou o SED detectou um cruzamento, tendo sido originado pelos Estados S_{10} e S_8. Se for originado por uma curva (pela *flag curve*), os valores do *encoders* são armazenados no vetor;

- S_5: Foi encontrada uma marcação esquerda. Nada se altera neste Estado;
- S_6: Foi perdida uma marcação esquerda. Desta forma, nesse estado, os valores do *encoders* são escritos no vetor de mapeamento e a variável de quantidade de curvas, *qMap*, tem o seu valor acrescido em uma unidade;
- S_7: Foi encontrada uma marcação esquerda. É ativada a *flag curve*, para indicar ao Estado 4 que o evento veio de uma curva e não de um cruzamento;
- S_8: O Estado está em um cruzamento e foi perdida uma marcação esquerda. Esperando perder a marcação da direita para sair do cruzamento;
- S_9: Início de um cruzamento, podendo ter sido ocasionado por encontrar um marcação da direita ou da esquerda. Espera-se perder uma marcação da direita e outra da direita para sair do cruzamento;
- S_{10}: Está em um cruzamento, logo para sair dele é necessário perder uma marcação da direita;
- S_{11}: Achou uma marcação da direita. Pode estar em cruzamento, caso encontre uma marcação da direita, ou na última reta do percurso, ao perder a marcação da direita.
- S_5: Foi encontrada uma marcação esquerda. Nada se altera neste Estado;
- S_6: Foi perdida uma marcação esquerda. Desta forma, nesse estado, os valores do *encoders* são escritos no vetor de mapeeamento e a variável de quantidade de curvas, *qMap*, tem o seu valor acrescido em uma unidade;
- S_7: Foi encontrada uma marcação esquerda. É ativada a *flag curve*, para indicar ao Estado 4 que o evento veio de uma curva e não de um cruzamento;
- S_8: O Estado está em um cruzamento e foi perdida uma marcação esquerda. Esperando perder a marcação da direita para sair do cruzamento;
- S_9: Início de um cruzamento, podendo ter sido ocasionado por encontrar um marcação da direita ou da esquerda. Espera-se perder uma marcação da direita e outra da direita para sair do cruzamento;
- S_{10}: Está em um cruzamento, logo para sair dele é necessário perder uma marcação da direita;
- S_{11}: Achou uma marcação da direita. Pode estar em cruzamento, caso encontre uma marcação da direita, ou na última reta do percurso, ao perder a marcação da direita.

O modelo apresentado é genérico e pode modelar a maioria das pista de competição para seguidores de linha que apresentem marcações laterais. A excessão é quando a pista apresenta o 'S', ou seja, para uma dada curva, ao invés de ter uma marcação de início e outra de fim de pista, esta apresenta três marcações: uma de início de pista, uma de continuidade de curva e outra que indica o fim da curva. Essa particularidade não pode ser corretamente tratada neste modelo, sendo que caso essa condição ocorra, o veículo pode parar em estados incorretos. Esse problema não foi corrigido neste trabalho, necessitando ser estudado e tratado em trabalhos futuros.

5.3 PROJETO DO SISTEMA DE MAPEAMENTO

Conforme visto anteriormente, na Seção 5.2, o mapeamento se inicia no controlador de Sistemas a Eventos Discretos. Cada Estado S_x não obstante executa uma ação imediata no veículo, mas armazena os valores dos *encoders* em um vetor, a partir do qual é possível que o veículo se oriente na pista, sabendo os pontos certos em que deve aumentar e reduzir velocidade, bem como o de desligar os seus motores.

O vetor responsável por armazenar os valores relacionados a quantidade de marcações (início e fim de pista e de curva) e os valores dos *encoders* para cada uma dessas marcações, é o vetor *MapBuff*. Esse vetor é do tipo *uint16_t*. O primeiro elemento é a quantidade de curvas que a pista possui (conta-se uma unidade para cada duas marcações, ou seja, começo e fim de curva ou começo e fim de pista). Em seguida, são armazenados quatro valores respectivos a marcação de início e fim de percurso: valor do *encoder* da direita, no começo e no fim pista, e do *encoder* da esquerda, também para o começo e no fim do percurso. Os seguintes índices do vetor seguem a mesma regra, no entanto são respectivos a marcações de curva. O diagrama de funcionamento deste vetor pode ser visto na Figura 27.

A função **void FollowMap(...)**, que pode ser vista no Apêndice A, é a responsável pelo controle do veículo quanto este estiver no modo de corrida pelo mapa. Ela recebe como parâmetros duas variáveis *uint16_t*, que são *countLeft* e *countRight*, referentes aos atuais valores dos *encoders* do motor esquerdo e direito, respectivamente. Nessa implementação, somente está sendo considerado o valor do motor direito para o controle do robô, pois não foi possível estabelecer um método mais eficaz para o mapeamento. Assim, para trabalhos futuros, a utilização dos valores dos dois *encoders* com uma abordagem mais inteligente, poderia ocasionar um desempenho melhor no veículo.

Nessa função, a variável *qMap* é a responsável pela orientação do veículo sobre o vetor *MapBuff*, indicando se o protótipo está ou não na última marcação. A função alterna os valores entre pista e reta pela variável *TypeLine* (se esta é zero, o controlador recebe os parâmetros de

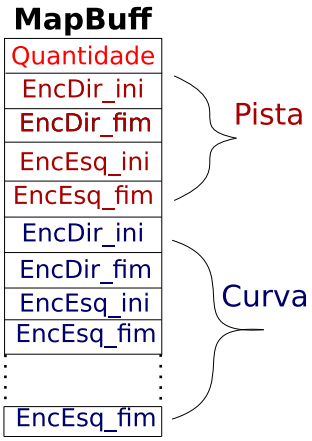


Figura 27 – Vetor com o mapeamento
Fonte: Autoria própria.

curva, caso contrário, recebe os parâmetros de reta). Quando chegar na marcação de fim de percurso, além de desligar os motores, também é acionada a *flag END_OF_PATH*, a qual não permitirá que essa função seja chamada novamente.

Esse vetor é armazenado na memória *Flash* do microcontrolador, a qual possui 64 KB de espaço de armazenamento (STMICROELECTRONICS, 2015). A memória *Flash* é um dispositivo não-volátil, que diferentemente da SRAM do microcontrolador, tem os dados perdidos quando a alimentação do dispositivo é interrompida, ela mantém os dados armazenados mesmo ausente de energia elétrica. No entanto, o processo de leitura e gravação na memória *Flash* não é tão trivial quanto escrever em EEPROM, sendo necessárias algumas etapas a serem tomadas.

Para escrever na *Flash*, os procedimentos a seguir são necessários:

1. Destravar a *Flash*, habilitando os registradores da memória;
2. Limpar as *flags* que foram ativadas no processo de destravamento;
3. Apagar a página (ou as páginas) que será gravada na *Flash* (todos os endereços desta página, que no SMT32F303K8 possui 2 KB, necessariamente precisam ser apagados e caso deseje-se manter alguma informação, é necessário reescrevê-la);
4. Escreve-se na memória *Flash* a informação desejada, no endereço especificado (o endereço inicial utilizado para gravar os valores na *Flash*, neste trabalho, é 0x08008000, com os restantes sendo incrementados a partir deste);
5. É travada a memória, de forma a não permitir a escrita no dispositivo.

Para a leitura na *Flash*, não é necessário nenhum procedimento adicional, somente saber

os corretos endereços a serem lidos. O código referente à leitura e escrita na memória *Flash* pode ser visto no Apêndice B.

É importante destacar que ao se utilizar a *Flash* é que apresenta um número limitado de gravações (na faixa das milhares de gravações). Desta forma, não é recomendado para informações sendo programadas com muita frequência, podendo causar danos permanentes no componente. Para esse caso, recomenda-se a utilização de uma memória EEPROM, a qual não apresenta um número limitado de operações de escrita.

O mapa feito nesse trabalho pode ser considerado como um mapa métrico, de acordo com Dudek e Jenkin (2010), visto que o veículo se baseia na referência absoluta de cada marcação e na estimativa numérica de onde cada objeto está no ambiente.

5.4 FUNÇÃO DE TRANSFERÊNCIA DO VEÍCULO

Nesta seção serão apresentados os procedimentos adotados para a obtenção da Função de Transferência do robô. Para esta etapa do trabalho, foi utilizado o *software* Matlab. Os resultados foram obtidos através da terceira placa do robô seguidor de linha, denominado CrazyFrog 3.

5.4.1 AQUISIÇÃO DOS VALORES DA PLANTA

Para obter a função de transferência do veículo, é necessário analisar o comportamento deste a partir de uma entrada. Desta forma, foi realizado um procedimento para adquirir o desempenho do protótipo a partir de uma entrada, com o objetivo de obter a posição do robô para cada ciclo de PWM.

A Figura 28 ilustra o procedimento adotado imediatamente antes dos valores serem adquiridos. O sensor mais a direita da barra frontal é posicionado sobre linha branca, enquanto que o restante fica fora da faixa branca, ou seja, sob a parte preta da pista. Os motores funcionam com o mesmo *duty cycle*, mas em diferentes sentidos, com o motor direito orientado para trás e o da esquerda para a frente, conforme pode ser visto pelas indicações na figura 28. As informações foram armazenadas na estrutura **Map** (uma *struct* em lingugagem C), a qual contém a posição do robô e o tempo em milisegundos, no qual a posição foi adquirida. Os motores desligam e a aquisição dos valores de posição são finalizados quando o sensor localizado mais a esquerda permanece sob a faixa branca. Após a aquisição de dados ter sido completada, o vetor da estrutura **Map** é enviado via *Bluetooth* para um celular.

Este mesmo procedimento também foi feito na versão anterior do seguidor de linha, CrazyFrog 2, no entanto os resultados não foram satisfatórios, o que foi relacionado com o fato de que esta placa não permitia a inversão de giro; neste caso, um dos motores funcionava enquanto o outro permanecia parado.

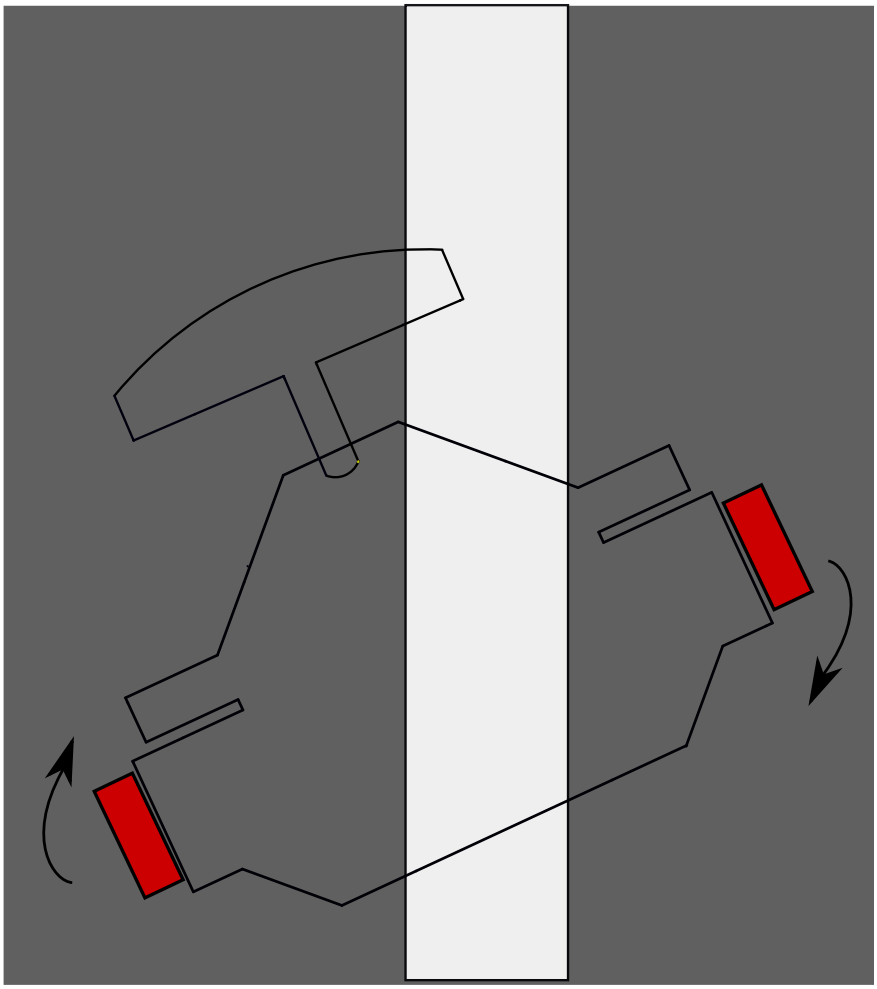


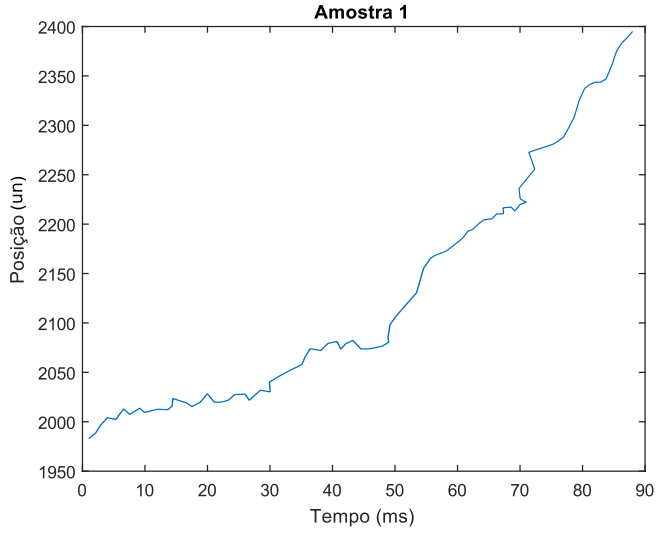
Figura 28 – Aquisição dos valores de posição do robô
Fonte: Autoria própria

5.4.2 MODELO DA FUNÇÃO DE TRANSFERÊNCIA

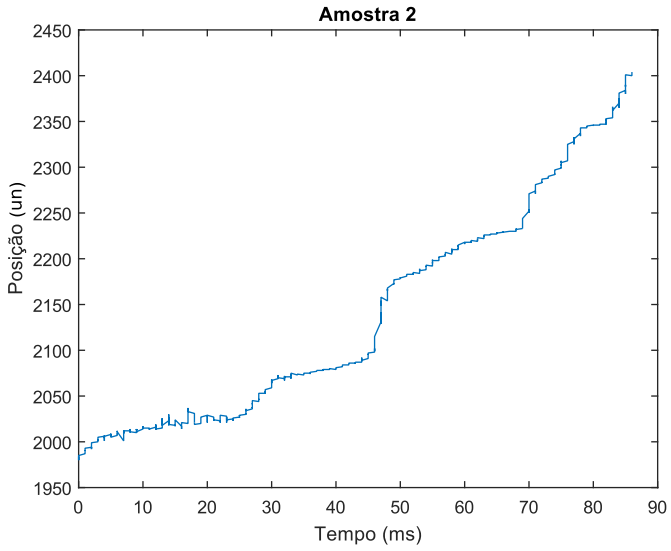
Na Seção 5.4.1 foi obtido um vetor, o qual contém os valores da posição do robô que são alterados durante a rotação dos sensores da barra frontal pela linha branca. A posição é calculada linearmente e os sensores sempre se encontram na mesma altura em relação à superfície. Assim, é esperado que o gráfico gerado por este vetor seja próximo a uma rampa, sendo que quanto mais próximo de uma rampa for este gráfico, melhor será a aproximação para a função de transferência. O cálculo da posição consiste em uma média ponderada feita com

os sensores de refletância, sendo que desta forma não se obtêm uma grandeza associada aos valores obtidos. Para tanto, a grandeza utilizada para a posição e velocidade é a de unidades (un).

Foram realizados vários testes de forma a obter o gráfico da posição o mais próximo o possível de uma rampa. Para os testes, foram alterados a frequência do *timer* e o ciclo de trabalho do PWM. Os melhores resultados foram obtidos com a frequência do *timer* de 16 KHz e *duty cycle* de aproximadamente 30%. As Figuras 29a e 29b mostram o gráfico da posição do robô obtido em duas tentativas, nessas mesmas configurações de frequência e *duty cycle*, sendo que a Figura 29a é a qual será a utilizada no decorrer deste trabalho.



(a)



(b)

Figura 29 – Gráficos da posição em função do tempo (ms), com frequência de 16 KHz e *duty cycle* de 30%: (a) Amostra 1; (b) Amostra 2
Fonte: Autoria própria

Ogata (2010) diz que a dinâmica do sistema a ser estudado, independentemente de ser biológico, mecânico ou elétrico, deve ser representada em termos de equações diferenciais. Para tanto, não é possível estabelecer uma relação entre o vetor de posições que foi obtido e a função de transferência. Assim, para que seja possível obter essa relação, utiliza-se uma derivada numérica no vetor de posições que foi encontrado, de forma a ocasionar um novo vetor, que esteja em termos de funções diferenciais.

A derivada numérica por diferenças finitas em ponto x_0 pode ser vista na Equação 12, em que $f'(x_0)$ é a derivada de uma função $f(x)$ no ponto x_0. Mas se caso h seja pequeno e diferente de zero, tem-se a aproximação vista na Equação 13 (AZEVEDO, 2017). Se caso h, a diferença entre os pontos x_0 e x_1, for exatamente a distância entre entre x_0 e x_1, logo $x_0 + h = x_1$, como pode ser visto na Equação 14. Assume-se que $f(x_1)$ e $f(x_0)$ são, respectivamente, y_1 e y_2. Adaptando a Equação 14 para o vetor de posições (em que cada posição é $p(t)$), tem-se que y_1 e y_0 são $p(t+1)$ e $p(t)$, respectivamente, e que h é t, o intervalo de tempo entre uma posição e outra, como pode ser visto na Equação 15.

$$f'(x_0) = \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{12}$$

$$f'(x_0) \approx \lim_{h \to 0} \frac{f(x_0 + h) - f(x_0)}{h} \tag{13}$$

$$f'(x_0) \approx \frac{f(x_1 - f(x_0)}{h} = \frac{y_1 - y_0}{h}. \tag{14}$$

$$\frac{y_1 - y_0}{h} = \frac{p(t+1) - p(t)}{t} = \frac{dp(t)}{dt} = v(t). \tag{15}$$

Após ter sido feita a derivação numérica, tem-se um novo vetor, que contém valores respectivos à variação da posição em função do tempo $dp(t)/dt$, em que $p(t)$ é a posição em função do tempo), ou seja, um vetor da velocidade do veículo, $v(t)$. O gráfico desse vetor de velocidade pode ser visto na Figura 30.

A partir do gráfico da velocidade, visto na Figura 30, é possível encontrar a função de transferência da planta. Para encontrar o modelo mais próximo do real, foi utilizada a *toolbox* **System Identification**, do Matlab. Essa ferramenta tem como intuito produzir e comparar modelos de função de transferência com a curva real do sistema, produzido a partir do vetor de posições e da entrada desse. Na ferramenta, com os dados importados, são feitas estimativas para obter-se as funções de transferência. Uma das maneiras de estimar a função é através do **Process Models**, em que é possível escolher a quantidade de zeros e polos do sistema, bem

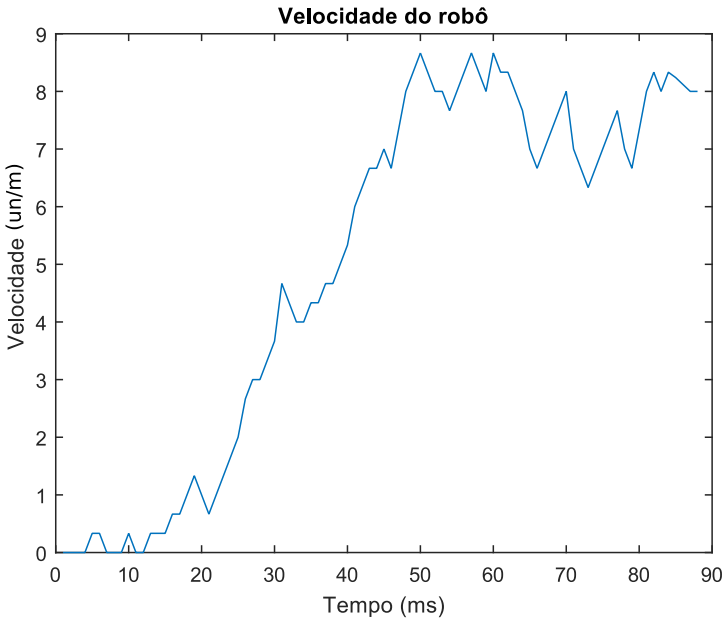
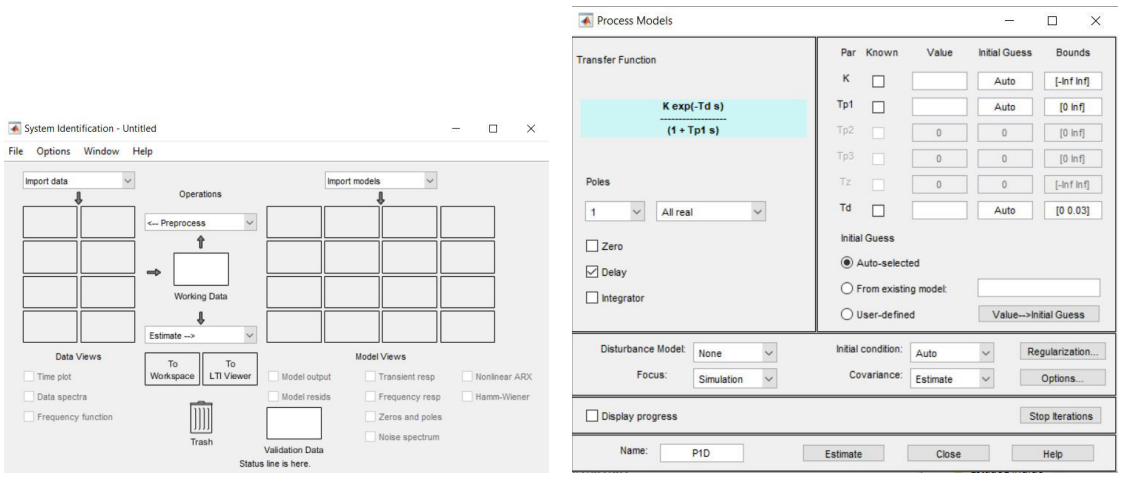


Figura 30 – Gráfico do vetor de velocidade do robô
Fonte: Autoria própria

como outras informações. As Figuras 31a e 31b mostram as interfaces gráficas do **System Identification** e do **Process Models**, respectivamente.



(a) **(b)**

Figura 31 – Ferramentas de estimativa de uma função de transferência do Matlab: (a) System Identification; (b) Process Models
Fonte: Autoria própria

A entrada de sinal dos sistema é o PWM dos motores, sendo que foi utilizado o valor de 30%. Esse valor percentual foi convertido de uma escala, a qual vai de 0 a 4095, visto que o período máximo do *timer* é de 4096. Assim, o valor da entrada é de 1228, conforme pode ser visto na Figura 32.

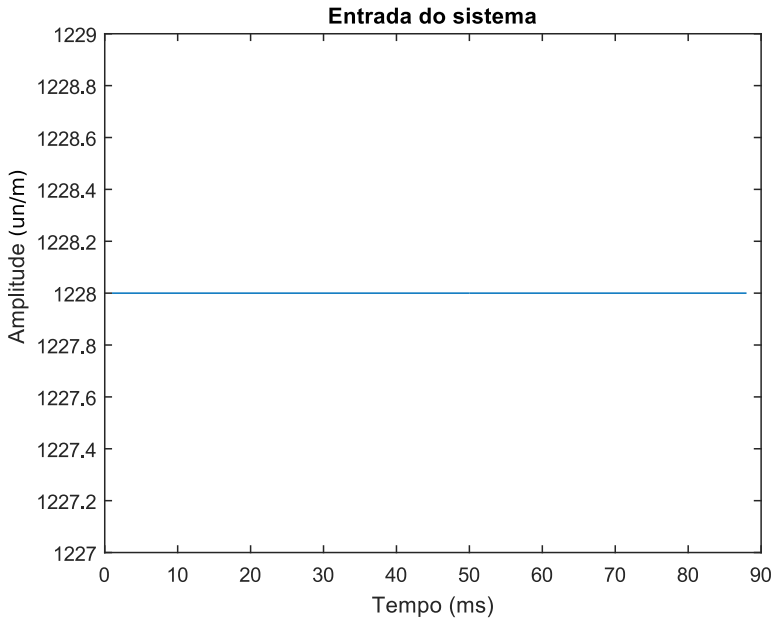


Figura 32 – Gráfico do sinal de entrada do sistema
Fonte: Autoria própria

O **System Identification** aproxima uma função de transferência com base em uma entrada de dados (entrada aplicada à planta), saída da planta e outros parâmetros definidos pelo usuário, como o período da aquisição dos dados (*sampling time*) e o tipo esperado da função (como a quantidade de zeros e polos). Para o período de amostragem foi feita uma média das unidades da velocidade, sendo então utilizado o período de 1 ms (milisegundos).

Pelo **Process Models**, foram encontradas as seis funções de transferência a seguir:

- $P1(s) = \frac{0{,}010971}{1+0{,}081618s}$, função de primeira ordem com 1 polo;
- $P1Z(s) = 0,0083694(\frac{1-0{,}007943s}{1+0{,}043801s})$, função de primeira ordem com 1 polo e 1 zero;
- $P2(s) = \frac{9{,}0888}{(1+5{,}7928e^{-5}s)(1+4192{,}6s)}$, função de segunda ordem com 2 polos;
- $P2Z(s) = 1,102(\frac{1-0{,}25668s}{(1+5{,}7932e^{-5}s)(1+9434{,}1s)})$, função de segunda ordem com 2 polos e 1 zero;
- $P2U(s) = \frac{0{,}0054952}{1+0{,}0014914s+0{,}000238s^2}$, função de segunda ordem com 3 polos;
- $P2ZU(s) = 0,0054915(\frac{1-0{,}011087s}{1+0{,}014889s+0{,}000238s^2})$, função de segunda ordem com 3 polos e 1 zero.

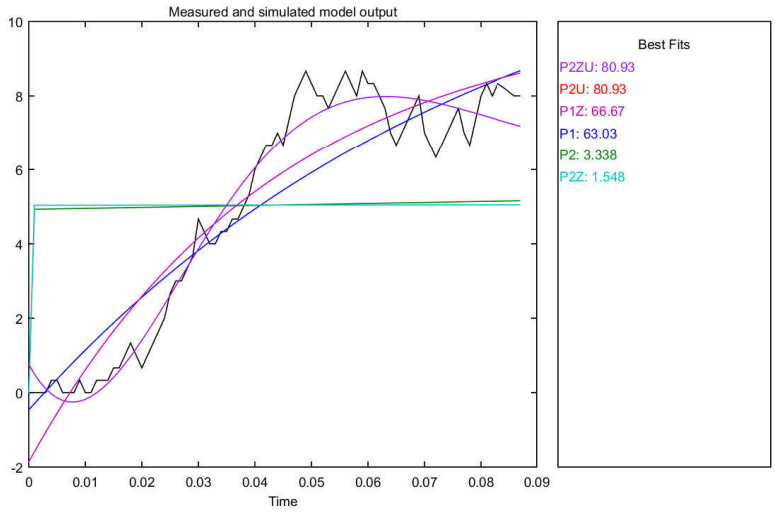


Figura 33 – Comparativo entre as Funções de Transferência
Fonte: Autoria própria

A Figura 33 mostra um comparativo entre as funções de transferência e o gráfico do vetor de velocidade. Pelo parâmetro *Best Fit*, o qual é mostrado na coluna a direita da figura, tem-se que a função que melhor se aproxima do gráfico de velocidade é $P2ZU(s)$, uma função de segunda ordem com três polos e um zero. Segundo o parâmetro *Best Fit*, tem-se que a função $P2ZU(s)$ tem 80,93% de compatibilidade com a função original. Desta forma, tem-se que $P2ZU(s)$ é a melhor aproximação para a função da velocidade do veículo.

No entanto, como esta função é originada a partir da derivação numérica da posição do protótipo, para encontrar a aproximação de sua planta, é necessário integrar $P2ZU(s)$. Para integá-la, é necessário acrescentar um polo $(1/s)$. Ao integrar-se $P2ZU(s)$, obtêm-se $P2ZU(s)/s$, que pode ser vista na Equação 16. Ao multiplicar essa função de transferência, $P2ZU(s)/s$, pelo ganho $K_{entrada}$, da entrada do sistema, é obtida a função de transferência da planta, a qual pode ser vista na Equação 17. O gráfico desta função, o qual foi obtido através da função **step(1228/s*P2ZU(s))**, pode ser visto na Figura 34.

$$\int P2ZU(s) = \frac{P2ZU(s)}{s} = Planta(s) = 0,0054915\frac{1-0,011087s}{s(1+0,014889s+0,000238s^2)} \tag{16}$$

$$K_{entrada}Planta = 6,743562\frac{1-0,011087s}{s(1+0,014889s+0,000238s^2))} \tag{17}$$

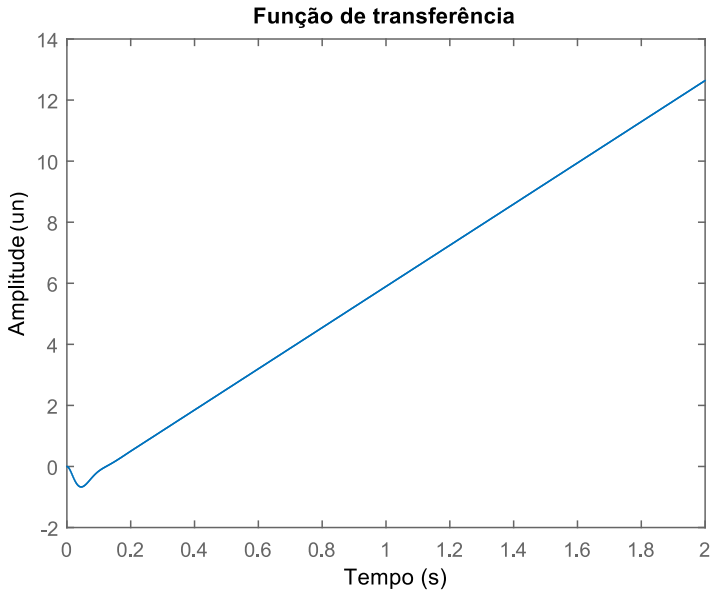


Figura 34 – Gráfico da Função de Transferência do veículo
Fonte: Autoria própria

5.5 PROJETO DO CONTROLADOR CONTÍNUO

O controlador contínuo desenvolvido foi do tipo PD (Proporcional-Derivativo), visto que a função de transferência da planta apresenta um integrador (como mostrado na Seção 5.4). Foi utilizado o sistema de malha fechada, como o que pode ser visto na Figura 4, em que o valor da saída é dependente do sinal de entrada e do sinal de referência. O código referente a esse controlador, o qual será detalhado a seguir, se encontra no Apêndice A.

No projeto do controlador PD, a variável a ser controlada é *posicao*, a qual é referente a posição do robô sob a linha branca. O sinal de referência é o valor da posição do sensor do meio da barra frontal, quando posicionado sob a linha branca, enquanto que os outros quatro estão fora da linha (sob a parte preta).

A variável *posicao* é obtida ao se multiplicar o valor de cada sensor por um número, múltiplo de 1000 e que corresponde a sua posição na barra, e dividir esse valor obtido pela soma dos valores de cada elemento. Esse procedimento pode ser melhor compreendido pelas Equações 18 e 19, sendo esta última já aplicada a barra de sensores desenvolvida neste trabalho. Nessas Equações, n corresponde à posição de cada sensor (quanto maior o n, mais a direita está localizado o componente) e V_n ao valor do respectivo sensor convertido pelo ADC, com valores possíveis de 0 a 4095, pois a resolução do periférico é de 12 *bits*.

Aproximando-se os valores lidos como 0 e 4095 para os sensores em cima da linha branca e na parte preta, respectivamente, e com base na Equação 19, tem-se que a posição de referência

para o controlador é 2000, a qual representa a barra de sensores no meio da linha branca. Mesmo que esse valor se altere no protótipo, é ainda uma ótima referência, sendo utilizada na prática satisfatoriamente.

$$posicao = \frac{\sum_{n=1}^{5} 1000(n-1)V_n}{\sum_{n=1}^{5} V_n} \tag{18}$$

$$posicao = \frac{0*V_1 + 1000*V_2 + 2000*V_3 + 3000*V_4 + 4000*V_5}{V_1 + V_2 + V_3 + V_4 + V_5} \tag{19}$$

Ao entrar na função **pdcontrol**, o que se nota primeiro é a escolha dos parâmetros, se são de curva ou de reta, sendo escolhidos pela variável *TypeLine* (se for 0, então o veículo está em uma curva ou ainda não iniciou a prova; se for 1, então está em uma reta).

Em seguida é encontrado o parâmetro proporcional, em que é subtraída a referência da posição, que foi calculada anteriormente. Esse cálculo é periódico, ocorrendo a cada 62,4 μs, visto que o valor dos sensores é lido somente a cada interrupção do *timer*, que está configurado com frequência de 16 KHz. A partir do valor da posição, é possível saber o erro do protótipo: se for maior do que 0, o carro está mais para a direita da curva; se for menor do que 0, o carro está mais para a esquerda; se for 0, então o veículo está exatamente no meio da linha.

O parâmetro derivativo, o qual é a derivada do erro, é obtido pela subtração do atual parâmetro proporcional pelo parâmetro proporcional da iteração anterior.

O sinal de erro é obtido pela variável $int16\_tsaida\_pwm$, o qual multiplica as respectivas constantes, de forma a obter a saída do PWM.

Em seguida, o novo cálculo do controlador PD é passado para os motores. Caso a saída do PWM seja negativa, o *duty cycle* aplicado ao motor da direita é aumentado e o da esquerda se mantém; caso PWM seja positivo, então o motor da direita se mantém e o da esquerda é reduzido.

5.6 PROGRAMAÇÃO DO MICROCONTROLADOR

A configuração do microcontrolador foi feita utilizando o CubeMX, um *software* disponibilizado pela ST, fabricante do microcontrolador STM32F303K8, que gera as configurações iniciais para o dispositivo. A interface do programa pode ser vista na Figura 35.

A DMA[13] do microcontrolador foi utilizada para obter os valores do ADC. A DMA tem a função de ler os valores dos ADCs e armazená-los na memória, sem a necessidade de utilizar o processador, o qual pode fazer cálculos durante esse tempo. O ADC também foi sincronizado

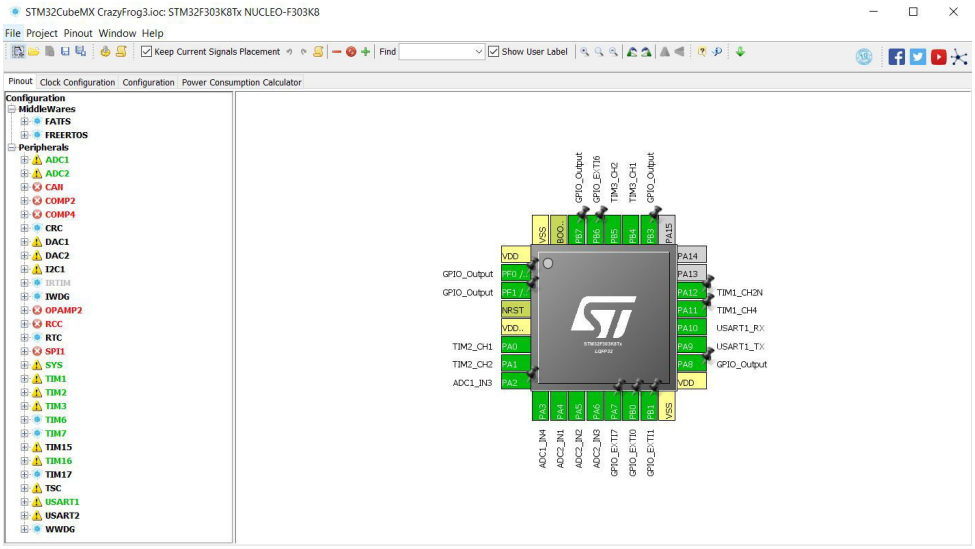


Figura 35 – Interface do STM32 CubeMX
Fonte: Autoria própria.

com o *timer* 1, o responsável pelo PWM dos motores, de forma que os valores dos sensores somente fossem lidos em um intervalo de tempo regular e o cálculo do controlador contínuo não fosse feito com base em valores antigos.

O código do microcontrolador, localizado no Apêndice A, contém as funções utilizadas no decorrer desse trabalho. Para reduzir o tamanho, esse código foi adicionado a este documento sem as configurações iniciais geradas pelo CubeMX (o que reduziu o documento em aproximadamente 10 páginas). Entretanto, a partir das configurações contidas no Apêndice C, é possível gerar essas mesmas configurações pelo *software* CubeMX. O código completo também está disponível para visualização no Google Drive da PatoBots, equipe de robótica móvel, e em repositório privado no Gitlab, o qual pode ser visto com autorização do autor desse Trabalho de Conclusão de Curso.

[13] A DMA (*Direct Memory Access* ou Acesso Direto à Memória) é um *hardware* que possibilita o acesso de periféricos diretamente à memória principal do computador, sem a necessidade de intervenção do processador.

6 EXPERIMENTOS E RESULTADOS

6.1 PROJETO E IMPLEMENTAÇÃO DA ESTRUTURA MECÂNICA

O desenvolvimento do *hardware* dos protótipos foi dividido em duas partes: O *chassi* do robô, no qual estão embarcados os principais componentes do sistema, como o microcontrolador, motores e o *driver* de acionamento destes; e a Barra frontal de sensores, a qual contém os sensores de refletância para a detecção da linha.

Foram projetados e confeccionados três (3) protótipos durante este trabalho, sendo que a primeira, segunda e terceira versão foram denominadas CrazyFrog, CrazyFrog2 e CrazyFrog3, respectivamente. Também foi projetada uma barra frontal de sensores, a qual é apresentada posteriormente.

Todas as placas descritas nessa seção foram confeccionadas na prototipadora do Bloco I, da UTFPR. A pista utilizada para os testes com os robôs durante esse trabalho, pode ser vista na Figura 36. Essa pista é feita com um material de borracha com coloração preta, com a linha e as marcações laterais tendo sido feitas com fita isolate branca. A pista segue com os mesmos parâmetros da Robocore.

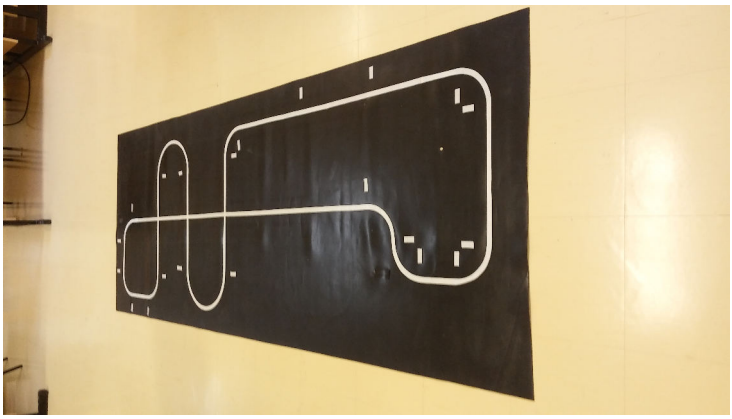


Figura 36 – Pista de testes da UTFPR
Fonte: Autoria própria

6.1.1 PROTÓTIPO 1

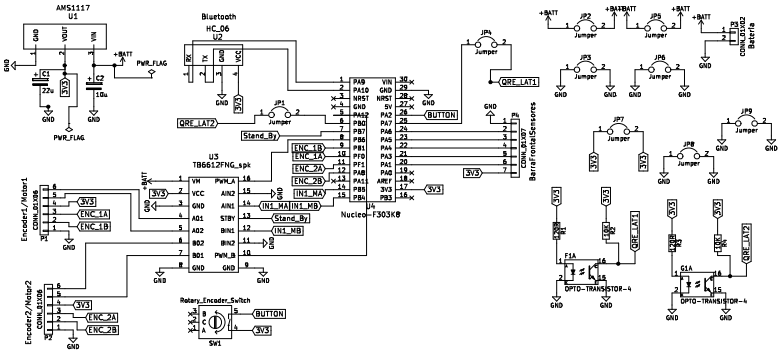
O *chassi* tem como finalidade ser a principal estrutura do veículo, sendo o responsável pela locomoção, controle e detecção das linhas laterais do robô. O esquemático desta placa pode

ser visto na Figura 37a, a qual contempla os dispositivos eletrônicos e os conectores utilizados, tendo o intuito de conectar outros elementos, tais como os motores e o módulo *bluetooth*.

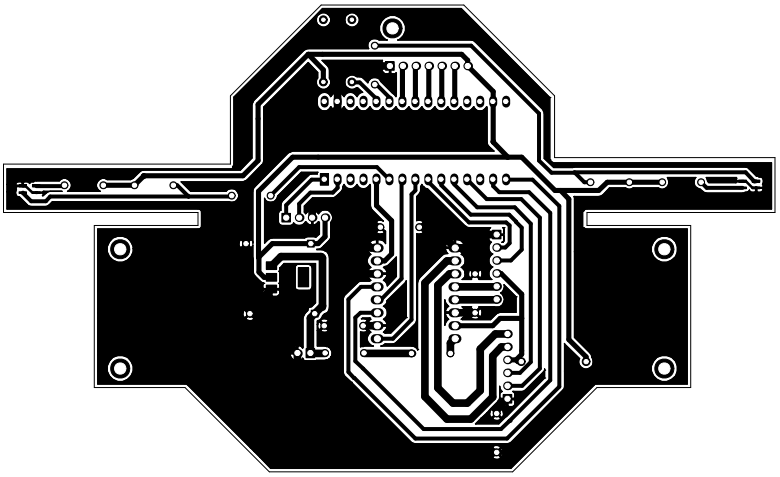
No projeto dessa parte do veículo, os componentes foram posicionados visando ter o seu peso uniformemente distribuído, possibilitando assim uma maior estabilidade para o robô, principalmente em curvas. Assim, o primeiro componente a ser posicionado foi a bateria 3S (de 11,4 V), a qual tem cerca de 19g (dezenove gramas). Logo em seguida, foram adicionados os motores e as rodas e em seguida os demais componentes. Os componentes de maior peso foram projetados para ficarem mais próximos das rodas do veículo, as quais fazem a sustentação desta placa.

O desenho da placa, gerado pelo Kicad, pode ser visto na Figura 37b. O veículo confeccionado pode ser visto na Figura 37c.

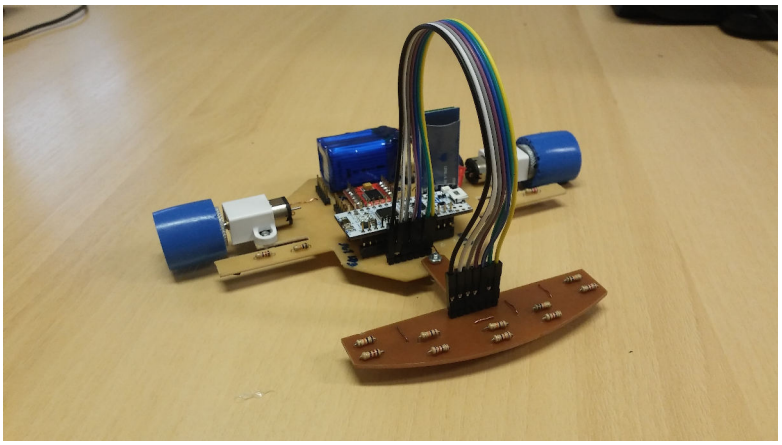
Não foi possível fazer muitos testes com essa placa, visto que um curto-circuito danificou permanentemente muitos componentes da placa, como o *driver* TB6612FNG e um dos capacitores de tântalo. Esse evento levou a necessidade de fazer uma nova placa para o *chassi*.



(a)



(b)



(c)

Figura 37 – CrazyFrog 1: (a) Esquemático; (b) Visualização do desenho da placa no Kicad; (c) CrazyFrog 1 confeccionado
Fonte: Autoria própria.

6.1.2 PROTÓTIPO 2

A segunda versão do *chassi* do veículo contou com algumas mudanças significativas. No projeto da placa feito no Kicad, foram adicionados dois MOSFETs, um *push-button* e um *slide switch* (chave deslizante) SMD.

O *switch*, o qual pode ser visto na Figura 38, foi adicionado na entrada de alimentação do protótipo, juntamente à bateria. Esse componente facilita muito os procedimentos de ligar e desligar o veículo, visto que não é mais necessário ligar a bateria diretamente no circuito (o que acontecia na versão anterior do protótipo).

Figura 38 – *Slide switch* utilizado no projeto
Fonte: (SWITCH,)

Os MOSFETs foram adicionados para garantir a segurança do circuito: o primeiro foi utilizado juntamente com o *slide switch*, de forma a evitar que seja aplicada uma tensão invertida na placa, o que pode ocorrer caso os fios de alimentação sejam trocados. O segundo MOSFET está localizado entre a alimentação positiva da bateria e a alimentação do *driver* de acionamento dos motores, a Ponte H TB6612FNG. Nessa configuração, o MOSFET impede que alguma corrente reversa dos motores ou do próprio *driver* possa retornar para a placa, o que poderia ocasionar danos nos componentes do protótipo.

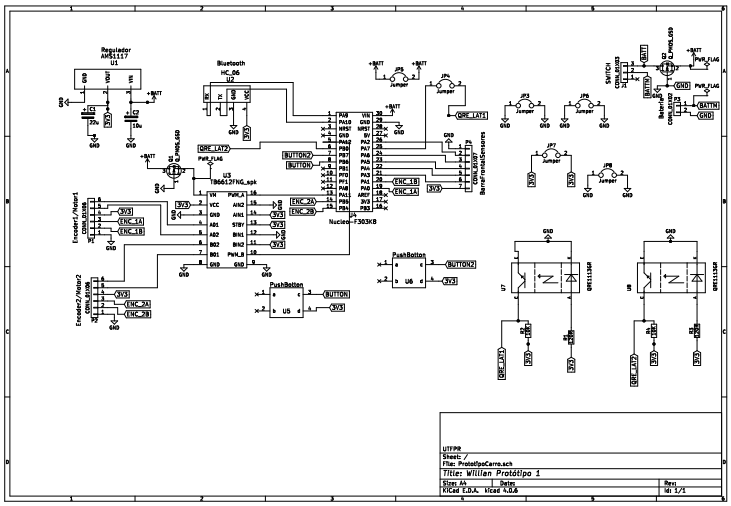
Os modelos de MOSFETs utilizados foram os AO3407, da Alpha & Omega Semiconductor, os quais são componentes SMD no encapsulamente SOT23 (SEMICONDUCTOR, 2011).

Foi alterado um dos capacitores: o capacitor de tântalo da entrada do regulador de tensão, que não necessita ser tão preciso quanto o da saída do dispositivo, foi substituído por um capacitor eletrolítico de alumínio com o mesmo valor. Essa mudança aconteceu devido a um curto-circuito na placa anterior ter danificado permanentemente o capacitor de tântalo. Entretanto, não foram notadas diferenças com esta substituição.

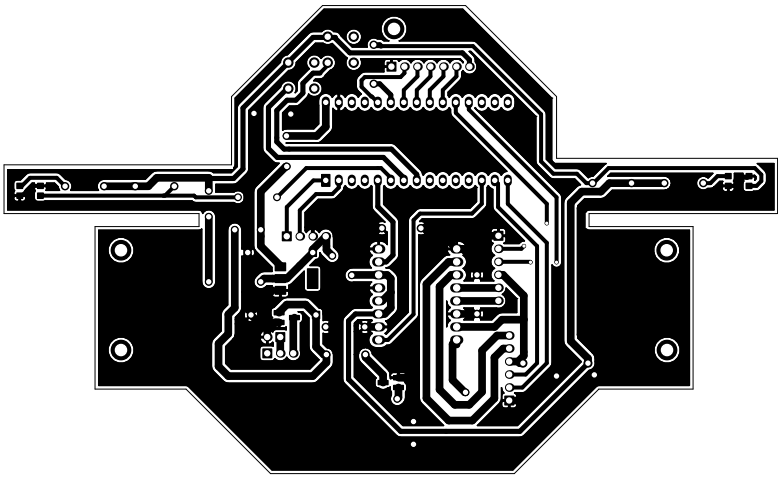
Para essa placa, também foram feitos três conectores, para conectarem o *chassi* à barra frontal de sensores e os dois motores. Os conectores foram feitos a partir de um cabo *flat* de um disco rígido e pinos fêmea (*female pin header*). Um desses conectores pode ser visto na Figura 39.

Figura 39 – Conector desenvovido
Fonte: Autoria própria

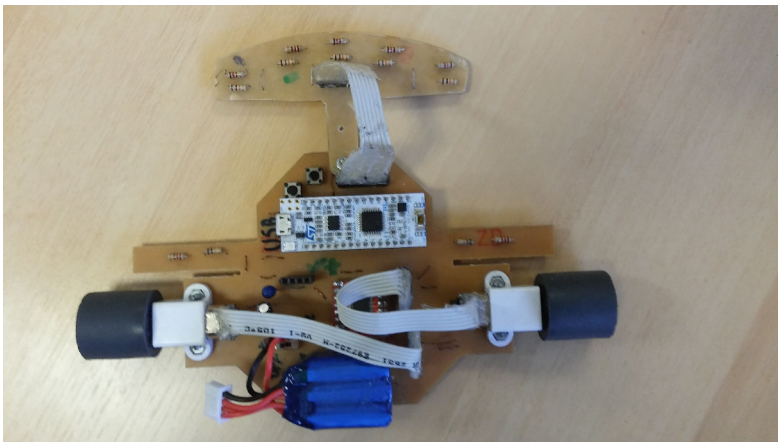
As Figuras 40a e 40c mostram o esquemático desta placa e o formato de desenho do *chassi*, respectivamente.



(a)



(b)



(c)

Figura 40 – CrazyFrog 2: (a) Esquemático; (b) Desenho da placa; (c) Protótipo confeccionado
Fonte: Autoria própria.

6.1.3 PROTÓTIPO 3

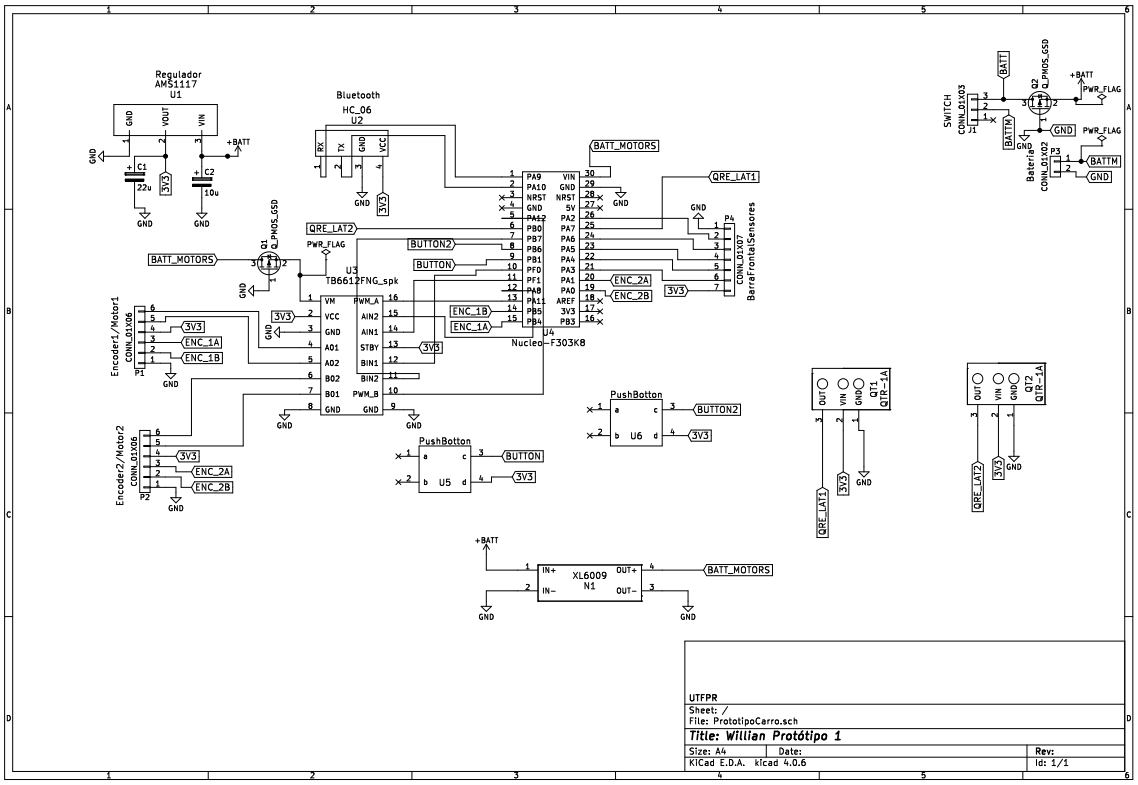
A terceira placa projetada, denominada CrazyFrog 3, foi a última desenvolvida e confeccionada nesse trabalho. As principais mudanças ocorridas foram a habilitação da inversão de giro na placa, a adição de um circuito *step-up*, a substituição do QRE1113, utilizado para a marcação lateral, pelo componente QTR-1A, da Pololu, e a troca da bateria LiPo de 11,4 V por uma de 7,6 V.

A inversão de giro dos motores tem a sua principal vantagem nas curvas. Quando está em uma curva, o controlador pode transmitir um PWM negativo a um dos motores. Caso isso ocorra, se o veículo tiver a inversão de giro implementada, o motor que foi acionado com um PWM negativo pode ter o seu sentido invertido, o que não necessariamente fará com que ele comece a girar no sentido oposto, mas que possa funcionar como um freio, permitindo assim que o veículo saia das curvas mais rapidamente.

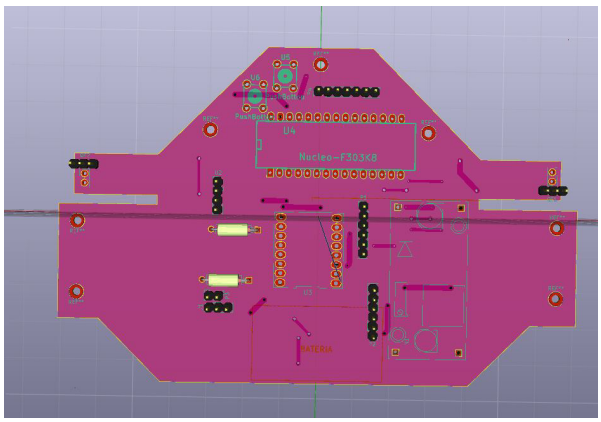
O circuito de *step-up* foi adicionado à placa por dois motivos: manter constante a tensão aplicada sobre os motores, que se variada afeta o desempenho do veículo, e diminuir a tensão aplicada sobre o regulador de tensão AMS1117, o qual esquentava consideravelmente quando aplicado à tensão de 11,4 V. O circuito de *step-up* também alimenta o *kit* de desenvolvimento Nucleo-F303K8, o qual necessita de uma tensão entre 12 V e 7 V para funcionar a partir de uma alimentação externa (STMICROELECTRONICS, 2016).

O sensor QRE1113, o qual era soldado diretamente na placa do protótipo, foi substituída pela placa QTR-1A, a qual mesmo possuindo este mesmo sensor, apresenta uma resposta superior. Essa troca foi feita com base de que é essencial para o controlador SED e o mapeamento desenvolvidos neste trabalho, a correta verificação das marcações laterais, logo se é necessária a utilização de um sensor com uma melhor resposta.

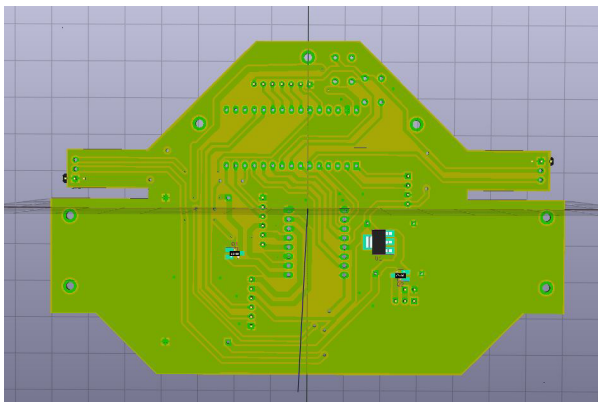
O esquemático do veículo, a visão superior e inferior em 3D (Três Dimensões) e o protótipo confeccionado podem ser vistos na Figuras 41a, 41b, 41c e 42, respectivamente.



(a)



(b)



(c)

Figura 41 – CrazyFrog 3: (a) Esquemático; (b) Desenho superior em 3D; (c) Desenho inferior em 3D

Fonte: Autoria própria.

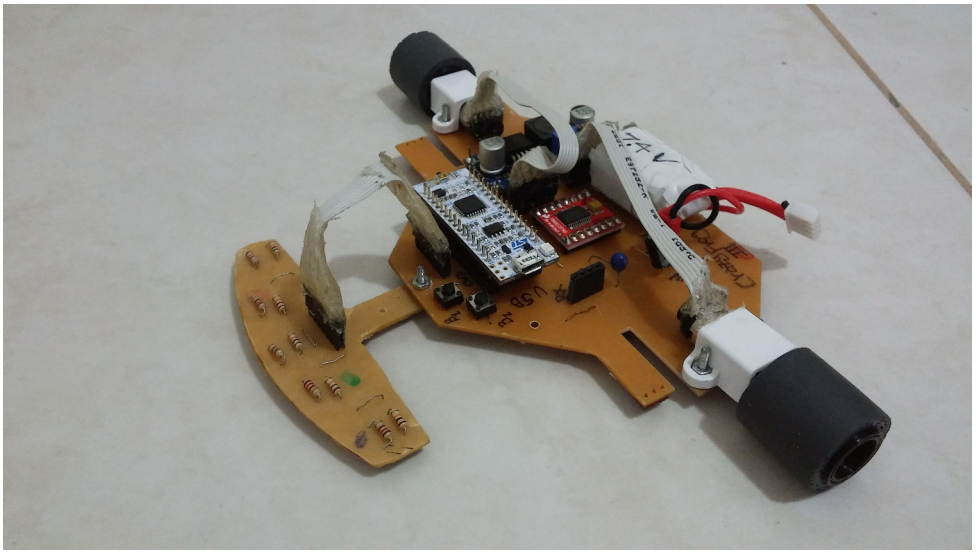
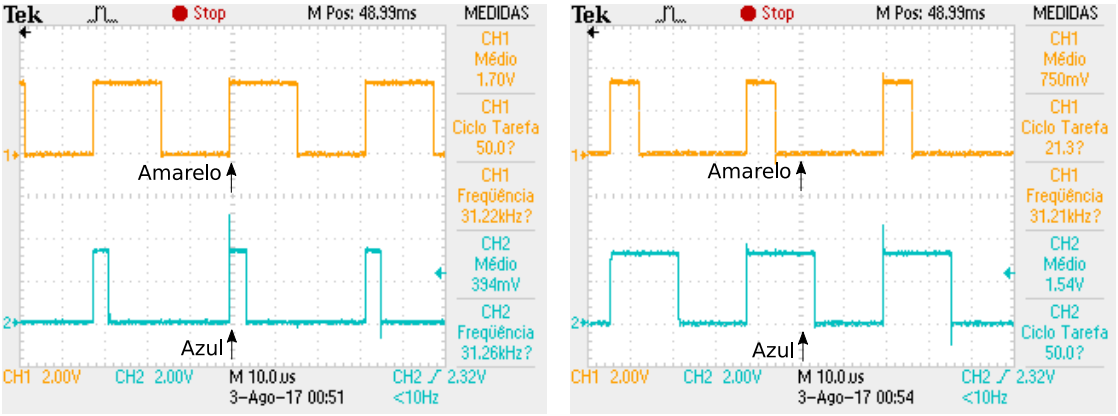


Figura 42 – Protótipo 3 confeccionado
Fonte: Autoria própria

A troca dos sensores laterais QRE1113 pelos QTR-1A proporcionaram uma melhora significativa na detecção de marcas laterais. O terceiro protótipo também foi o qual apresentou o melhor desempenho, conseguindo completar uma volta na pista de testes com um PWM de 37% (a versão anterior, CrazyFrog 2, teve o melhor desempenho com um PWM de pouco mais de 30%).

As Figuras 43a e 43b mostram a saída do controlador contínuo para duas situações, em que o canal em amarelo representa o motor da direita e o azul, o da esquerda. Na Figura 43a, é mostrado o robô saindo da linha branca pelo lado direito, ou seja, a parte mais a direita do veículo está saindo da linha, sendo necessário aumentar o *duty cycle* do motor da direita. Na Figura 43b, é mostrada a saída do controlador com o protótipo saindo para o lado esquerdo, de forma que a tensão aplicada no motor da esquerda deve ser maior do que o da direita.



(a) **(b)**

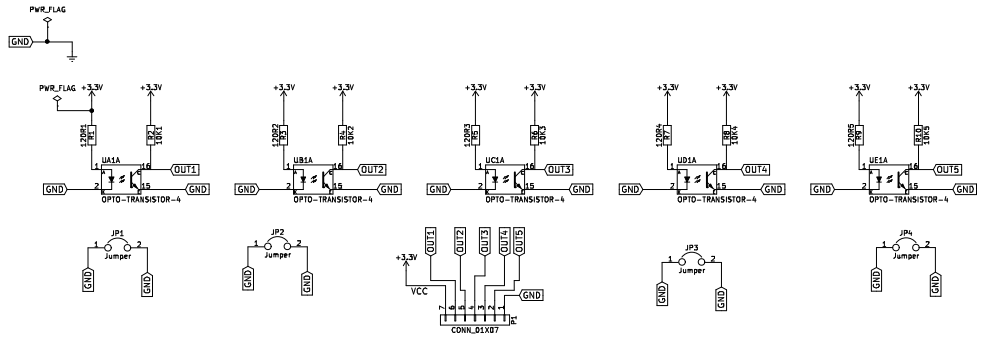
Figura 43 – Veículo saindo da linha branca (Amarelo, motor da direita; Azul, motor da esquerda): (a) Pelo lado direito; (b) Pelo lado esquerdo
Fonte: Autoria própria.

Para que o *kit* de desenvolvimento possa ser utilizado corretamente no protótipo CrazyFrog 3, se faz necessário remover o *solder bridge* SB18 (uma conexão feita com um resistor de 0 Ω), pois este conector, quando soldado no *kit*, altera a conexão do microcontrolador com a barra de pinos. Nesse caso, o pino PA5 é desconectado da placa, precisando ser configurado como entrada flutuante (STMICROELECTRONICS, 2016). No entanto, PA5 é utilizado para as leituras do ADC, sendo de extrema importância a utilização deste pino.

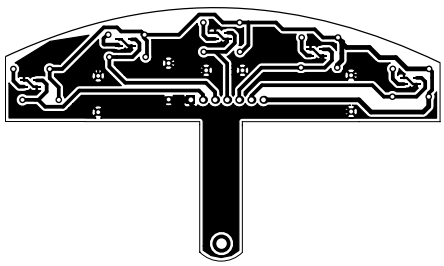
6.1.4 BARRA FRONTAL DE SENSORES

A Barra Frontal dos sensores foi o circuito mais simples de ser projetado, pois contém somente os sensores de refletância, resistores e um conector, conforme pode ser visto no esquemático do circuito, na Figura 44a. Esta barra contém cinco sensores e possui um comprimento de aproximadamente 8 cm (oito centímetros), sendo capaz de identificar a linha frontal da WinterChallenge, a qual possui 2cm (dois centímetros) de comprimento ((ROBOCORE, 2016c)). A placa pode ser vista na Figura 44b.

Foram feitas duas placas, sendo que a única diferença entre estas foi uma mudança no *footprint* (modelo do componente na placa) da segunda placa. Na primeira placa, ocorreu também o espelhamento dos *footprints* do QRE1113, sendo necessário fazer ajustes para que estes pudessem ser devidamente soldados na placa. No entanto, esses ajustes traziam risco aos sensores, os quais acabaram queimando com certa frequência. Com base nisso que foi desenvolvida a segunda barra de sensores, a qual pode ser vista na Figura 44c.



(a)



(b)

(c)

Figura 44 – Barra de sensores: (a) Esquemático; (b) Desenho da placa; (c) Barra frontal de sensores
Fonte: Autoria própria.

6.2 PARTICIPAÇÃO EM COMPETIÇÕES

Durante o andamento desse trabalho, aconteceram participações em dois eventos de competição de robótica móvel: O WinterChallenge, em São Caetano do Sul, em julho de 2017, e a FACE, em Chapecó, em setembro do mesmo ano. Ambas as competições foram disputadas com a segunda versão desenvolvida, CrazyFrog 2.

As duas competições tiveram um impacto significativo no desenvolvimento desse trabalho. Na WinterChallenge, no segundo dia de competição, um dos sensores da barra frontal

foi danificado, sendo necessária a sua substituição. Após o sensor ter sido trocado da placa, o veículo já não se comportava mais da mesma forma, não conseguindo seguir a linha com os mesmos parâmetros do PD que antes funcionava adequadamente. Esse fato, juntamente com a conversa com integrantes de outras equipes deste segmento, indicaram a possibilidade de ter uma importante diferença na resposta dos sensores QRE1113 (o novo sensor da placa foi adquirido da China, enquanto que os restantes foram adquiridos dos Estados Unidos). Esse fato será abordado na seção posterior deste capítulo.

Na mesma competição,junto com o problema dos sensores de refletância da barra frontal, os *encoders* e os sensores para marcações laterais não estavam funcionando corretamente, o que levou a uma grande perda no desempenho no veículo. Ainda assim, o CrazyFrog 2 conseguir atingir a 28^a (vigésimo e oitava) colocação dentre os 37 (trinta e sete) competidores que finalizaram ao menos uma volta, na categoria Seguidor de Linha Pro (ROBOCORE, 2017).

A Figura 45 mostra a pista utilizada na categoria Seguidor de Linha Pro, com Márcio Petry em destaque, o qual foi o autor de (PETRY, 2016). Na pista também nota-se um empecilho, chamado de ”curva do ’S’”, o que já foi explicado na seção referente ao controlador SED.

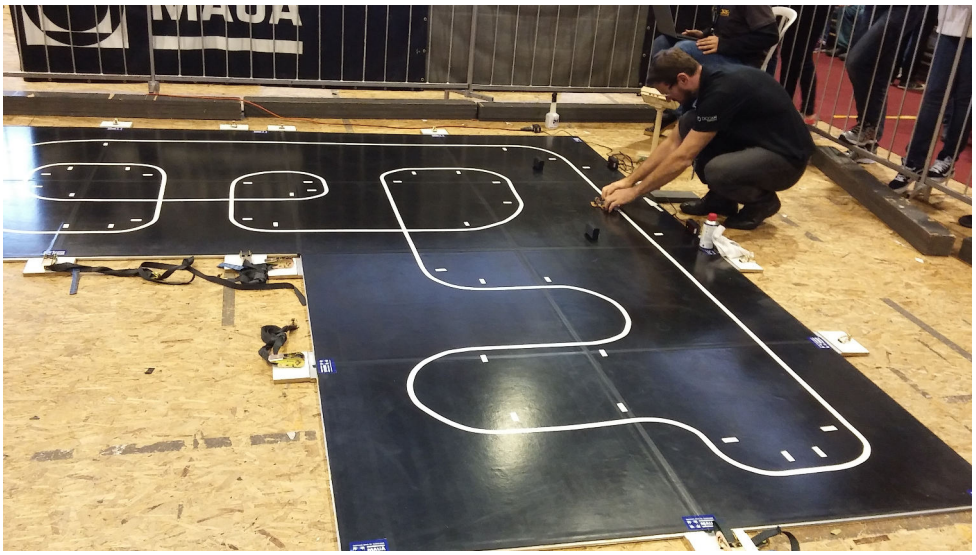


Figura 45 – Pista do Seguidor de Linha Pro na WinterChallenge 2017
Fonte: Autoria própria

Na FACE 2017, os problemas foram minimizados mas ainda assim o veículo não obteve um desempenho desejável, com os sensores laterais e *encoders* ainda falhando, com a barra frontal de sensores também tendo uma resposta inferior à antes da WinterChallenge. O robô conseguiu terminar a competição na 5^a colocação. No entanto, por ser uma competição muito menor do que a WinterChallenge e consequentemente de pouca abrangência, não foram encontradas fontes (referência) para sustentar a colocação obtida.

6.3 TESTE DE RESPOSTA DOS SENSORES DE REFLETÂNCIA

Durante a participação da competição WinterChallenge 2017, após a substituição de um sensor de refletância QRE1113 por um outro do mesmo modelo, mas adquirido de outro estabelecimento, verificou-se uma diferença significativa entre estes dispositivos. Antes da substituição, os parâmetros do controlador PID funcionavam de maneira satisfatória para a velocidade na qual o veículo estava submetido; após a troca do componente, o carro não conseguia acompanhar a linha com a mesma velocidade e parâmetros. Assim, notou-se uma possível discrepância entre a velocidade de resposta destes sensores, argumento este que foi complementado ao conversar com competidores da equipe de outras equipes, os quais notaram uma diferença entre estes mesmos sensores.

Com base no ocorrido, realizou-se um teste para analisar a velocidade de resposta de diferentes tipos de sensores de refletância disponíveis na universidade. Foram utilizados quatro (4) sensores para este teste:

- QTR-1A, que é uma placa da Pololu, a qual contém um QRE1113;
- QTR-8A, que é um placa da Pololu, que contém oito sensores QRE1113;
- QRE1113, adquirido da (ROBOCORE, 2016a) com recursos da universidade;
- QRE1113 adquirido da empresa Arrow, dos Estados Unidos e utilizado na confecção da primeira barra frontal de sensores;
- QRE1113 adquirido pelo Aliexpress, da China e utilizado na confecção da segunda barra frontal de sensores;
- QRD1114, o qual foi emprestado de um aluno da universidade;

Os gráficos foram gerados pelo Matlab versão 2013b, um *software* proprietário, o qual a Universidade Tecnológica Federal do Paraná possui alguma licenças.

6.3.1 PROCEDIMENTO

Para obter corretamente o tempo de resposta, foi projetado um mecanismo que provoca um mesmo sinal de entrada para todos os sensores analisados. Este mecanismo consistiu de uma folha de papel, tamanho A4, e um servo motor SG90. O sistema funciona da seguinte maneira: o servo motor, com uma hélice na ponta, está conectado por um fio à folha, a qual contém algumas tiras de fita isolante preta; ao ser acionando por um Arduino, o servo motor

puxa a folha e o sensor fica sujeito à mudança que ocorre na folha, entre a cor branca original desta e o preto da fita isolante. Essa transição é capturada por um osciloscópio, o qual está configurado no modo *single shot* (única captura). O servo puxa o papel por aproximadamente 2,4 cm (centímetros). Todos os sensores utilizados neste teste eram os encontrados no meio de cada placa, de modo a ficarem dispostos na metade da folha, de forma a proporcionar a mesma transição para cada um dos dispositivos. Esse sistema pode ser visto na Figura 46.

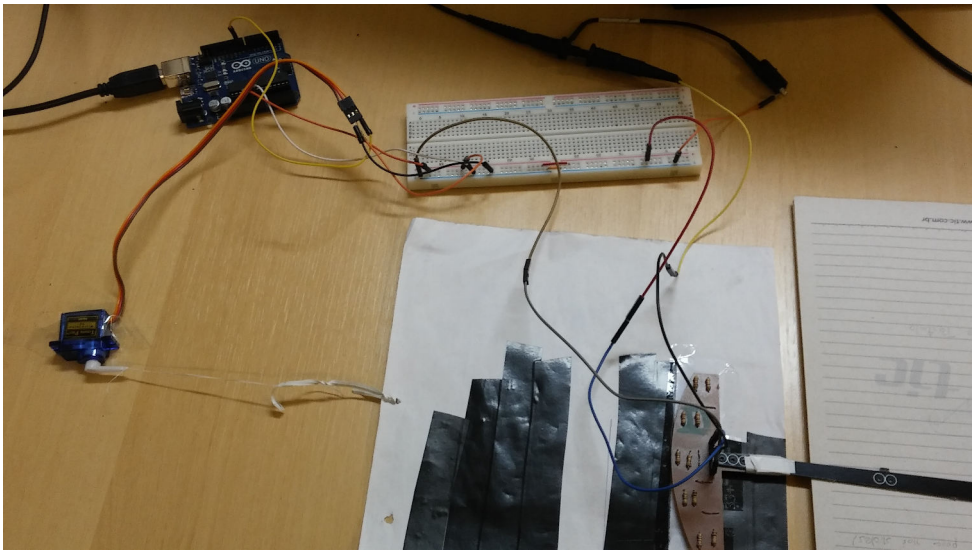


Figura 46 – Mecanismo para aquisição do tempo de resposta dos sensores
Fonte: Autoria própria

6.3.2 TEMPO DE RESPOSTA DOS SENSORES

Pelo *datasheet* do componente QRE1113, verifica-se que a melhor distância possível para a resposta do sensor é de aproximadamente de 0,65 mm (milímetros), mas obtém-se uma resposta satisfatória a 1 mm de distância do objeto refletivo (SEMICONDUCTOR, 2016). O *datasheet* do QRD1114 já não traz nenhuma informação a respeito da resposta do sensor em relação à distância deste ao objeto refletivo (SEMICONDUCTOR, 2000).

Com base nisso, foram obtidos dois tempos de resposta para cada sensor: o primeiro com o componente localizado a 1 mm de distância da folha e o segundo com este localizado a 2 mm, os quais podem ser vistos nas Figuras **??** e 48, respectivamente. O ajuste de altura dos sensores foi feito com base em folhas de papel, de modo que cada dispositivo fique na altura especificada.

O critério utilizado na avaliação da resposta do sensores foi a obtenção do T_s (Tempo de assentamento) de cada sensor. Desta forma, os valores informados nas Figuras **??** e 48 são referentes ao tempo de assentamento de cada um dos componentes analisados.

No teste feito com uma distância de 1 mm, o qual pode ser visto na Figura **??**, nota-se uma aparente diferença entre a respostas obtidas. A resposta vista pelo QRE1113 do Aliexpress, em 47a, com T_s de 39,6 ms, é quase 10 ms mais lenta do que a resposta mostrada pelo QRE1113 obtido da Arrow, visto na Figura 48d e com T_s de 30 ms; componentes estes que foram utilizados, respectivamente, na segunda e na primeira barra frontal de sensores. O sensor QRD1114, em que sua resposta pode ser vista na Figura 47e, apresenta uma velocidade intermediária entre as respostas anteriores e o sensor QTR-1A, da 47f, apresenta a melhor resposta dentre os dispositivos analisados, com T_s de 28 ms (uma resposta cerca de 70% mais rápida do que o QRE1113 da Aliexpress, o sensor atualmente utilizado no robô). O sensor da Robocore apresenta uma resposta ligeiramente melhor do que a apresentada pelo QRE1113 do Aliexpress e o QTR-8A tem uma resposta muito próximo do QTR-1A.

No teste feito a uma distância de 2 mm da base refletora, o qual pode ser visto na Figura 48, verifica-se que todos os sensores tem uma grande redução no tempo de resposta.

Essa análise proporcionou um resultado importante, visto que o mesmo modelo de sensor (QRE1113), ao ser adquirido de três locais diferentes, apresenta características muito distintas, o que acarreta em um efeito direto no desempenho do veículo desenvolvido. Também é visto que para obter um bom desempenho com os sensores, independente do qual seja utilizado, é necessário que este esteja a uma altura adequada em relação ao solo.

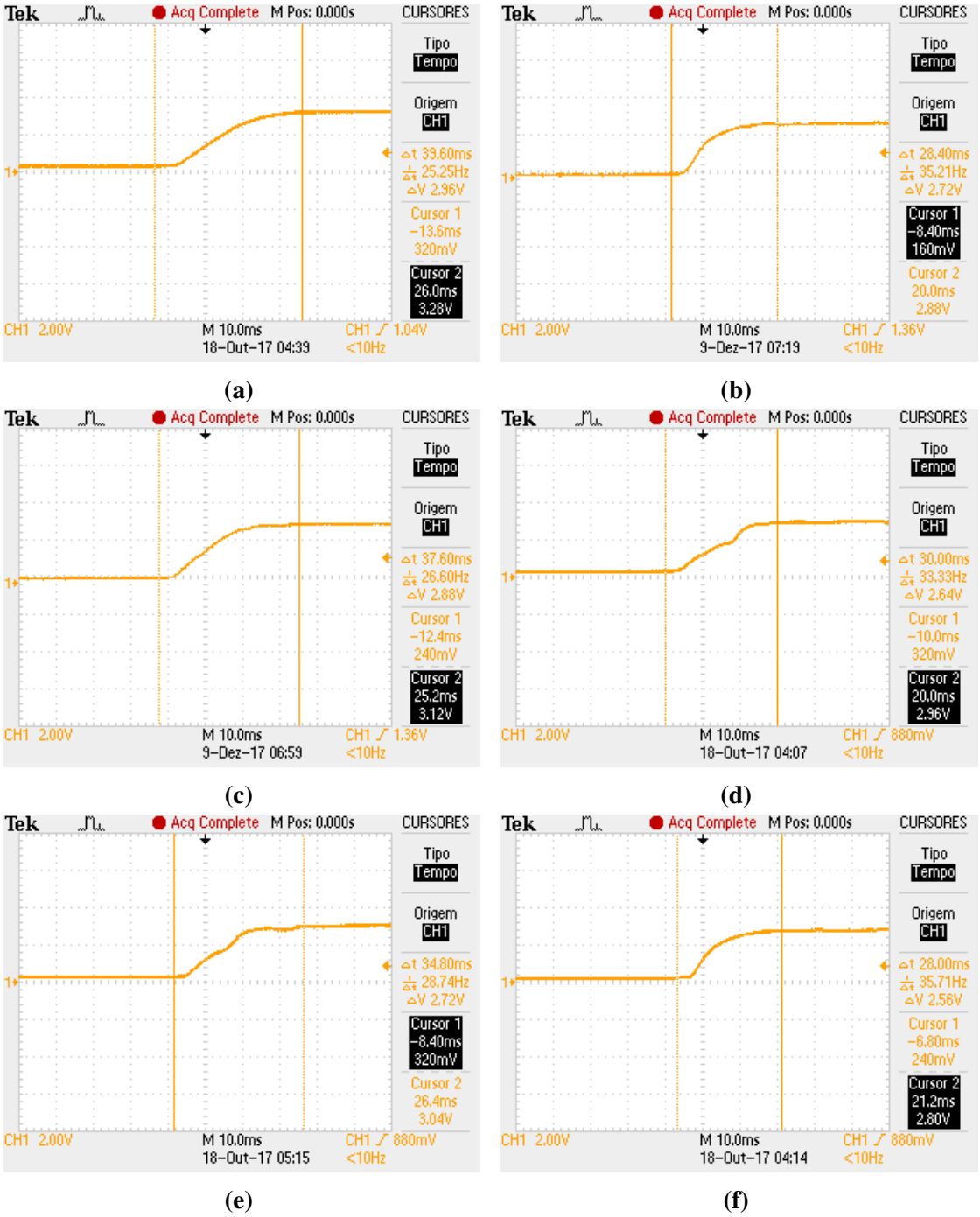


Figura 47 – Resposta do sensores a 1 mm: (a) QRE1113 do Aliexpress; (b) QTR-8A; (c) QRE1113 da Robocore; (d) QRE1113 da Arrow; (e) QRD1114; (d) QTR-1A
Fonte: Autoria própria

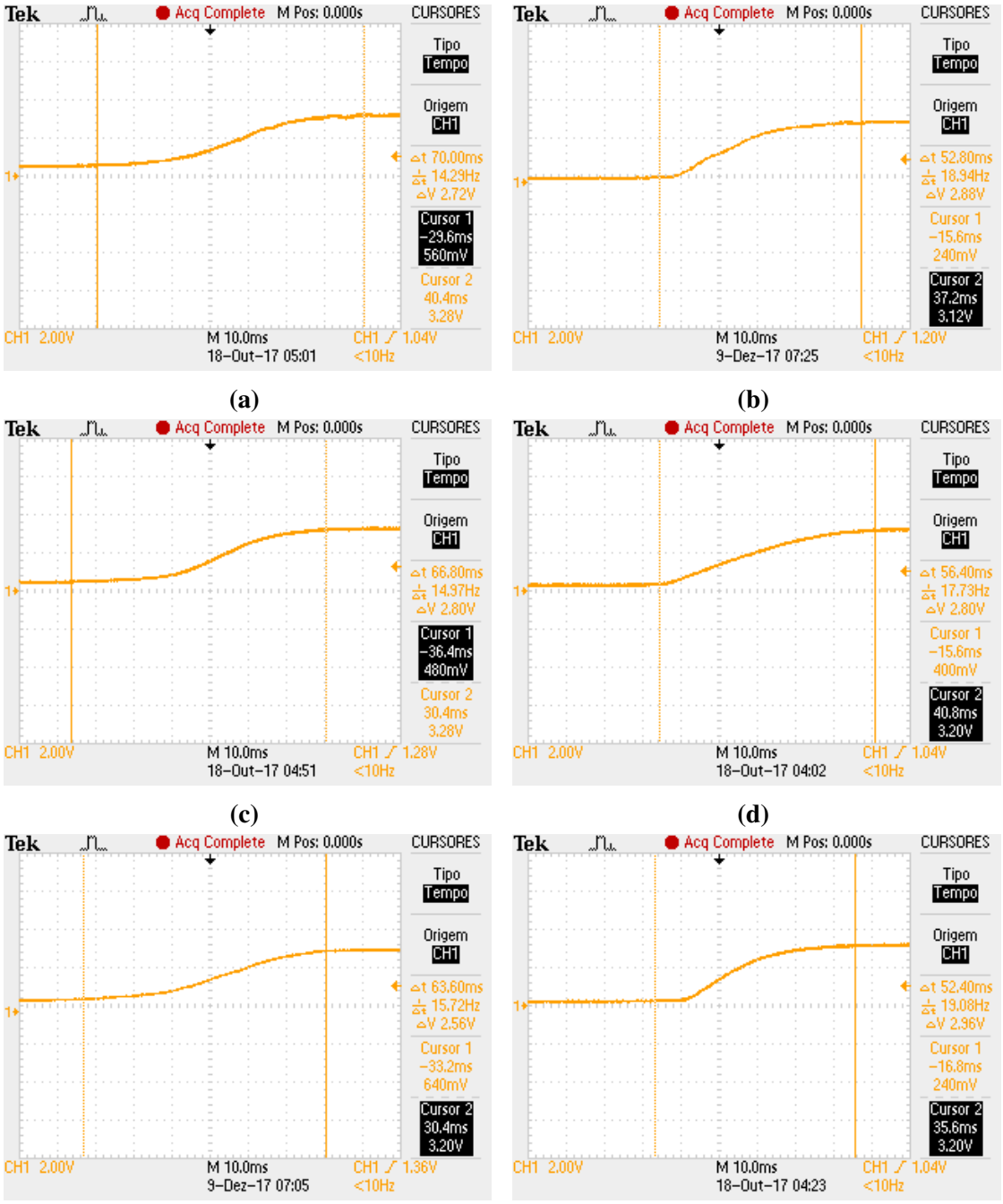


Figura 48 – Resposta do sensores a 2 mm: (a) QRE1113 do Aliexpress; (b) QTR-8A; (c) QRE1113 da Robocore; (d) QRE1113 da Arrow; (e) QRD1114; (d) QTR-1A
Fonte: Autoria própria

7 CONCLUSÃO E TRABALHOS FUTUROS

Este trabalho teve como objetivo o desenvolvimento de um robô autônomo seguidor de linha, utilizando controle híbrido, composto por um controlador de tempo contínuo e um controlador de tempo discreto.

O controlador de Sistemas a Eventos Discretos (SED), foi modelado graficamente pelo *software* Supremica e o teve o seu código gerado pelo *software* Deslab, no qual foi modelado por um autômato de Moore, em que cada transição de estados é feita por um evento não-controlável que foi gerado e a ação referente à este estado acontece no estado seguinte. O SED foi utilizado para detectar as marcações laterais da pista e com base nessas informações, foi feito um mapeamento de pista, o qual consiste em um vetor de inteiros e que contém os valores do *encoder* para cada início e fim de curva e a quantidade total de curvas no percurso. Nos testes realizados, a pista pode ser mapeada corretamente, com os valores dos *encoders* sendo adquiridos e armazenados no vetor satisfatoriamente.

A função de transferência da planta do veículo foi obtida através da aquisição dos valores da posição do robô, calculado a partir da média ponderada dos valores dos sensores de refletância da barra frontal, a medida em que este se move longitudinalmente pela faixa branca. Para obter a planta, foi utilizado o *software* Matlab, a partir do qual foram geradas funções de transferência e comparadas com a função obtida da planta do veículo, que foi a derivada numérica do vetor de posição, acarretando assim em um vetor de velocidade. Devido à imprecisão dos sensores, o modelo do processo obtido experimentalmente não ficou com uma representatividade adequada, o qual fez que os valores teóricos obtidos para os ganhos do controlador tenham sido readequados experimentalmente. Para tanto, os parâmetros do controlador contínuo da planta foram obtidos manualmente. O controlador contínuo utilizado foi o Proporcional-Derivativo (PD), escolhido devido à planta já possuir um integrador. Esse algoritmo é bastante utilizado comercialmente e foi feito com base na literatura encontrada. Inicialmente previa-se projetar os parâmetros do controlador PD de acordo com a planta obtida, no entanto essa etapa não foi feita devido à restrições no tempo de execução desta parte.

Neste trabalho foram desenvolvidos três robôs, sendo que o segundo participou de duas competições de robótica móvel: a FACE, em Chapecó, e a WinterChallenge, em São Caetano do Sul. Durante a participação no último evento, notou-se a diferença na qualidade entre sensores de refletância QRE1113 adquiridos de dois lugares diferentes. Com base nisso, foi feito um teste comparativo entre seis sensores de refletância disponíveis na UTFPR. Para esse teste foi

montado um procedimento com um servo motor que, quando acionado, puxa uma folha de papel com uma fita isolante preta, a qual provoca o mesmo sinal de entrada nos sensores. Com esse teste, verificou-se que existe uma diferença considerável entre estes dispositivos, o que pode ocasionar perdas significativas no desempenho do veículo. Foi inicialmente proposta a telemetria do veículo, no entanto esta não foi finalizada, com falhas no aplicativo do dispositivo móvel e na recepção da comunicação pelo *bluetooth*.

Este trabalho teve resultados relevantes e que ajudarão no projeto de futuros robôs seguidores de linha desenvolvidos na universidade. Dentre esses resultados, os que se destacam são a inversão de giro dos motores, que permitem a trava das rodas nas curvas, como um freio; e a teste dos sensores de refletância, em que pode ser visto que a procedência dos sensores utilizados é fundamental para o desempenho do veículo, visto que os componentes apresentam tempos de resposta diferentes. Esses resultados somente foram alcançados pela participação em competições de robótica móvel, as quais proporcionam um ambiente de troca de conhecimentos entre as equipes e permitem verificar todas as deficiências e avanços obtidos nos veículos desenvolvidos.

Devido aos problemas encontrados, não se obteve o desempenho esperado para o veículo, o qual apresentou um desempenho um pouco inferior aos robôs desenvolvidos por (PETRY, 2016). Os sensores da barra frontal ocasionaram uma perda de desempenho significativa nesse trabalho, em que não teve tempo suficiente e condições para testes com a barra QTR-8A, a qual tem um desempenho excelente, igual aos obtidos pela placa QTR-1A. No entanto, assim como o de (PETRY, 2016) serviu para esse trabalho, espera-se que esse possa servir como uma base para trabalhos posteriores, principalmente pelas alterações feitas no *hardware*, com o mapeamento de pista e os testes dos sensores de refletância.

Para trabalhos futuros, ficam algumas sugestões, como: utilizar um giroscópio para identificar a intensidade da curva e fazer o robô acelerar com diferentes velocidades para cada uma; fazer um sistema de telemetria que possa alterar os parâmetros dos robôs e receber informações que possam identificar a configuração da pista, auxiliando toda a equipe; melhoria do sistema de mapeamento, utilizando algum método de inteligência artificial para o correto controle do veículo; e principalmente a alteração da barra frontal por uma barra QTR-8A, da Pololu, de forma a melhor a leitura da linha. O terceiro protótipo possui os furos necessários para a instalação dessa placa, necessitando somente a construção de um novo conector.

REFERÊNCIAS

ANSI/RIA.R15.06-1999. **Robot Terms and Definitions**. 2010. Disponível em: <http://www.robotics.org/product-catalog-detail.cfm/productid/2953>. Acesso em: 15 ago. 2016.

ARAÚJO, F. M. U. de. **Sistemas de Controle**. Natal: DCA-UFRN, 2007. Apostila.

AZEVEDO, F. **Derivaćão e Integraćão Numéric**. 2017. Disponível em: <http://www.mat.ufrgs.br/˜fabio/der_int.pdf>. Acesso em: 01 set. 2017.

BRANICKY, M. S.; BORKAR, V. S.; MITTER, S. K. A unified framework for hybrid control: Model and optimal control theory. **IEEE Transaction on Automatic Control**, IEEE, p. 31–45, 1998. Disponível em: <http://eprints.iisc.ernet.in/1453/1/unified.pdf>.

CASSANDRAS, C. G.; LAFORTUNE, S. **Introduction to Discrete Event Systems**. 2. ed. New York: Springer US, 2008.

DUDEK, G.; JENKIN, M. **Computational principles of mobile robotics**. 2. ed. [S.l.]: Cambridge University Press, 2010.

DUNN, W. C. **Introduction to Instrumentation, Sensors and Process Control**. Norwood: Artech House, 2006. (Artech House sensors library).

DYNAMICS, B. **LS3 - Legged Squad Support System**. 2016. Disponível em: <http://www.bostondynamics.com/robot_ls3.html>. Acesso em: 25 set. 2016.

FRADEN, J. **Handbook of Modern Sensors: Physics, Designs, and Applications**. 3. ed. New York: Springer New York, 2006.

FREITAS, R. S. de. **Arquitetura híbrida e controle de missão de robôs autônomos**. Dissertação (Mestrado) — Universidade Federal do Rio de Janeiro, Rio de Janeiro, 2016.

GUADAGNIN, A. J. **Controle Híbrido de um Robô Seguidor de Linha**. 2014. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

HEINEN, F. J. **Sistema de Controle híbrido para Robôs Móveis Autônomos**. Dissertação (Mestrado) — Universidade do Vale do Rio dos Sinos, São Leopoldo, 2002.

HIRAI, A. **Cartisx04**. 2014. Disponível em: <http://anikinonikki.cocolog-nifty.com/blog/2014/11/cartisx04.html>. Acesso em: 13 out. 2016.

HIRAI, A. **All Japan Micro Mouse 2015 Robotrace Three Consecutive Victories**. 2016. Disponível em: <http://anikinonikki.cocolog-nifty.com/blog/2015/11/2015-8a43.html>. Acesso em: 08 nov. 2016.

MAUÁ, I. **Instituto Mauá de Tecnologia sedia o evento de Robótica Winter Challenge 2016**. 2016. Disponível em: <http://maua.br/imprensa/press-releases/instituto-maua-tecnologia-sedia-evento-robotica-winter-challenge-2016>. Acesso em: 23 ago. 2016.

NASA. 1997. Disponível em: <https://mars.jpl.nasa.gov/MPF/rover/sim2.jpg>. Acesso em: 24 set. 2016.

NASA. **Mars Pathfinder/ Sojourner Rover**. 1997. Disponível em: <http://www.jpl.nasa.gov/missions/mars-pathfinder-sojourner-rover>. Acesso em: 24 set. 2016.

NISE, N. S. **Engenharia de Sistemas de Controle**. 6. ed. Rio de Janeiro: LTC, 2012.

OGATA, K. **Engenharia de Controle Moderno**. 5. ed. São Paulo: Pearson Prentice Hall, 2010.

PESSIN, G. **Estratégias inteligentes aplicadas em robôs móveis autônomos e em coordenação de grupos de robôs**. Dissertação (Mestrado) — Universidade de São Paulo, São Carlos, 2013.

PETRY, M. L. **Controle Híbrido de um robô autônomo seguidor de linha**. 2016. Trabalho de Conclusão de Curso. Universidade Tecnológica Federal do Paraná.

POLOLU. **Pololu magnetic encoder datasheet**. 2016. Disponível em: <https://www.pololu.com/product/3081>. Acesso em: 01 set. 2016.

POLOLU. **Pololu micromotor datasheet**. 2016. Disponível em: <https://www.pololu.com/product/3048>. Acesso em: 01 set. 2016.

RECORDS, G. W. **Robotics tournament VEX Worlds is named largest in the world after 1,075 teams take part**. 2016. Disponível em: <http://www.guinnessworldrecords.com/news/2016/4/robotics-tournament-vex-worlds-is-named-largest-in-the-world-after-1-075-teams-ta-426576>. Acesso em: 29 ago. 2016.

ROBOCORE. 2016. Disponível em: <https://www.robocore.net/eventos>. Acesso em: 22 ago. 2016.

ROBOCORE. **Classificação WinterChallenge**. 2016. Disponível em: <https://www.robocore.net/modules.php?name=GR_Eventos&evento=24&tab=2>. Acesso em: 23 ago. 2016.

ROBOCORE. **Regras Seguidor de Linha**. 2016. Disponível em: <https://www.robocore.net/upload/attachments/robocore_regras_seguidor_de_linha_108.pdf>. Acesso em: 28 ago. 2016.

ROBOCORE. **Classificać/ ao WinterChallenge 2017**. 2017. Disponível em: <https://www.robocore.net/eventos/wc13/2>. Acesso em: 24 set. 2016.

ROBOCUP. 2016. Disponível em: <http://www.robocup.org/about-robocup/objective>. Acesso em: 22 ago. 2016.

ROBOGAMES. 2016. Disponível em: <http://robogames.net/index.php>. Acesso em: 29 ago. 2016.

SECCHI, H. A. **Uma introdução aos robôs móveis**. Serra: NERA-IFES, 2012. Traduzido do original *Una Introducción a los Robots Móviles*.

SEDRA, A. S.; SMITH, K. C. **Microelectronic Circuits**. 6. ed. [S.l.]: Oxford University Press, 2010.

SEMICONDUCTOR, A. . O. **30V P-Channel MOSFET**. 2011. Disponível em: <http://www.aosmd.com/pdfs/datasheet/ao3407.pdf>. Acesso em: 13 nov. 2017.

SEMICONDUCTOR, F. **QRD1113/QRD1114 Reflective Object Sensor**. 2000. Disponível em: <https://www.sparkfun.com/datasheets/BOT/QRD1114.pdf>. Acesso em: 31 out. 2017.

SEMICONDUCTOR, F. **Miniature Reflective Object Sensor**. 2016. Disponível em: <http://cdn.sparkfun.com/datasheets/Sensors/Proximity/QRE1113.pdf>. Acesso em: 01 set. 2016.

SIEGWART, R.; NOURBAKHSH, I. R.; SCARAMUZZA, D. **Introduction to autonomous mobile robots**. Cambridge: The MIT Press, 2011.

SPARKFUN. **Motor driver - Dual TB6612FNG**. 2017. Disponível em: <https://www.sparkfun.com/products/9457>. Acesso em: 12 abril 2017.

STMICROELECTRONICS. **STM32F303x6/x8 datasheet**. 2015. Disponível em: <http://www.mouser.com/ds/2/389/DM00092070-524505.pdf>. Acesso em: 01 set. 2016.

STMICROELECTRONICS. **UM1956 User manual**. 2016.

SWITCH, S. Disponível em: <https://cdn.sparkfun.com//assets/parts/5/8/8/3/10860-01.jpg>. Acesso em: 13 nov. 2017.

SYSTEM, A. M. **AMS1117 1A Low Dropout Voltage Regulator**. 2017. Disponível em: <http://www.advanced-monolithic.com/pdf/ds1117.pdf>. Acesso em: 12 abril 2017.

TECHNOLOGY, G. H. I. **HC-05 datasheet**. 2016. Disponível em: <http://cdn.sparkfun.com/datasheets/Sensors/Proximity/QRE1113.pdf>. Acesso em: 01 set. 2016.

TEIXEIRA, M. **Explorando o uso de Distinguidores e de Autômatos Finitos Estendidos na teoria do Controle Supervisório de Sistemas a Eventos Discretos**. Tese (Doutorado) — Universidade Federal de Santa Catarina, Florianópolis, 2013.

TORRICO, C. R. C. **Controle Supervisório Hierárquico de Sistemas a Eventos Discretos: Uma Abordagem Baseada na Agregação de Estados**. Tese (Doutorado) — Universidade Federal de Santa Catarina, Florianópolis, 2003.

TORRICO, C. R. C. **Deslab**. 2015. Disponível em: <https://sites.google.com/site/controlediscreto9/instaladores/DESLAB3.6.rar?attredirects=0&d=1>. Acesso em: 10 nov. 2015.

TOSHIBA. **Driver IC for Dual DC motor**. 2016. Disponível em: <https://www.pololu.com/file/0J86/TB6612FNG.pdf>. Acesso em: 01 set. 2016.

WORLDS, V. 2016. Disponível em: <http://www.roboticseducation.org/competition-teams/vex-robotics-competition/>. Acesso em: 29 ago. 2016.

XLSEMI. **400KHz 60V 4A Switching Current Boost / Buck-Boost / Inverting DC/DC Converter**. 2016. Disponível em: <https://www.pollin.de/shop/downloads/D351434D.PDF>. Acesso em: 01 set. 2016.

APÊNDICE A – CÓDIGO DO MICROCONTROLADOR STM32F303K8

```
/* Includes
   ------------------------------------------------------------------------*/
#include "main.h"
#include "stm32f3xx_hal.h"

// ////////////////////////////////////////// INCLUDES
   //////////////////////////////////////////////
/* USER CODE BEGIN Includes */
#include "Flash.c"
/* USER CODE END Includes */

ADC_HandleTypeDef hadc1;
ADC_HandleTypeDef hadc2;
DMA_HandleTypeDef hdma_adc1;

TIM_HandleTypeDef htim1;
TIM_HandleTypeDef htim2;
TIM_HandleTypeDef htim3;
TIM_HandleTypeDef htim6;
TIM_HandleTypeDef htim7;
TIM_HandleTypeDef htim16;

UART_HandleTypeDef huart1;

// ////////////////////////////////////////// DEFINES
   //////////////////////////////////////////////

#define CurveLineP 4.75
#define CurveLineI 0
#define CurveLineD  0
#define CurveLineSpeed 960

#define StraightLineP  7
#define StraightLineI  0
#define  StraightLineD  50
#define  StraightLineSpeed  1500
```

```
// Para salvar no lugar certo do vetor
#define ENC_DIR_INI 1
#define ENC_DIR_FIM 2
#define ENC_ESQ_INI 3
#define ENC_ESQ_FIM 4

//////////////////////////////////////// VARIAVEIS
    ////////////////////////////////////////////

uint16_t MapBuff[4*MAPSIZE+1]; // Armazena todas as curvas possiveis
uint8_t CALIBRACTR;
uint32_t pos_divisor;
 //ADC_buffer[0] -> ADC1_IN3
 //ADC_buffer[1] -> ADC2_IN1
 //ADC_buffer[2] -> ADC1_IN4
 //ADC_buffer[3] -> ADC2_IN2
 //ADC_buffer[4] -> Vazio apenas para completar o nmero par
 //ADC_buffer[5] -> ADC2_IN2
uint16_t ADC_buffer[6];
int qMap=0;
int END_OF_PATH= 0; // Flag para indicar o fim de percurso
 // TypeLine e a variavel responsavel para dizer ao PID se o controlador
    esta na reta ou na curva.
 // Se TypeLine = 0, entao o PID segue os parametros da Curva;
 // Se TypeLine = 1, logo o PID segue os parametros da Reta.
uint8_t TypeLine;
 // TypeSED e a variavel responsavel pela verificacao se o metodo utilizado
     sera o mapeamento ou o SED, para mapear a pista
 // Se TypeSED = 0, entao sera utilizado o mapemanto, gravado na Flash do
    ucon;
 // Se TypeSED = 1, entao sera utilizado o SED, para fazer o mapeamento e
    grava-lo na Flash.
uint8_t TypeSED;
float posicao = 0;
int16_t velocidade=0;
int16_t proporcional_passado = 0;
uint32_t ADCSensors[5];
unsigned long int B[5] = {0};
unsigned int max[5] = {0};
unsigned int min[5] = {300,300,300,300,300};
char statusADC=0;
float Kp, Ki, Kd;
```

```
uint8_t MANDA=0; //Utilizado para obter o vetor de posicoes
uint8_t curve=0; // Utilizado para saber se foi uma curva no Estado 4 do
    SED

// PARTE DO SED ABAIXO

//Dados do automato (Nao pode ser declarado dentro da funcao main por ser
    const)
#define NTRANS 16 //Numero de Transies
#define NESTADOS 12 //Numero de Estados
#define BUFFER 10   //Maximo Numero de Eventos no Buffer

//Dados do autmato (No pode ser declarado dentro da fun  main por ser
    const)
#define NTRANS 16 //Nmero de Transies
#define NESTADOS 12 //Nmero de Estados
#define BUFFER 10   //Mximo Nmero de Eventos no Buffer

const unsigned int event[NTRANS]={2,2,4,6,8,10,8,10,4,10,6,6,10,8,6,4};
const unsigned int in_state[NTRANS]={1,2,3,4,5,6,7,4,9,8,4,10,4,9,1,11};
const unsigned int rfirst[NESTADOS] = {1,2,3,4,16,9,7,8,11,12,13,15};
const unsigned int rnext[NTRANS] = {0,0,0,0,0,0,0,0,6,0,0,10,0,0,14,5};

//mapeamento de eventos nao controlaveis como entradas

//mapeamento de eventos no controlveis como entradas

#define B1 2 //Entrada 0
#define AchaDir 4 //Entrada 1
#define PerdeDir 6 //Entrada 2
#define AchaEsq 8 //Entrada 3
#define PerdeEsq 10 //Entrada 4

unsigned char buffer[BUFFER];  //Buffer para armazenar a fila de enventos
    externos

unsigned char n_buffer=0;  //Numero de eventos no Buffer

unsigned char n_buffer=0;  //Nmero de eventos no Buffer


unsigned int k;
char occur_event;   //Evento ocorrido
```

```
unsigned char current_state = 0; //Estado atual inicializado com estado
    inicial
char g=0;     //Flag para gerador aleatorio de eventos

char gerar_evento=1;    //Flag para habilitar a temporiza de eventos
    controlaveis
char moore_output = 0;  //Inicializa saida periferica

char gerar_evento=1;    //Flag para habilitar a temporiza de eventos
    controlveis
char moore_output = 0;  //Inicializa sada perifrica




/* Private function prototypes
    -----------------------------------------------*/
void SystemClock_Config(void);
static void MX_GPIO_Init(void);
static void MX_DMA_Init(void);
static void MX_ADC2_Init(void);
static void MX_ADC1_Init(void);
static void MX_TIM1_Init(void);
static void MX_TIM7_Init(void);
static void MX_TIM6_Init(void);
static void MX_TIM3_Init(void);
static void MX_TIM2_Init(void);
static void MX_USART1_UART_Init(void);
static void MX_TIM16_Init(void);

void HAL_TIM_MspPostInit(TIM_HandleTypeDef *htim);


// ////////////////////////////// PROTOTIPOS DE FUNCOES
    ////////////////////////////////////////////
void SED(uint16_t countRight, uint16_t countLeft);
void FollowMap(uint16_t countLeft, uint16_t countRight);
void setMotors(char Left, char Right);
void controlInterrupts(uint8_t type, uint8_t peripheral);
void EXTI0_IRQHandler(void);
void EXTI9_5_IRQHandler(void);
void EXTI1_IRQHandler(void);
void pdcontrol();
```

```
void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc);
void CalculaPosicao(void);
void CalibraSensors(void);
void blink(uint8_t No, uint32_t ms);
void SendPosition();
void MapHandler();
void initMap(void);

//////////////////////////////////////// MAIN
    ////////////////////////////////////////////////////

void main(void)
{

  /* Reset of all peripherals, Initializes the Flash interface and the
     Systick. */
  HAL_Init();

  /* Configure the system clock */
  SystemClock_Config();

  /* Initialize all configured peripherals */
  MX_GPIO_Init();
  MX_DMA_Init();
  MX_ADC2_Init();
  MX_ADC1_Init();
  MX_TIM1_Init();
  MX_TIM7_Init();
  MX_TIM6_Init();
  MX_TIM3_Init();
  MX_TIM2_Init();
  MX_USART1_UART_Init();
  MX_TIM16_Init();

  /* USER CODE BEGIN 2 */
   TypeSED=0; // O programa sempre comeca para mapear a pista
   TypeLine=0; // Para mapear a pista com a velocidade minima
   pos_divisor = 0;
   qMap=0;
   CALIBRACTR = 0; // Permitir a execucao do calibramento 1 vez
   END_OF_PATH = 0;
   uint32_t countRight, countLeft; //Encoders da direita e esquerda
```

```
    HAL_TIMEx_PWMN_Start(&htim1,TIM_CHANNEL_2); // Inicia PWM Ch2
    HAL_TIM_PWM_Start(&htim1,TIM_CHANNEL_4); // Inicia PWM Ch4n
    HAL_ADC_Start(&hadc2);
    // Numero de elementos precisa ser par "6",
    // pois e armazenado de forma interlacada no vetor ADC_buffer
    HAL_ADCEx_MultiModeStart_DMA(&hadc1, (uint32_t*)ADC_buffer, 6);
    HAL_TIM_Base_Start(&htim1);

    while(1)
    {

  /* USER CODE BEGIN 3 */
      countLeft = TIM2->CNT;
      countRight = TIM3->CNT;

      // Se statusADC for setado, esta permitido fazer o calculo do PD
      if(statusADC)
      {
       statusADC=0;
       CalculaPosicao();
       pdcontrol();
      }
      if(!TypeSED)
      {
      // Segue a pista conforme o mapeamento feito
       if(!END_OF_PATH) FollowMap(countLeft,countRight);
      }else
       SED(countLeft,countRight);

    }
  /* USER CODE END 3 */

}

// ////////////////////////////////////////////////// INTERRUPCOES
    ////////////////////////////////////////////

/*
 Esta funcao eh para desabilitar ou habilitar interrupcoes
 type == 0 -> Desabilita a interrupcao
 type == 1 -> habilita a interrupcao
```

```
 peripheral == 0 -> Desabilita EXTI0_IRQn, ou seja, o sensor lateral da
    direita
 peripheral == 1 -> Desabilita EXTI1_IRQn, ou seja, o botao B1
 peripheral == 6 -> Desabilita EXTI9_5_IRQn, ou seja, o sensor lateral da
    esquerda e botao B2
 peripheral == 7 -> Desabilita EXTI9_5_IRQn, ou seja, o sensor lateral da
    esquerda e botao B2*/
void controlInterrupts(uint8_t type, uint8_t peripheral)
{
 if(!type)
 {
  if(peripheral == 0) NVIC_DisableIRQ(EXTI0_IRQn); // Desabilita EXTI0
  else if(peripheral == 1) NVIC_DisableIRQ(EXTI1_IRQn); // Desabilita EXTI1
  else if(peripheral == 6 || peripheral == 7) NVIC_DisableIRQ(EXTI9_5_IRQn)
     ; // Desabilita EXTI6 ou EXTI7
 }
 else
 {
  if(peripheral == 0) NVIC_EnableIRQ(EXTI0_IRQn); // Habilita EXTI0
  else if(peripheral == 1) NVIC_EnableIRQ(EXTI1_IRQn); // Habilita EXTI1
  else if(peripheral == 6 || peripheral == 7) NVIC_EnableIRQ(EXTI9_5_IRQn);
      // Habilita EXTI6 ou EXT7
 }
}

//////////////////////////////////// INTERRUPCOES
   ////////////////////////////////////////////////

void EXTI0_IRQHandler(void)
{
 // Essa interrupcao e para o handler do sensor lateral da esquerda
   if(HAL_GPIO_ReadPin(GPIOB,GPIO_PIN_0) == 1)
   {
    buffer[n_buffer] = AchaEsq;

   }else{
    buffer[n_buffer] = PerdeEsq;
   }

   if(n_buffer < BUFFER-1) n_buffer++;
  /* USER CODE END EXTI0_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_0);
  /* USER CODE BEGIN EXTI0_IRQn 1 */
```

```
  /* USER CODE END EXTI0_IRQn 1 */
}

void EXTI1_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI1_IRQn 0 */
 if(__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_1))
   {
  controlInterrupts(0,1); // Desabilita para nao causar mais interrupcoes
    buffer[n_buffer] = B1;
    if(n_buffer < BUFFER-1) n_buffer++;
   }
  /* USER CODE END EXTI1_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_1);
  /* USER CODE BEGIN EXTI1_IRQn 1 */
  /* USER CODE END EXTI1_IRQn 1 */
}


void EXTI9_5_IRQHandler(void)
{
  /* USER CODE BEGIN EXTI9_5_IRQn 0 */

 // A Interrupcao do Pino 6 eh referente ao Botao B2
     if(__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_6))
   {
     TypeSED = 0;
   }

    // A Interrupcao do Pino 7 eh referente o handler do sensor lateral da
       direita
     if(__HAL_GPIO_EXTI_GET_FLAG(GPIO_PIN_7))
     {
      if(HAL_GPIO_ReadPin(GPIOA,GPIO_PIN_7) == 1)
      {
       buffer[n_buffer] = PerdeDir;

      }else{
       buffer[n_buffer] = AchaDir;
      }
      if(n_buffer < BUFFER-1) n_buffer++;
     }
```

```
  /* USER CODE END EXTI9_5_IRQn 0 */
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_6);
  HAL_GPIO_EXTI_IRQHandler(GPIO_PIN_7);
  /* USER CODE BEGIN EXTI9_5_IRQn 1 */
  /* USER CODE END EXTI9_5_IRQn 1 */
}

//////////////////////////////////// FUNCOES
    ////////////////////////////////////////////////


void blink(uint8_t No, uint32_t ms)
{
 uint8_t x = 0;
 while(x < No)
 {
  HAL_GPIO_TogglePin(GPIOB,GPIO_PIN_3);
  HAL_Delay(ms);
  x++;
 }
}

void CalibraSensors(void)
{
 char i;
 int c;

 CALIBRACTR = 1; // Para nao fazer novamente o calibramento
 uint32_t time, timeTick = HAL_GetTick();

 ///// SENTIDO 1

 // Motor esquerdo para a tras e direito para frente
   setMotors(-1,1);

   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 500);
   __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 500);

   time = HAL_GetTick();
 while(time < timeTick+700)
 {
  if(statusADC)
  {
```

```
  statusADC=0;
  // Para impedir que sejam utilizados valores passados
  for(i=0;i<5;i++)
  {
   if(ADCSensors[i] > max[i]) max[i] = ADCSensors[i];
   if(ADCSensors[i] < min[i]) min[i] = ADCSensors[i];
  }
 }
 time = HAL_GetTick();
}

// Desliga os motores apos fazer uma etapa
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
__HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);

// /////// SENTIDO 2

HAL_Delay(500);

// Motor esquerdo para a frete e direito para tras
setMotors(1,-1);
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 500);
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 500);

  timeTick = HAL_GetTick();
  time = HAL_GetTick();
 while(time < timeTick+1200)
 {
  if(statusADC)
  {
   statusADC=0;
   for(i=0;i<5;i++)
   {
    if(ADCSensors[i] > max[i]) max[i] = ADCSensors[i];
    if(ADCSensors[i] < min[i]) min[i] = ADCSensors[i];
   }
  }
  time = HAL_GetTick();
 }

 __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
 __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);
```

```
 //////////////////// SENTIDO 3

 HAL_Delay(500);
 // Motor esquerdo para a tras e direito para frente
 setMotors(-1,1);

 __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 500);
 __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 500);

 // OBS.: NO CODIGO DO SED, ELE SO SAI QUANDO O BOTAO FOR PRESSIONADO
 timeTick = HAL_GetTick();
 time = HAL_GetTick();
 while(time < timeTick+675) // x s
  {
   if(statusADC)
   {
    statusADC=0;
    // Para impedir que sejam utilizados valores passados
    for(i=0;i<5;i++)
    {
     if(ADCSensors[i] > max[i]) max[i] = ADCSensors[i];
     if(ADCSensors[i] < min[i]) min[i] = ADCSensors[i];
    }
   }
   time = HAL_GetTick();
  }

  setMotors(0,0);
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
  __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);

}

void CalculaPosicao(void)
{
 char c=0;
 for(c=0;c<5;c++)
  {
   B[c] = (1000 - ((unsigned long)((ADCSensors[c]-min[c])*1000)/(max[c]-min
       [c])));
  //1000 eh a diferenca entre os sensores (0, 1000 ... 4000)
  }
```

```
 pos_divisor = B[1]*1000 + B[2]*2000 + B[3]*3000 + B[4]*4000; // + B[0]*0
 posicao = (unsigned int)(pos_divisor/(B[0]+B[1]+B[2]+B[3]+B[4]));
}

void HAL_ADC_ConvCpltCallback(ADC_HandleTypeDef* hadc) // Chamada de
    conversao do ADC completa
{

 ADCSensors[0] = ADC_buffer[0]; // Mais a esquerda  - 0
 ADCSensors[1] = ADC_buffer[5]; // 1000
 ADCSensors[2] = ADC_buffer[3]; // 2000
 ADCSensors[3] = ADC_buffer[1]; // 3000
 ADCSensors[4] = ADC_buffer[2]; // Mais a direita - 4000

 // OBS. IMPORTANTE: O elemento 4 (ADC_buffer[4]) eh lixo -> Nao pega um
     valor util
 statusADC = 1;
 // Se statusADC = 1, entao novos valores foram lidos pelo ADC pelo Trigger
     do Timer.
 // Se statusADC = 0, entao o valor ja foi utilizado anteriomente; um novo
     Trigger eh necessario.
}

void pdcontrol() // Controlador PD
{
 // Curva
 if(!TypeLine)
 {
  Kp = CurveLineP;
  Ki = CurveLineI;
  Kd = CurveLineD;

  // Desacelera em rampa
  if(velocidade-5 > CurveLineSpeed) velocidade-=5;
  else velocidade = CurveLineSpeed;

 }else{ // Reta
  Kp = StraightLineP;
  Ki = StraightLineI;
  Kd = StraightLineD;

  // Acelera em rampa
  if(velocidade+5 < StraightLineSpeed) velocidade+=5;
```

```
  else velocidade = StraightLineSpeed;
 }

 int16_t proporcional = posicao - 2000; // referencia e o valor 2000
 int16_t derivativo = (proporcional - proporcional_passado); //obtendo o
     derivativo
 proporcional_passado = proporcional;

 int16_t saida_pwm = (int16_t)(((float)proporcional * Kp ) + ((float)
     derivativo * Kd );

 int16_t PWM = saida_pwm;


 if ( PWM > velocidade )  PWM =  velocidade; //Saida limitada em 0 e
     velocidade
 if ( PWM < -velocidade ) PWM = -velocidade;

 // Veiculo se deslocando para a esquerda (erro negativo)
 if (PWM < 0)
 {
   // Corrige o motor da direita, decrementando o PWM deste
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, (velocidade+PWM));
    // Motor da esquerda nao se altera
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, velocidade);
 }
 // Veiculo se deslocando para a direita (erro positivo)
 if (PWM > 0)
 {
   // Motor da direita nao se altera
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, velocidade);
    // Corrige o motor da esquerda, decrementando o PWM deste
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, velocidade-PWM);
 }
}


void SED(uint16_t countRight, uint16_t countLeft)
{

 if(n_buffer == 0)//se no existir evento no buffer ento gerar um evento
     interno(evento controlvel)
   {
```

```
 if(gerar_evento==1)
 {
  switch(g) //Aqui   implementado um gerador automtico de eventos
      controlveis

  {
  }
 }
}

else  //se existir evento nao controlavel, pegar do buffer
{
 occur_event = buffer[0];
 n_buffer--;
 k = 0;
 while(k<n_buffer)
 {
  buffer[k] = buffer[k+1];
  k++;
 }
}

//Jogador de automato
k = rfirst[current_state];
if(k==0)
{
 return;      //Dead Lock!!!
}
else
{
 while(k>0)
 {
  k--;
  if(event[k] == occur_event)
  {
   current_state = in_state[k];
   moore_output = 1;
   break;
  }
  k = rnext[k];
 }
}
```

```
if(moore_output) //Se o evento ocorrido for valido, entao imprimir saida
    fisica
{
 gerar_evento=1;
 switch(current_state)
 {

  case(0): //Adicionar Acaoo para o Estado 0 -> Estado Parado;
   controlInterrupts(1,0);
   controlInterrupts(1,1); // Todas as int. habilitadas
   controlInterrupts(1,1);
   if(!TypeSED) return; // Caso o botao B2 seja pressionado, sai do SED
      e vai para a pista ja mepeada
   break;
  case(1):// Estado 1 -> Calibramento dos sensores;
   // Desabilita todas as interrupcoes
   controlInterrupts(0,0);
   controlInterrupts(0,1);
   controlInterrupts(0,6);
   HAL_Delay(1000);
   if(!CALIBRACTR){
    CalibraSensors();
    setMotors(1,1); // Motores em posicao de corrida
    HAL_Delay(1000);
    // Habilita os timers 2 e 3 no modo Encoder
    HAL_TIM_Encoder_Start(&htim2,TIM_CHANNEL_ALL);
    HAL_TIM_Encoder_Start(&htim3,TIM_CHANNEL_ALL);
   }
   if(qMap) // Se qMap!=0, entao ja encontrou o fim da pista
   {
    setMotors(0,0); // Desativa os motores e coloca os PWMs em 0
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
    __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);
    MapBuff[0] = qMap; // Armazena no vetor a quantidade de curvas
    MapBuff[ENC_ESQ_FIM]=countLeft;
    MapBuff[ENC_DIR_FIM]=countRight;
    write2Flash(&MapBuff);
   }
   // Habilita todas as interrupcoes novamente
   controlInterrupts(1,0);
   controlInterrupts(1,1);
   controlInterrupts(1,6);
   if(!TypeSED) return;
```

```
 break;
case(2):
 blink(4,300); // Indicacao que comecara a correr
 HAL_Delay(1000);
 if(!TypeSED) return;
 break;
case(3):
 if(!TypeSED) return;
 break;
case(4):
 if(curve){
  curve=0;
  MapBuff[qMap*4+ENC_DIR_INI]=countRight; // Armazena o valor dos
      encoders
  MapBuff[qMap*4 + ENC_ESQ_INI]=countLeft; // no inicio da curva
 }
 if(!TypeSED) return;
 break;
case(5): //Achou um AchaEsq;
 if(!TypeSED) return;
 break;
case(6): //Adicionar  A   para o Estado 6 -> Achou um PerdeEsq;
 MapBuff[qMap*4+ENC_DIR_FIM]=countRight; // Armazena o valor dos
     encoders
 MapBuff[qMap*4+ ENC_ESQ_FIM]=countLeft; // no final da curva
 qMap++; //incrementa qMap, para indicar que saiu de uma curva
 if(!TypeSED) return;
 break;
case(7): // Recebeu sinal de AchaEsq;
 curve=1; // Para indicar que foi uma curva e nao um cruzamento
 if(!TypeSED) return;
 break;
case(8): //
 // Saindo do Cruzamento - nao faz nada
 if(!TypeSED) return;
 break;
case(9): //
 // Possibilidade de cruzamento
 if(!TypeSED) return;
 break;
case(10): //
 if(!TypeSED) return;
 break;
```

```
      case(11): //
       // Possibilidade de cruzamento ou fim de percurso
       if(!TypeSED) return;
       break;
     }//fim switch
     moore_output = 0;
     occur_event = -1;
    }//fim if(mealy_output)
}

void setMotors(char Left, char Right)
{

 // Se Left == 1, o motor esquerdo sera direcionado para a frente
 // Se Left == 0, o motor permanecera como desligado
 // Se Left == -1, o motor sera direcionado para tras
 if(Left<0)
 {
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,0);
    HAL_GPIO_WritePin(GPIOF,GPIO_PIN_0,1);
 }else if(!Left)
 {
  // Motor esquerdo fica em alta impedancia
    HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,0);
    HAL_GPIO_WritePin(GPIOF,GPIO_PIN_0,0);
 }else
 {
  HAL_GPIO_WritePin(GPIOB,GPIO_PIN_7,1);
  HAL_GPIO_WritePin(GPIOF,GPIO_PIN_0,0);
 }


 // Se Right == 1, o motor direito sera direcionado para a frente
 // Se Right == 0, o motor permanecera como desligado
 // Se Right == -1, o motor sera direcionado para tras
 if(Right<0)
 {
    // Motor direito para a tras
    HAL_GPIO_WritePin(GPIOF,GPIO_PIN_1,0);
    HAL_GPIO_WritePin(GPIOA,GPIO_PIN_8,1);

 }else if(!Right)
 {
```

```
  // Motor direito fica em alta impedancia
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, 0);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);

 }else
 {
  HAL_GPIO_WritePin(GPIOF, GPIO_PIN_1, 1);
  HAL_GPIO_WritePin(GPIOA, GPIO_PIN_8, 0);
 }
}

void FollowMap(uint16_t countLeft, uint16_t countRight)
{
 // Funcao utilizada para seguir a linha com a marcacao
 // Por enquanto, somente o countRight (Encoder da direita) esta sendo
     utilizado

 if(countRight >= MapBuff[qMap*4+ENC_DIR_FIM])
   {
    // Aguarda 150ms para garantir que o veiculo pare
    // entre as faixas de inicio/ fim de percurso
    HAL_Delay(150);
       __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_2, 0);
     __HAL_TIM_SET_COMPARE(&htim1, TIM_CHANNEL_4, 0);
     setMotors(0,0);
     END_OF_PATH = 1; //Flag para indicar fim de percurso
   }
 if(!qMap) // Eh a primeira iteracao -> ou seja, a curva de inicio e fim de
     percurso
 {
  // Esta entre a marcacao de inicio de pista e inicio de curva -> Uma reta
  if(countRight >= MapBuff[qMap*4+ENC_DIR_INI] && countRight < MapBuff[(
     qMap+1)*4+ENC_DIR_INI])
  {
   TypeLine = 1;
  }else if(countRight >= MapBuff[(qMap+1)*4+ENC_DIR_INI]){
   // Passa para o outro inicio de marcacao -> Esta em uma reta
   TypeLine = 0;
   qMap++;
  }
 }
 else if(qMap < MapBuff[0]) // Enquanto estiver dentro das marcacoes de
     curva mapeadas, entrara aqui
```

```
 {
  // Esta em uma curva
  if(countRight >= MapBuff[qMap*4+ENC_DIR_INI] && countRight < MapBuff[qMap
      *4+ENC_DIR_FIM])
  {
   TypeLine = 0;
  }
  // Esta em uma reta
  else if(countRight >= MapBuff[qMap*4+ENC_DIR_INI])
  {
   TypeLine = 1;
   qMap++;
  }
 }

}

// Funcao usada para enviar via Bluetooth o vetor de posicoes para o
    computador
void SendPosition()
{
 int count=0;
 uint8_t string[15] = "";
 sprintf(string,"Posicao, Tempo");
 while(count < MAP_SIZE_BUFFER)
 {
   sprintf(string,"%.5u, %.5u\n\r",mapeamento.MapVar[count].Position,
       mapeamento.MapVar[count].Time - CleanTime);
   HAL_UART_Transmit(&huart1,(uint8_t *)string,15,100);
   count++;
 }
 MANDA = 1;
}

// Funcao utilizada para obter a posicao -> FT da planta
void MapHandler()
{
   pos_divisor = (ADCSensors[0]*0)+(ADCSensors[1]*1000)+(ADCSensors
       [2]*2000)+(ADCSensors[3]*3000)+(ADCSensors[4]*4000);
   posicao = pos_divisor/(ADCSensors[0] + ADCSensors[1] + ADCSensors[2] +
       ADCSensors[3] + ADCSensors[4]);
   if(mapeamento.counterMap < MAP_SIZE_BUFFER && ADC_buffer[0] > 1000)
   {
```

```
      mapeamento.MapVar[mapeamento.counterMap].Time = (uint16_t) HAL_GetTick
          ();
      mapeamento.MapVar[mapeamento.counterMap].Position = (uint16_t) posicao;
      mapeamento.counterMap++;
    }else
    {
      // Desliga o motor da direita
      __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_2, 0);
      __HAL_TIM_SET_COMPARE(&htim1,TIM_CHANNEL_4, 0);

      // Desligar os motores e enviar, caso todos tenham alcanado a linha
          preta
      HAL_TIMEx_PWMN_Stop_DMA(&htim1,TIM_CHANNEL_2);
      HAL_TIM_PWM_Stop_DMA(&htim1,TIM_CHANNEL_4);
      HAL_ADCEx_MultiModeStop_DMA(&hadc1);
      // Funcao para enviar os dados via bluetooth
      SendPosition();
    }
}

void initMap(void)
{
 mapeamento.counterMap = 0;
}


////////////////////// END OF FILE
    ////////////////////////////////////////////
```

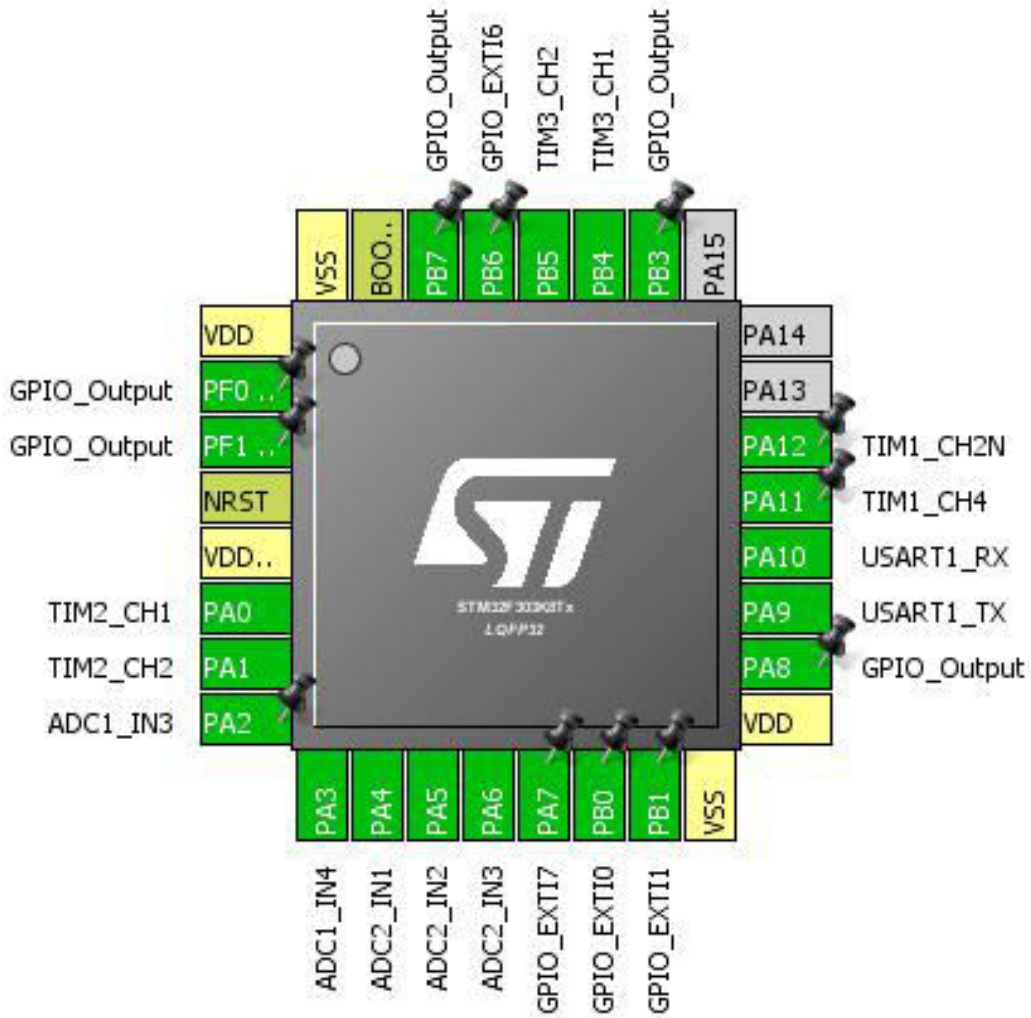
APÊNDICE B – CÓDIGO DA MEMÓRIA FLASH

```
//// Desenvolvido por Willian A Lopes
//// Todos os direitos reservados
//////////////////////////////////////

///// Includes /////
#include "stm32f3xx_hal.h"
#include "stm32f3xx_hal_flash.h"

///// Defines //////
#define MAPSIZE 30 // Quantidade maxima possivel de marcacoes na pista
#define startAdd 0x08009000 // Comecando em 36,864 KB da Flash

///// Funcoes /////

void write2Flash(uint16_t vetor[])
{
  uint32_t i;

  /* Destrava a Flash /*
  HAL_FLASH_Unlock();
  HAL_FLASH_OB_Unlock();

  // Limpa as flags que foram ativadas
  __HAL_FLASH_CLEAR_FLAG(FLASH_FLAG_EOP | FLASH_FLAG_WRPERR |
      FLASH_FLAG_PGERR);

  /*****************************************************/
  // Handler para gerenciar a programacao na Flash
  FLASH_EraseInitTypeDef Erase;
  Erase.TypeErase = FLASH_TYPEERASE_PAGES;
  Erase.PageAddress = startAdd;
  Erase.NbPages = 1;

  // Apaga a pagina antes de escrever nela
  uint32_t pageError = 0;
  if(HAL_StatusTypeDef re = HAL_FLASHEx_Erase(&Erase,&pageError) != HAL_OK)
```

```
    return; // Nao foi possivel apagar a pagina

   for(i=0;i<(4*vetor[0]+1);i++)
   {
  HAL_FLASH_Program(FLASH_TYPEPROGRAM_HALFWORD,(startAdd + 2*i),vetor[i]);
   }

  // Trava a Flash
  HAL_FLASH_Lock();
}

void ReadFromFlash(uint16_t vetor[])
{
  uint32_t i;
  i=0;
  // Primeira posicao contem a quantidade de marcacoes
  while(i < ((*(uint16_t *)(startAdd)) *4+1))
  {
    vetor[i++] = *(uint16_t *)(startAdd + 2*i);
  }
}

//////////////////////// END OF FILE ////////////////////////////////
```

APÊNDICE C – CONFIGURAÇÕES INICIAIS NO CUBEMX

2. Pinout Configuration



3. Pins Configuration

Pin Number LQFP32	Pin Name (function after reset)	Pin Type	Alternate Function(s)	Label
1	VDD	Power		
2	PF0 / OSC_IN *	I/O	GPIO_Output	
3	PF1 / OSC_OUT *	I/O	GPIO_Output	
4	NRST	Reset		
5	VDDA/VREF+	Power		
6	PA0	I/O	TIM2_CH1	
7	PA1	I/O	TIM2_CH2	
8	PA2	I/O	ADC1_IN3	
9	PA3	I/O	ADC1_IN4	
10	PA4	I/O	ADC2_IN1	
11	PA5	I/O	ADC2_IN2	
12	PA6	I/O	ADC2_IN3	
13	PA7	I/O	GPIO_EXTI7	
14	PB0	I/O	GPIO_EXTI0	
15	PB1	I/O	GPIO_EXTI1	
16	VSS	Power		
17	VDD	Power		
18	PA8 *	I/O	GPIO_Output	
19	PA9	I/O	USART1_TX	
20	PA10	I/O	USART1_RX	
21	PA11	I/O	TIM1_CH4	
22	PA12	I/O	TIM1_CH2N	
26	PB3 *	I/O	GPIO_Output	
27	PB4	I/O	TIM3_CH1	
28	PB5	I/O	TIM3_CH2	
29	PB6	I/O	GPIO_EXTI6	
30	PB7 *	I/O	GPIO_Output	
31	BOOT0	Boot		
32	VSS	Power		

* The pin is affected with an I/O function

4. Clock Tree Configuration