

CENTRO UNIVERSITÁRIO FEI

Danilo Pilotto

**CRIAÇÃO E IMPLEMENTAÇÃO DO FIRMWARE DA PLACA PRINCIPAL  
PARA FUTEBOL DE ROBÔS**

**Relatório Final**

São Paulo

2018

Danilo Pilotto

**CRIAÇÃO E IMPLEMENTAÇÃO DO FIRMWARE DA PLACA PRINCIPAL  
PARA FUTEBOL DE ROBOS**

Relatório Final de Iniciação Científica  
apresentada ao Centro Universitário FEI,  
como parte dos requisitos do Programa  
PIBIC-FEI. Orientado pelo Prof. Dr. Flavio  
Tonidandel.

São Paulo

2018

## **RESUMO**

Este projeto consiste em um relatório final do desenvolvimento do novo firmware relacionado à placa principal utilizada no robô F-180 da equipe RoboFEI (ROBOFEI Centro Universitário FEI, 2018). Analisando o futuro e os novos desafios propostos pela liga Small Size (ROBOCUP Small Size League, 2018), categoria que a equipe participa nas competições, é evidente que mudanças constantes são necessárias tanto no software quanto no hardware dos robôs. Com o novo firmware, pretendia-se reduzir o número elevado de falhas encontradas no atual. Em primeiro momento, o novo código seria modelado com base no conceito das Redes de Petri (YAKOVLEV, A., 2003), contudo, isso se mostrou muito complexo. Para contornar essa situação foi feita uma melhoria e reestruturação do código já existente. Essas mudanças proporcionaram uma melhor manutenção e desempenho do código.

Palavras-Chave:

1. RoboFEI
2. Small Size League
3. Firmware Embarcado

## LISTA DE ILUSTRAÇÕES

Figura 1 - RoboCup 2017 Nagoya Japão SSL- Final .....	9
Figura 2 - Nomenclatura das partes do Firmware. ....	11
Figura 3 - Modelo ilustrativo desenvolvido pelo autor. ....	13
Figura 4 - Modelo ilustrativo desenvolvido pelo autor. ....	13
Figura 5 - Modelo ilustrativo desenvolvido pelo autor. ....	14
Figura 6 - Modelo 3D da placa Main do projeto.....	18
Figura 7 - Modelo 3D da placa de Chute do projeto.....	19
Figura 8 - Mecanismo dos chutes do robô. ....	20
Figura 9 - Janela apresentado pelo ISE após ter sucesso em gerar o Hardware Design .....	22
Figura 10 - Janela de Debugger gerada no SDK. ....	23
Figura 11 - Estrutura final do firmware após a reorganização.....	24
Figura 12 - Estruturas de comunicação.....	25
Figura 13 – Função presente no Component Test para testar os motores das rodas uma a uma. ....	26
Figura 14 - Vista 3D do posicionamento dos conjuntos das rodas.....	27
Figura 15 - Modelo 3D do funcionamento de um encoder. ....	28
Figura 16 – Módulo utilizado pela placa principal para fazer conexão com o rádio. ....	30
Figura 17 - Estrutura interna do motor CC utilizado nas rodas do robô. ....	31
Figura 18 - Exemplificação do funcionamento de um sensor de Efeito Hall.....	32
Figura 19 - Função desenvolvida para ler o valor dos sensores Hall e controlar os motores.....	33
Figura 20 - Sistema com três vagões utilizado para a coleta de itens.....	34
Figura 21 – Modelo IOPT gerado utilizando o software Snoopy. ....	35
Figura 22 - Submodelos gerados a partir da ferramenta de divisões presente no Snoopy. ....	35
Figura 23 - Passos para o desenvolvimento de um projeto utilizando Redes de Petri dentro do software Snoopy. ....	36

## SUMÁRIO

<b>1. INTRODUÇÃO .....</b>	<b>6</b>
1.1    Objetivos .....	7
1.2    Justificativa.....	7
<b>2. REVISÃO BIBLIOGRÁFICA .....</b>	<b>8</b>
2.1    RoboCup .....	8
2.2    Small Size League .....	8
2.3    Outras Equipes .....	9
<b>3. REVISÃO TEÓRICA.....</b>	<b>11</b>
3.1    Firmware .....	11
3.2    Firmware Atual .....	11
3.3    Redes de Petri .....	12
<b>4. METODOLOGIA.....</b>	<b>16</b>
4.1    Propostas para o Projeto.....	16
<b>5. DESENVOLVIMENTO DO PROJETO PROPOSTO .....</b>	<b>18</b>
5.1    Estrutura eletrônica do projeto .....	18
5.2    Problemas encontrados durante o início da execução do projeto. ....	20
5.3    Geração do mecanismo de Debug.....	21
5.4    Reestruturação do Código .....	24
5.4.1    Communication.....	25
5.4.2    Component Test .....	25
5.4.3    Encoders .....	27
5.4.4    Main .....	28
5.4.5    PIController.....	29
5.4.6    Nordic .....	30
5.4.7    Utils.....	30
5.5    Uso do Sensor Hall .....	31
5.5.1    Sensores de Efeito Hall .....	31
5.6    Modelagem com Redes de Petri .....	33
<b>6. CONSIDERAÇÕES FINAIS .....</b>	<b>38</b>
<b>7. REFERÊNCIAS BIBLIOGRÁFICAS .....</b>	<b>40</b>

## 1. INTRODUÇÃO

Iniciando seus trabalhos em 2003, a equipe de Futebol de Robôs é o principal grupo de pesquisas e projetos robóticos do Centro Universitário FEI (CENTRO UNIVERSITÁRIO FEI, 2018). Ao longo de seus 15 anos de existência, muitas experiências e conhecimentos tecnológicos foram adquiridos, desenvolvendo robôs capazes de participar de diversas competições e categorias, como na Very Small Size (IEEE VERY SMALL SIZE SOCCER, 2018), Small Size F-180 (SMALL SIZE ROBOT LEAGUE, 2018) e Humanoid (HUMANOID ROBOT LEAGUE, 2018), proporcionando o desenvolvimento científico e tecnológico dos integrantes do projeto, além de contribuir para toda a comunidade de robótica e inteligência artificial.

Além disso, com os avanços e novidades na competição, também se formou uma equipe para competir na categoria @Home, o que levou a equipe a um novo patamar de inovações.

A equipe possui grandes resultados em competições, e é vista como referência na maior competição nacional, CBR (competição brasileira de robótica) e latino-americana (LARC/CBR OFFICIAL SITE, 2018), sendo heptacampeã na categoria F-180 e bicampeã na categoria Humanoid, além de conquistar diversos outros prêmios em competições menores e demonstrações. As equipes são formadas por alunos de graduação, de mestrado e doutorado.

A categoria que o grupo participa por mais tempo é a Small Size League e, a cada ano, a equipe RoboFEI realiza diversas melhorias e inovações nos aspectos de hardware e software, com o objetivo de competir e ganhar uma RoboCup.

Para que isso se torne uma realidade, o primeiro passo é solucionar os problemas existentes no projeto. Um desses problemas é o firmware atual da placa eletrônica principal (main). O código utilizado atualmente, além de estar ultrapassado, faz com que o robô não opere da maneira esperada.

## 1.1 Objetivos

Este projeto tem como objetivo desenvolver um novo código a ser utilizado como firmware da placa principal dos robôs utilizados no projeto RoboFEI. O novo código tem a proposta de sanar os problemas atuais e tornar o reparo e a análise do código mais rápida e direta.

## 1.2 Justificativa

A liga Small Size foi uma das categorias pioneiras a participar da RoboCup, em 1997. Desde então, ocorreram muitas mudanças, como as características do robô, tamanhos do campo, sistema de regras e estilos dos jogos, além da evolução nos aspectos de software e hardware. Esta categoria é considerada a mais dinâmica, pois os robôs são rápidos, trabalham em equipe, possuem tomadas de decisões mais ágeis e desenvolvem com mais facilidade a inteligência artificial.

Dentre os problemas do código, pode-se ressaltar dois que mais prejudicam na hora de programar o robô e criar jogadas: a movimentação e o chute. O código atual não permite o acesso à parte que controla as rodas do robô, dificultando a manutenção. Em relação ao chute, o ideal seria poder alterar sua intensidade pelo firmware, porém, como isso não acontece, há a necessidade de realizar alterações no hardware e incluir funções desnecessárias no software responsável por esse controle.

Como justificativa para esta Iniciação Científica, o aprimoramento do firmware fará com que as falhas relacionadas a ele sejam reduzidas ao máximo. Atualmente enfrentamos problemas como, o ID (identificação do robô) ser limitado no código, mal funcionamento dos sensores e entre outros. Além disso, esse aprimoramento é importante para que a equipe possa jogar cada vez melhor como um time e se adequar a novas mudanças que estão por vir.

## **2. REVISÃO BIBLIOGRÁFICA**

### **2.1 RoboCup**

A RoboCup é uma competição a nível mundial que se desenrola todos os anos. Visa o estudo e desenvolvimento da Inteligência Artificial (IA) e da Robótica, fornecendo desafios e problemas onde várias tecnologias e metodologias se podem combinar para obter os melhores resultados (ROBOCUP OFFICIAL SITE, 2018).

A primeira edição decorreu em 1997 em Nagoya no Japão. A iniciativa foi lançada pelo Dr. Hiroaki Kitano (Tóquio), um pesquisador na área da Inteligência Artificial que se tornou no presidente e fundador da RoboCup Federation, com o objectivo de dinamizar a evolução da IA (A BRIEF HISTORY OF ROBOCUP, 2018).

A RoboCup possui diversas ligas, onde cada uma possui suas características, objetivos e dificuldades próprias. As categorias são a RoboCup Soccer, RoboCup Rescue, RoboCup @Home, RoboCup Industrial e RoboCup Junior (ROBOCUP OFFICIAL SITE, 2018).

As equipas participantes das competições têm por obrigação enviar um documento que será analisado a respeito das informações técnicas do robô e os desenvolvimentos relacionados com o projeto, chamado Team Description Paper – TDP. Isso é uma forma de incentivar a contínua evolução dessas equipas (ROBOCUP OFFICIAL SITE, 2018).

### **2.2 Small Size League**

Nesta liga os robôs são pequenos que jogam futebol em equipe. Contudo, o ambiente é muito controlado. Embora sejam independentes uns dos outros, todo o processamento é efetuado num computador central, adquirindo informações (posição dos jogadores, da bola, etc.) por meio de uma câmara colocada por cima do campo de jogo. Os comandos são enviados para os robôs via wireless (ROBOCUPSOCER, 2018).



Para fazer com que os robôs atuem de tal maneira, são desenvolvidas técnicas para controlar toda a movimentação de forma precisa e rápida, jogadas e passes para fazer um jogo dinâmico. Além disso, é preciso contar com um sistema de estratégia que se adapte automaticamente para resolver teorias matemáticas para a previsão de jogadas e localização dos robôs. Também é necessário um sistema mecânico robusto para aguentar todos os impactos durante os jogos e, por fim, uma parte eletrônica que garanta uma boa e rápida resposta aos comandos enviados pelo controlador e software de estratégia (SMALL SIZE ROBOT LEAGUE, 2018).



Figura 1 - RoboCup 2017 Nagoya Japão SSL- Final

### 2.3 Outras Equipes

Ao consultar o *Team Description Paper* (TDP) e *Extended Team Description Paper* (ETDP) de outras equipes também consegue-se retirar informações úteis, como, por exemplo, em WASUNTAPICHAIKUL, Piyamate et al. (2010), vê-se que a equipe Skuba utiliza funções simples (muito parecidas com as utilizadas no código atual) para controlar os motores e definir a movimentação do robô.

Além disso, pode-se verificar que eles introduziram um sistema para detectar e corrigir correntes altas que possam causar danos sérios ao robô durante uma partida. A maneira como controlam a placa de chute não foge do padrão, onde ela é conectada diretamente a um dos pinos do FPGA que a aciona quando estipulado pelo Software.

De acordo com ISHIKAWA, Akeru et al. (2010), a equipe RoboDragons utiliza apenas três módulos em seu firmware, eles são: comunicação, comando e controle do motor. A comunicação recebe um pacote de dados e os extrai para serem utilizados pelo comando que irá calcular a velocidade linear e angular das rodas e as enviará para o controlador do motor. Quando o pacote de dados enviado possui algum tipo de erro, este é descartado.

Durante essa busca por outras equipes, foi identificado pelo time que a equipe ITAndroids utiliza a mesma estrutura eletrônica que a equipe RoboFEI. De acordo com Benny Mirahy et al. (2019), para simplificar o uso da FPGA, apenas parte do micro controlador presente é utilizada. Além disso, em contato direto com a equipe, descobriu-se que foi implementado em seu firmware uma maneira simples de descobrir um problema de hardware, para isso foi projetado um código para realizar testes de certos componentes utilizando a ferramenta de debug disponível no software utilizado na programação do controlador da placa eletrônica.

### 3. REVISÃO TEÓRICA

#### 3.1 Firmware

Também conhecido pela nomenclatura “software embarcado”, o Firmware é um conjunto de instruções operacionais que são programadas diretamente no hardware de equipamentos eletrônicos (OLIVEIRA, 2006).

Os códigos transcritos por este tipo de programa são fundamentais para iniciar e executar os hardwares e os seus recursos, fornecendo informações idênticas sempre que o dispositivo for ligado (OLIVEIRA, 2006).

Desta forma, este sistema responsável por operar aparelhos eletrônicos estará gravado diretamente no chip de memória de seus hardwares, mais especificamente nas memórias PROM e EPROM (memória somente leitura programável e memória somente leitura programável apagável, respectivamente) (OLIVEIRA, 2006).

#### 3.2 Firmware Atual

O algoritmo atual utilizado pela equipe é dividido em 10 partes. No geral, essas partes deveriam fazer com que o firmware operasse da melhor maneira esperada.

1 - Futbots.c	2 - Futbots.h	3 - IIRFilter.c	4 - IIRFilter.h	5 - perifericos.c
6 - perifericos.h	7 - timers.c	8 - timers.h	9 - xgpio_tapp_example.c	10 - xparameters.h

Figura 2 - Nomenclatura das partes do Firmware.

Contudo, algumas dessas partes não estão sendo mais utilizadas. As abas 3 e 4 já não fazem mais parte do código atual. Essas estruturas constituíam filtros para prevenir ruídos, porém, durante a passagem por esse filtro, as variáveis eram transformadas de *int* para *float* diversas vezes, isso fazia com que parte da informação fosse se perdendo durante a sua execução.

As estruturas 2, 6 e 8 (.h) são responsáveis por definir as principais funções que serão utilizadas posteriormente. Além disso, temos as estruturas 9 e 10 que são programas prontos para fazer com que o rádio, utilizado no envio de informações entre robô e computador de estratégia, opere corretamente.

A estrutura Futbots.c é a principal do programa, ela é responsável por controlar as principais funções do robô, como por exemplo, a tensão que será enviada para um determinado motor da roda que o fará girar em uma determinada rotação e, assim, fará com que o robô se mova da maneira que foi solicitada.

A estrutura de controle perifericos.c é um bloco a parte que executa determinadas funções e envia os valores ao bloco principal. Essa estrutura ocorre constantemente e o fato dela ser isolada faz com que o tempo de execução do bloco principal diminua.

A estrutura timers.c controla os atrasos, para fazer isso ela conta o número de vibrações (em Hertz) do cristal que há na placa e assim determina o tempo, já que a placa não possui nenhum tipo de relógio. Isso é importante para que tudo ocorra no seu devido momento. Além disso, sem esses atrasos, poderiam surgir ruídos, o que faria com que a placa apresentasse um mau funcionamento ao executar duas ações simultâneas.

### 3.3 Redes de Petri

Uma Rede de Petri ou rede de transição é uma das várias representações matemáticas para sistemas distribuídos discretos. Como uma linguagem de modelagem, ela define graficamente a estrutura de um sistema distribuído como um grafo direcionado com comentários (CARDOSO, 1997). Possui nós de posição, nós de transição e arcos direcionados conectando posições com transições (figura 3).

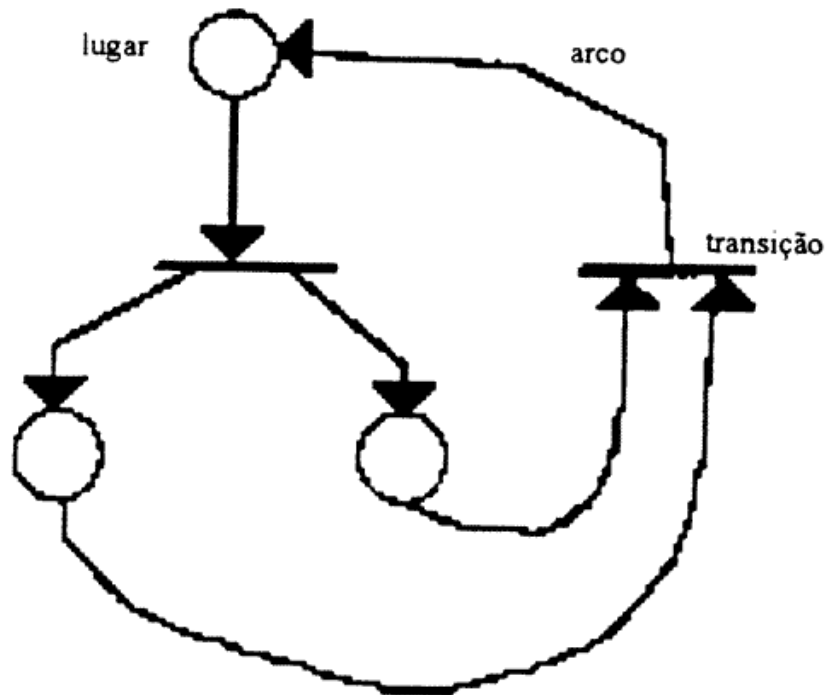


Figura 3 - Modelo ilustrativo desenvolvido pelo autor.

Uma transição possui lugares de entrada (um ou vários) e lugares de saída (um ou vários) como mostrado na figura 4 (CARDOSO, 1997).

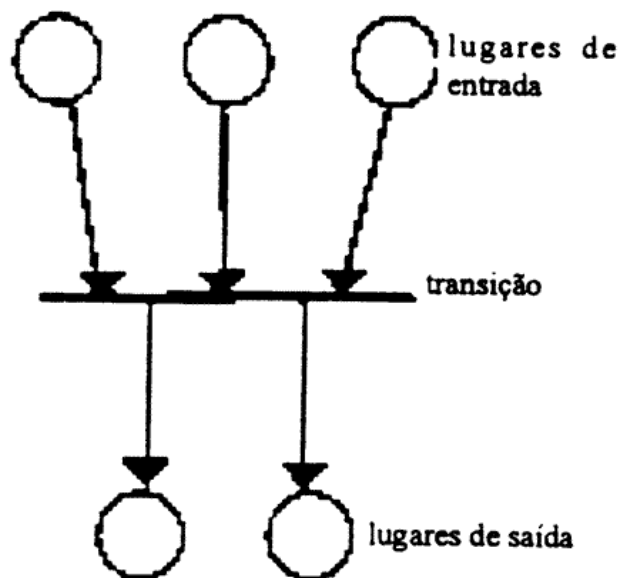


Figura 4 - Modelo ilustrativo desenvolvido pelo autor.

Para representar o dinamismo dos fenômenos, é necessário que se coloquem marcas (fichas ou, do inglês, *tokens*) em determinados lugares da

rede. A circulação destas marcas através da rede é o que vai dar a noção deste dinamismo.

Uma transição é dita como habilitada quando todos os seus lugares de entrada estão marcados. Ocorrendo isto, a transição pode ser realizada, isto é, retira-se uma marca de cada lugar de entrada e coloca-se uma marca em cada lugar de saída.

Na figura 5 temos um exemplo do uso das marcas. No *momento 1* existe uma marca no lugar L1 e não existem marcas nos lugares L2 e L3, neste caso apenas a transição T1 está habilitada. Ao disparar, a transição T1 faz com que a marca do seu lugar de entrada seja transferida para os seus lugares de saída que resulta no *momento 2* (CARDOSO, 1997).

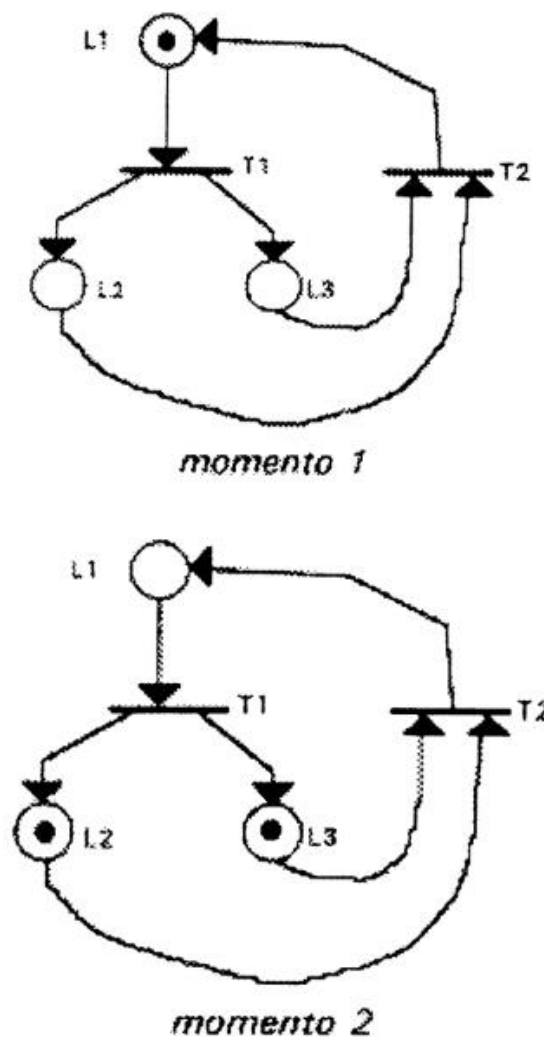


Figura 5 - Modelo ilustrativo desenvolvido pelo autor.

Agora a transição T1 não está mais habilitada, mas a transição T2 está, pois, todos os seus lugares de entrada, no caso L2 e L3, contêm marcas, o que faz com que a rede volte ao estado representado pelo *momento 1* quando T2 for disparada (CARDOSO, 1997).

Os conceitos apresentados sobre as Redes de Petri trarão uma melhoria e rapidez durante as manutenções no código. Isso porque, como já mencionado, o seu conceito se baseia em condições para que cada estrutura ou função do código seja efetuada. A sua implementação estará ligada a criação de blocos e/ou funções que utilizarão dessas condições para determinar se parte do código deve ser executada.

## 4. METODOLOGIA

Em primeiro momento será estudado o funcionamento do firmware utilizado atualmente nas placas. Posteriormente, será desenvolvido o código que ocupará o lugar do antigo e, com isso, será analisada a melhora obtida.

O código atual utilizado pela equipe é de difícil compreensão e manutenção. E também, ao realizar testes com os robôs, se faz evidente que ele não é eficaz como deveria ser.

Outro ponto positivo é que o método de desenvolvimento utilizado nas Redes de Petri sobre lugares, transições e fichas faz com que o código tenha uma fácil compreensão, o que ajudará em reparos futuros.

### 4.1 Propostas para o Projeto

Para conseguir o melhor controle possível, será feito um estudo sobre o firmware utilizado por equipes de alto nível para ver qual a chance de implementar algo parecido. E ainda, será feita uma pesquisa sobre programações em placas parecidas com a utilizada para, com isso, diminuir ao máximo o número de falhas.

Além disso, será feita a adaptação desses códigos, que muitas vezes estão em inglês ou em forma de algoritmo, para a linguagem C++, que é a utilizada na programação feita pela equipe e para a estruturação das Redes de Petri. Para fazer com que o código fique de acordo com esses conceitos estudados, diversos artigos sobre a implementação das Redes de Petri em linguagem C/C++ serão estudados.

Após essa implementação serão feitos diversos testes de controle para saber se a placa está operando como deveria, se está enviando os sinais corretos, passando as tensões devidas, entre outros. Isso será feito diversas vezes ao longo do projeto de iniciação. Esse elevado número de testes irá prevenir ao máximo as falhas e mal funcionamentos.

Características importantes para serem avaliadas nos testes são:



- Tempo de processamento: algo muito demorado pode atrapalhar (ou até mesmo alterar) o envio de informações ao rádio.
- Uso da memória: quanto menor o espaço utilizado melhor. Isso, além de reduzir o tempo de processamento, previne que o código exceda o limite da memória utilizada.
- Verificar se as estruturas de controle estão atuando da maneira correta: como cada bloco de controle é responsável por uma parte do robô, é extremamente importante verificar constantemente se cada uma delas está operando da maneira como deveria e se está enviando os valores corretos ao rádio.

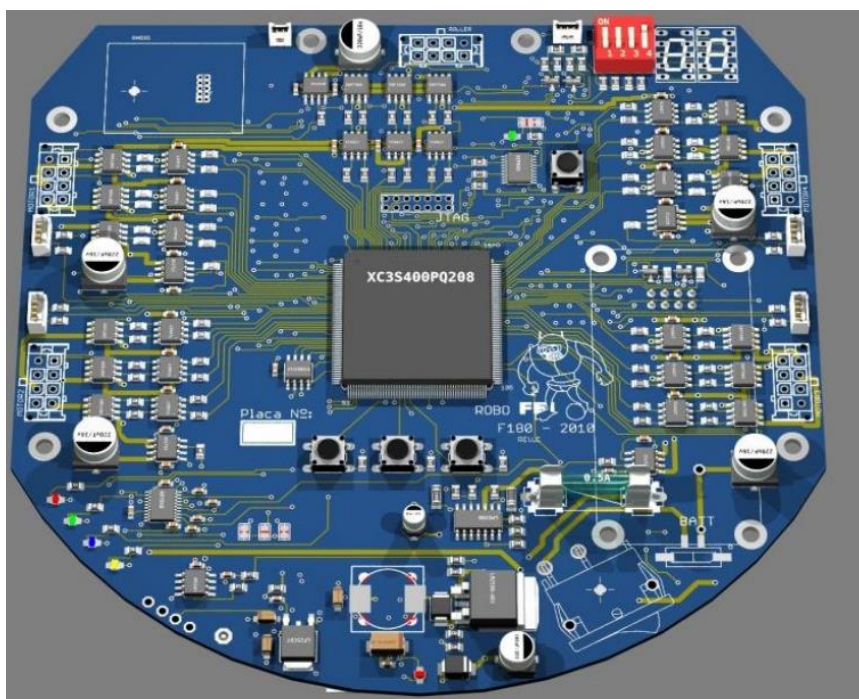
## 5. DESENVOLVIMENTO DO PROJETO PROPOSTO

Durante o início do desenvolvimento do projeto, muito foi alterado com relação ao que seria feito. A princípio, a ideia seria dedicar os dois primeiros meses à pesquisa e a análise do código que compõe o firmware atual. Contudo, o tempo de pesquisa se estendeu devido à complexidade do projeto proposto.

Gerar um código a partir de uma placa já confeccionada pode parecer algo simples. Porém, essa tarefa se torna simples apenas se a placa em questão tiver um circuito eletrônico básico.

### 5.1 Estrutura eletrônica do projeto

Analisando o circuito atual da placa principal (Main) do projeto RoboFEI ficou claro que instituir um código ao seu micro controlador do zero seria algo inviável.

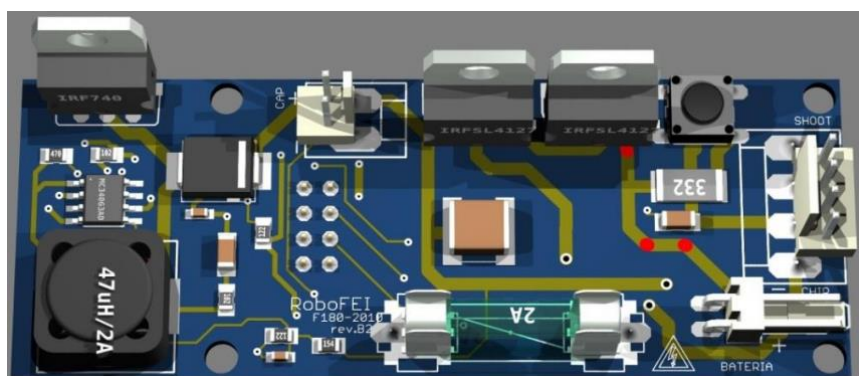


*Figura 6 - Modelo 3D da placa Main do projeto*

A placa é composta por diversos componentes como o FPGA (Field Programmable Gate Array), conversores A/D, circuitos dos motores, memórias flash e PROM e muitos outros. Além disso, a placa é composta por cinco *layers* (camadas), cada uma responsável por formar os caminhos das trilhas internas que conectam os componentes que as utilizam.

São necessárias 5 dessas camadas pois a placa utiliza de cinco tensões diferentes. A bateria fornece 12V que são regulados em cascata para 5V, 3.3V, 2.5V e 1.2V, sendo as duas últimas exclusivas para o FPGA. Os 12 volts não são utilizados de fato por essa placa, a quinta camada interna seria a do GND (terra) utilizada por todos os componentes.

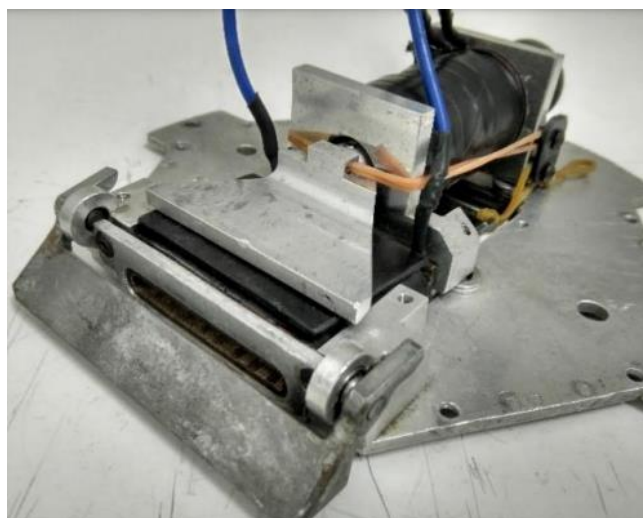
Em adição, é utilizado uma segunda placa, a placa denominada “de Chute”. Como o nome propõe, esta é a placa responsável por controlar os dois chutes existentes no robô.



*Figura 7 - Modelo 3D da placa de Chute do projeto.*

Por se tratar de um circuito de potência e por medidas de segurança, o sistema eletrônico de chute está contido numa placa separada opto-acoplada à placa onde está o FPGA (placa principal).

O objetivo da placa de chute é controlar o carregamento de dois capacitores, ligados em paralelo, até um valor de tensão muito alto (em torno de 160 volts). Caso o sistema de estratégia decida que o robô execute um chute, toda a energia armazenada nos capacitores será descarregada em um solenoide, que com o campo magnético gerado pela passagem da corrente em seu enrolamento, movimentará um êmbolo para realizar o chute na bola.



*Figura 8 - Mecanismo dos chutes do robô.*

## 5.2 Problemas encontrados durante o início da execução do projeto

Como mencionado anteriormente, gerar um código para compor o firmware do zero seria algo inviável. Para gerar esse código, o caminho básico a ser seguido é detalhar e descrever toda a trajetória a ser percorrida e isso deve ser feito para cada componente partindo de uma das pernas do FPGA. Esse trabalho se prolongaria ao tempo proposto para este projeto.

Diante da situação descrita, a solução encontrada foi analisar o firmware utilizado por outras equipes no mesmo nível do RoboFEI para encontrar uma solução possível para o prazo determinado. Durante essa busca, como mencionado na sessão dois deste artigo, foi encontrada a equipe ITAndroids que faz o uso das mesmas placas eletrônicas apresentadas acima.

Com isso, descobriu-se que o firmware utilizado pela equipe ITAndroids foi gerado com base no código disponível no repositório online<sup>1</sup> do RoboFEI. Além disso, esta equipe não possui problemas atrelados a este firmware. Com isso, ao invés de criar um novo código, surgiu a possibilidade de alterar e melhorar um código antigo e funcional do projeto.

<sup>1</sup> – Disponível em: <<https://bitbucket.org/robofei/electronics/overview>>.

### 5.3 Geração do mecanismo de Debug

Após um longo tempo de pesquisa e com o novo objetivo em mente, a nova primeira etapa constituiu-se da análise e entendimento do programa não didático *ISE Design Suite 14.7*, software da Xilinx, que permite a geração da base de arquivos utilizados na programação dos FPGAs desta empresa.

Seguindo como base os tutoriais encontrados no site da Xilinx sobre programação no FPGA Spartan 3, micro controlador utilizado na placa Main apresentada acima, foi possível dar início a geração dos arquivos base para a implementação futura do código desejado. Esta base de arquivos é denominada *workspace* e é ela que irá constituir as estruturas de controle necessárias para dar função ao controlador.

Dentro do programa ISE 14.7, o primeiro passo a ser seguido é gerar um *new source* do tipo *embedded system* com as configurações disponíveis no repositório da equipe RoboFEI. Além disso, devem-se incluir os *pcores*, encontrados no mesmo local, nas pastas geradas pela criação citada acima. Esses *pcores* são o grupo de arquivos que serão utilizados como as bibliotecas base no programa para gerar a estrutura de *hardware* a ser utilizada pelo Spartan 3 no projeto.

Ainda no ISE, o próximo passo a ser seguido é gerar o *Top HDL Souce*, bases de arquivos HDL que irão configurar o *design de hardware* da placa em questão. Feito isso, o próximo passo é editar as estratégias para a criação desse design. Editá-las fará com que com o programa consiga superar alguns obstáculos que são encontrados durante a sua execução. Sem isso, o programa não é gerado. Nesta etapa, o *hardware design* já está gerado e pronto para ser utilizado nas próximas etapas (figura 9).

Em seguida, deve-se exportar os arquivos gerados para o Xilinx Software Development Kit (SDK), programa este que irá gerar o código a ser executado pelo controlador que até o momento possui apenas o *hardware design* modulado. O ISE abrirá o SDK e irá lhe pedir para selecionar uma pasta onde ele salvará os arquivos que irão compor o *workspace* do programa.

system_top Project Status				
Project File:	fpga.xise	Parser Errors:	No Errors	
Module Name:	system_top	Implementation State:	Programming File Generated	
Target Device:	xc3s400-4pq208	• Errors:	No Errors	
Product Version:	ISE 14.7	• Warnings:	<a href="#">4855 Warnings (0 new)</a>	
Design Goal:	Balanced	• Routing Results:	<a href="#">All Signals Completely Routed</a>	
Design Strategy:	<a href="#">xlxo</a>	• Timing Constraints:	<a href="#">All Constraints Met</a>	
Environment:	<a href="#">System Settings</a>	• Final Timing Score:	0 ( <a href="#">Timing Report</a> )	

XPS Reports				
Report Name	Generated	Errors	Warnings	Infos
<a href="#">Platgen Log File</a>	dom 13. jan 21:55:44 2019	0	<a href="#">17 Warnings (13 new)</a>	<a href="#">69 Infos (15 new)</a>
Simgen Log File				
<a href="#">BitInit Log File</a>	qua 10. abr 14:39:49 2019	0	<a href="#">2 Warnings (0 new)</a>	<a href="#">54 Infos (0 new)</a>
System Log File				

Device Utilization Summary				
Logic Utilization	Used	Available	Utilization	Note(s)
Number of Slice Flip Flops	4,787	7,168	66%	
Number of 4 input LUTs	5,273	7,168	73%	
Number of occupied Slices	3,545	3,584	99%	

Figura 9 - Janela apresentado pelo ISE após ter sucesso em gerar o Hardware Design

No SDK, o próximo passo é gerar um *Application Project*, que finalizará a construção do *workspace* desejado. Esta aplicação irá utilizar o *hardware design* feito anteriormente para implementar um arquivo simples que atuará com o firmware do processador.

O próximo passo é acessar a pasta gerada pela aplicação e substituir o arquivo implementado pelo código que atuará, de fato, como firmware. Em primeiro momento foram usados os códigos disponíveis no repositório do time para assegurar o domínio da ferramenta. Com isso feito, basta comandar o programa para gerar os arquivos que irão compor o *debug*.

O último passo é gerar o arquivo .elf que será o código fonte (firmware) quando lido pelo FPGA. Para isso, é necessário conectar a placa com as memórias vazias ao SDK e baixar o arquivo gerado para a placa. Quando isso é feito, a placa já começa a se comportar como esperado. Além disso, a janela “console” surge no SDK e com isso temos acesso à parte de debug do código (figura 10).

O debug nada mais é do que um "encontrador" de erros que irá impedir o código de funcionar normalmente. Com ele é possível saber exatamente o que está acontecendo dentro do código fonte, muitas vezes o próprio programa sugere ações para a solução de um erro encontrado. Outra função do debug é ajudar o programador a entender melhor o que acontece com o seu código em tempo de execução (em tempo real).

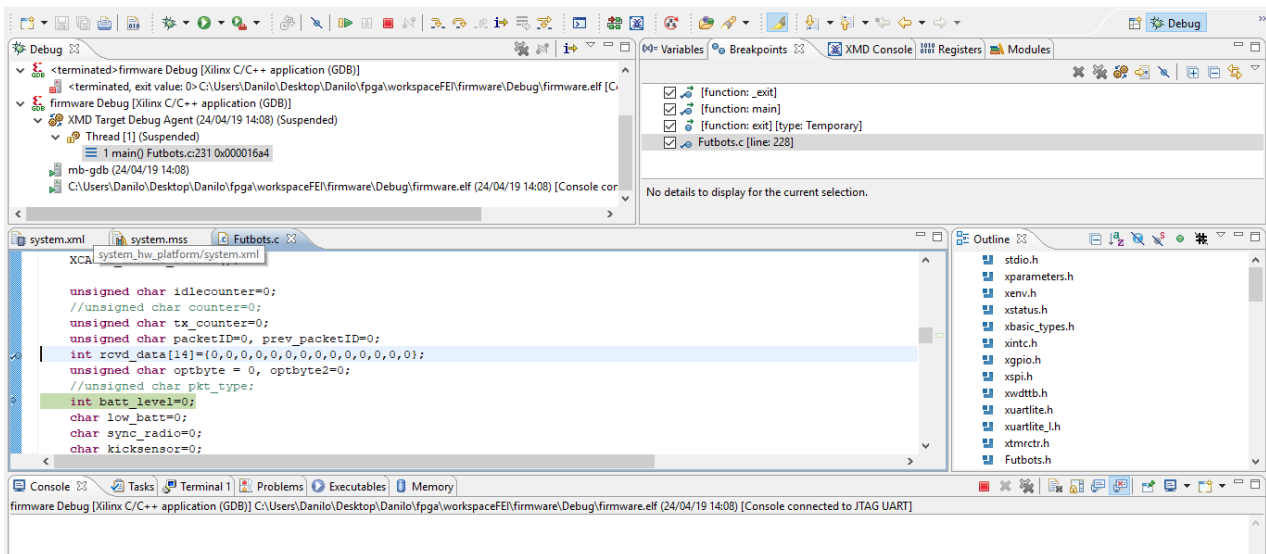


Figura 10 - Janela de Debugger gerada no SDK.

Porém, ainda não é possível utilizar a placa no dia a dia. Os FPGAs possuem o mesmo funcionamento de memórias do tipo RAM, portanto, quando a placa for desligada o código desaparecerá. Para que o procedimento de baixar o código do programa para a placa não tenha que ser feito todas as vezes, a placa conta com memórias do tipo EPROM (*Erasable Programmable Read Only Memory*). Esse tipo de memória se faz útil, pois ela mantém os seus dados quando desligada da energia de alimentação, e ainda, esses dados podem ser apagados e substituídos, se necessário.

Para isso, deve-se gerar um arquivo .bit que irá conter tanto o hardware design desenvolvido quando os dados existentes no arquivo .elf gerado anteriormente. Esta etapa será realizada no ISE 14.7.

Um dos últimos passos é gerar o arquivo .mcs que será gravado na memória da placa e transferido ao FPGA quando a placa for alimentada. Isso é feito utilizando o software iMPACT. Para isso, deve-se especificar o tipo de



memória que será utilizada e incluir o arquivo .bit, que foi gerado anteriormente, no software. Com o arquivo gerado, basta conectar a placa e baixa-lo para a sua memória.

Com toda a parte para analisar e gravar o código já gerada e com a devida documentação feita, o próximo passo se deu por reestruturar o código que estava sendo utilizado pela equipe. Ao longo do desenvolvimento seguinte, alguns ajustes precisaram ser feitos para que a sua conclusão se desse no tempo estipulado. Além disso, algumas precauções foram tomadas para que tanto a documentação quanto o código desenvolvido fossem de fácil entendimento para futuras mudanças necessárias.

#### 5.4 Reestruturação do Código

No código antigo as funções e declarações estavam separadas do código principal, mesmo assim, estavam todos aglomerados em um único local, `perifericos.c`.

Para isso, foi necessário identificar cada função presente dentro do firmware atual e separá-las de acordo com a função que a estrutura em questão controla. Isso será de grande ajuda para as próximas etapas do processo de desenvolvimento uma vez que será mais intuitivo acessar os arquivos do código.

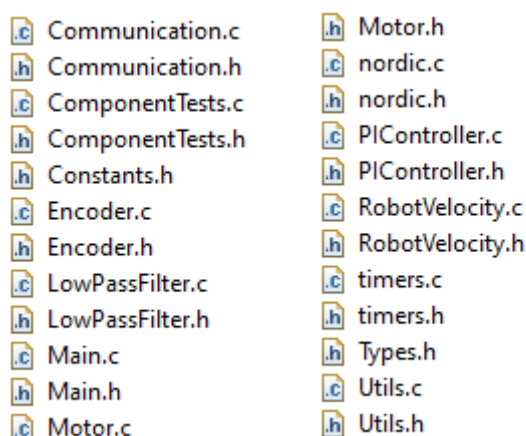


Figura 11 - Estrutura final do firmware após a reorganização.



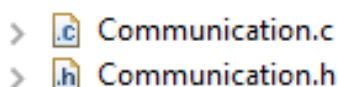
Além da reorganização, outras estruturas foram adicionadas ao código. Essas estruturas têm como objetivo facilitar a manutenção do robô e do próprio programa desenvolvido.

Para um melhor entendimento geral do projeto proposto, as principais estruturas modificadas serão explicadas. E ainda, por motivos de simplificação e melhoria no tempo de processamento deste código, foi utilizado apenas o necessário do micro controlador presente no módulo de controle da placa eletrônica do projeto.

#### 5.4.1 Communication

Como o nome indica esta parte do código é responsável por fazer a comunicação entre computador principal e robô, ou seja, quando um pacote de informações gerado pelo software de estratégia for enviado via rádio ao jogador contendo os seus próximos passos, esta função será acionada.

O papel desempenhado por essa estrutura é de identificar quais são os próximos movimentos a serem efetuados pelo robô, decodifica-los e passar esses parâmetros às funções que farão esses movimentos acontecerem. Além disso, como pode ser visto na figura 12, existem duas estruturas presentes no código que assumem este mesmo nome. Porém, uma delas termina em .c e a outra em .h, isso acontece devido à necessidade de declarar uma função antes de se poder escrevê-la, o que acontece no arquivo neste arquivo. Nele também existe uma breve explicação sobre o funcionamento de cada função presente.



```
> .c Communication.c
> .h Communication.h
```

*Figura 12 - Estruturas de comunicação.*

#### 5.4.2 Component Test

Diferente da estrutura citada acima, que foi uma realocação de estruturas já existentes, este bloco de controle foi adicionado ao firmware. Um problema muito presente durante a realização de manutenções é saber quais componentes apresentam seu funcionamento fora do esperado. Por isso, criou-

se uma maneira de executar funções básicas do robô enquanto o mesmo está conectado ao computador.

Como mencionado na subseção 2.3 deste artigo, uma das equipes analisadas faz uso de estruturas que podem ser utilizadas em conjunto com a ferramenta de *debug* para analisar o funcionamento de componentes do hardware. Com isto em mente, foi desenvolvido um bloco que será responsável por testar esses componentes. Para isso, existem funções que, por exemplo, fazem o acionamento das rodas uma a uma e retornam o valor das rotações ao usuário. Isso mostra se os dispositivos físicos que controlam as rodas estão funcionando como deveriam.

Além disso, esse bloco se mostra importante uma vez que é possível verificar se uma nova função, ou modificação feita ao código, irá operar como o esperado. Utilizando do debugger gerado pode-se simular a execução de uma das funções existentes nesse trecho do código e ,com isso, concluir se ela irá executar como esperado.

```
void testWheelMotors() {
    xil_printf("Testing wheel motors...\n");
    Motor motors[4];
    Motor_initialize(&motors[0], XPAR_BLDCMOTOR_1_BASEADDR);
    Motor_initialize(&motors[1], XPAR_BLDCMOTOR_2_BASEADDR);
    Motor_initialize(&motors[2], XPAR_BLDCMOTOR_3_BASEADDR);
    Motor_initialize(&motors[3], XPAR_BLDCMOTOR_4_BASEADDR);
    int count = 0;
    int maxCount = 1000;
    xil_printf("Sending PWM 40 for each motor in positive direction...\n");
    while (count < maxCount) {
        int i = 2;
        for (i = 0; i < 4; ++i)
            Motor_setPWM(&motors[i], 40);
        delayMilliseconds(10);
        ++count;
    }
    delayMilliseconds(1000);
    count = 0;
    xil_printf("Sending PWM 40 for each motor in negative direction...\n");
    while (count < maxCount) {
        int i;
        for (i = 0; i < 4; ++i)
            Motor_setPWM(&motors[i], -40);
        delayMilliseconds(10);
        ++count;
    }
}
```

Figura 13 – Função presente no Component Test para testar os motores das rodas uma a uma.

Dentro do *Component Test* também é possível encontrar a função que é responsável por fazer o robô funcionar completamente, chamada de loop principal. Isso se deve ao fato de que, caso seja necessário testar as rodas, é possível desativar o loop principal que iria interferir na aquisição dos dados.

#### 5.4.3 Encoders

Para fazer o controle dos motores de corrente contínua utilizado nas rodas existem encoders acoplados a esses motores, destacados na figura 14. Portanto, deve-se haver um controle relacionado a este dispositivo para que seja possível controlar a velocidade de rotação dele.

E esta é a estrutura de controle responsável por isso. Para isso, deve-se primeiro especificar a quantidade de ranhuras existentes em seu disco (resolução), com esta informação, o software saberá o tamanho do sinal que deverá ser enviado para que a roda gire em uma determinada velocidade.

Então, quando for iniciado o comando para movimentar o robô, primeiro deve-se inicializar o leitor do encoder e ir realizando medições a cada ciclo que o sistema rodar. A diferença entre a nova medição com a anterior dará o número de ranhuras que foram lidas neste intervalo de tempo, e, com isso, o tamanho do sinal. Fazendo uma comparação com o calculado teoricamente pelo software, o robô saberá quando deve parar de aumentar a velocidade das suas rodas.

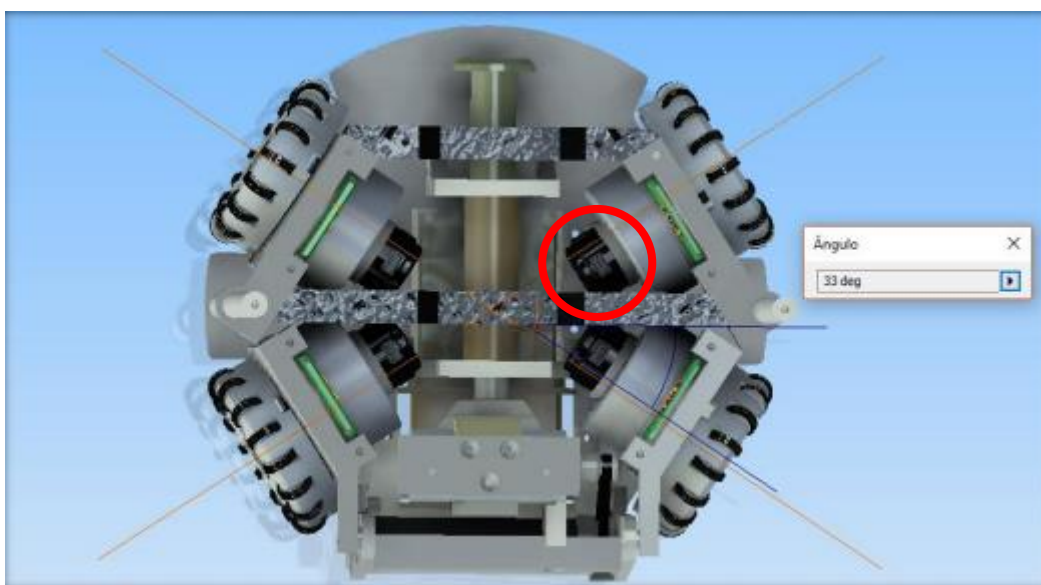
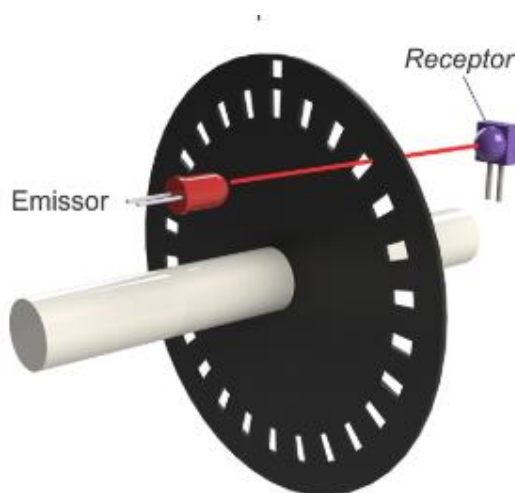


Figura 14 - Vista 3D do posicionamento dos conjuntos das rodas.

#### 5.4.3.1 Encoder

Este dispositivo é um equipamento que funciona a base de eletricidade, ele converte os movimentos circulares ou as mudanças lineares em pulsos elétricos. O encoder funciona como um sensor que consegue passar os dados sobre a velocidade ou a posição em que uma parte do equipamento está. Geralmente esta peça faz movimento circulares ou lineares.

A leitura do encoder é fundamentada em um disco que tem janelas radiais transparentes e opacas, como pode ser visto na figura 15. Esta janela é iluminada de forma perpendicular por uma espécie de raio infravermelho, as imagens formadas são projetadas no receptor. Este, por sua vez, converte essas imagens em impulsos elétricos que serão dados de entrada para um código, por exemplo. (HI TECNOLIGIA, 2019).



*Figura 15 - Modelo 3D do funcionamento de um encoder.*

#### 5.4.4 Main

Dentro desta parte do código pode-se encontrar o programa principal que fará o robô funcionar em uma situação real de jogo. Este também é o espaço onde estão as chamadas para as funções presentes no bloco *Component Test*.

Para que um dos jogadores comece a operar, a primeira coisa a ser feita é iniciar a comunicação dele com o computador responsável por desenvolver

as estratégias de jogo, ou seja, conectar-se ao rádio. Depois de conectado, inicia-se o loop principal, nele, todos os motores, sensores e chutes são iniciados, com isso, pode-se dizer que o firmware já se encontra na função presente dentro do *Component Test* que atua como este loop.

Esta função de controle principal deve ser um loop, pois o computador principal está analisando a situação de jogo, gerando novas estratégias constantemente e as enviando ao robô num intervalo de 16 milissegundos, aproximadamente. Quando um desses pacotes é recebido, o firmware deve decifrá-los, executá-los e ficar no aguardo do recebimento de um novo pacote.

#### 5.4.5 PIController

Como mencionando anteriormente, para haver um controle preciso das velocidades nas rodas dos robôs, o programa deve comparar o valor lido nos sensores com um valor pré-calculado teoricamente. Este cálculo é feito no bloco de controle denominado PIController.

Para a execução deste controle o primeiro passo é atribuir o valor máximo para as rotações dos motores, seja de roda ou do rolete. Em seguida, deve-se compará-lo com o valor lido pelos sensores analógicos e, por fim, verificar se a velocidade ainda deve ser aumentada ou mantida.

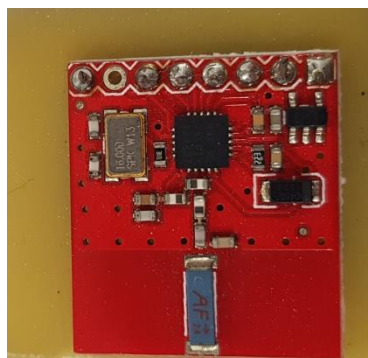
Um dos grandes problemas identificados pela equipe durante testes e em situações reais de jogo foi o controle das rodas, que não existia. Portanto, se um parafuso de uma das rodas estivesse mais apertado esse motor não compensaria isso com mais corrente, deixando esta roda mais devagar que as outras, ou seja, o código responsável por esta parte não estava formulado da maneira correta, fazendo com que o robô andasse de forma desgovernada.

O método de comparações utilizado pelo PIController contorna essa situação, pois, uma vez que a velocidade real não chegue na calculada o sistema irá continuar aumentando a sua velocidade. Então, mesmo que uma das rodas esteja mais presa, todas terão a mesma velocidade angular.

#### 5.4.6 Nordic

Para que o computador da estratégia possa passar os pacotes de dados para a placa principal do robô, é utilizado um rádio. Portanto, deverá existir um módulo que faz a conexão desta placa com ele, como pode ser visto na figura 16. Com isso, deve haver também um controle sobre essa conexão.

Dentro da estrutura do firmware denominada como Nordic pode-se encontrar funções que irão inicializar a conexão com o rádio e viabilizar a transferência de informações entre placa e estratégia, seja para enviar ou receber uma informação.



*Figura 16 – Módulo utilizado pela placa principal para fazer conexão com o rádio.*

Para que haja um controle do robô por meio da estratégia, há também uma função responsável por escrever um vetor que será retornado ao computador contendo informações sobre velocidade angular das rodas, chutes ativados, rolete ativo, entre outros.

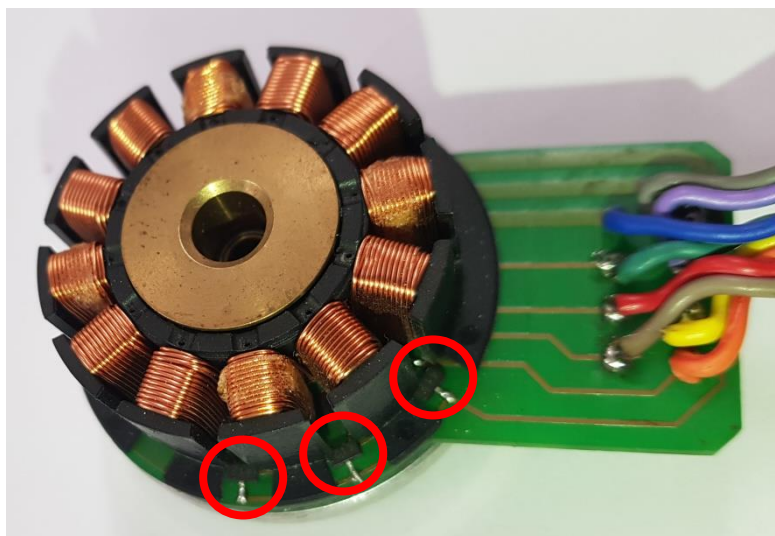
#### 5.4.7 Utils

Ao separar cada função e realoca-las em seus respectivos locais algumas delas ficaram deslocadas por serem muito específicas e únicas. Por isso, foi criada a aba utilitária, onde essas funções foram colocadas. Isso foi feito apenas por motivos de organização, uma vez que elas poderiam ser colocadas em qualquer local. Dentre as funções presentes, pode-se encontrar desde funções responsáveis por acender LEDs até as responsáveis por ativar os chutes, ou até mesmo por identificar a bola.

## 5.5 Uso do Sensor Hall

Durante o desenvolvimento do projeto proposto um grande problema surgiu. Algumas semanas antes da competição latino americana, da qual a equipe RoboFEI iria participar, foi identificado que não haveriam encoders suficiente para a montagem completa de 6 robôs para os jogos a serem disputados.

A solução encontrada para contornar a situação foi utilizar os sensores de efeito Hall presente no motor das rodas. Porem, para que isso fosse feito, deveria haver uma grande modificação no código apresentado acima.



*Figura 17 - Estrutura interna do motor CC utilizado nas rodas do robô.*

### 5.5.1 Sensores de Efeito Hall

Os sensores de campos magnéticos são partes fundamentais de uma infinidade de aplicativos, que vão desde aparelhos de consumo até máquinas industriais. A forma mais simples de se fazer o sensoramento magnético é através de uma bobina, mas existem elementos semicondutores próprios para isso, que são os sensores de Efeito Hall.

Existem substâncias no grupo dos semicondutores, que possuem propriedades elétricas importantes, que podem ser aproveitadas na construção de diversos tipos de dispositivos eletrônicos. Para o caso dos sensores Hall, são utilizados os sensíveis à variação do campo magnético.



Essa sensibilidade faz com que o material permita uma maior passagem de corrente quando não há nenhum campo magnético presente e uma menor passagem quando há campo. Por outro lado, quando não há campo magnético também não existe tensão sobre os terminais deste sensor e a presença de um campo faz com que uma determinada tensão de saturação apareça (SMAR Automação Industrial, 2019).

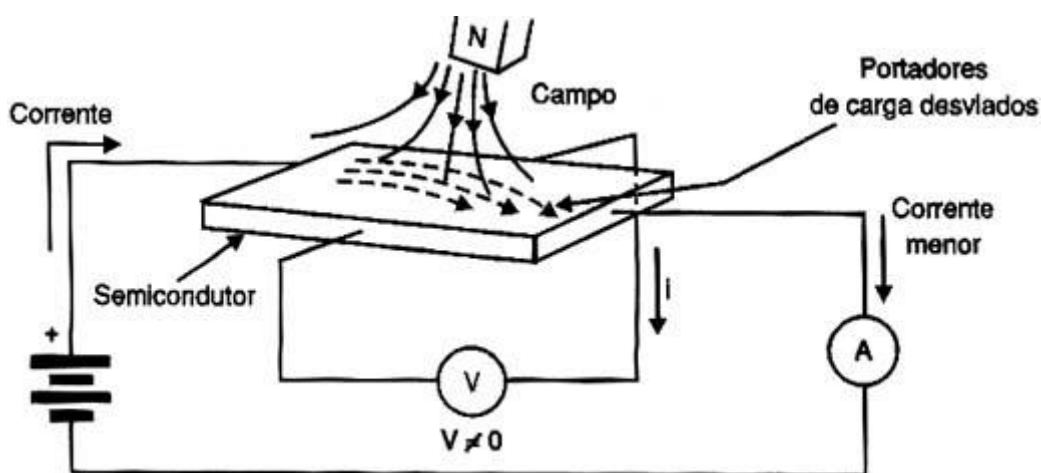


Figura 18 - Exemplificação do funcionamento de um sensor de Efeito Hall.

Sabendo do conceito apresentado acima, pôde-se concluir que seria viável fazer a transição do encoder para o sensor Hall. Além disso, essa mudança traria uma diminuição dos gastos da equipe, pois, não haveria mais a necessidade de gastar com a compra de encoders. Para fazer o controle da velocidade do robô utilizando os sensores apresentados, seria necessário alterar as funções responsáveis por isso. Primeiro, no módulo PIController, seria necessário alterar os cálculos teóricos, pois, como pode ser visto na figura 17, existem três desses sensores em cada motor ao invés de apenas um, como acontecia no caso do encoder. Contudo, o método para efetuar o controle não seria tão complicado, o princípio baseia-se apresentado na sessão encoders. Primeiro, deveria ser inicializado a leitura deste sensor analógico, depois deveria ser feita a contagem dos pulsos por intervalo de clock do sistema e em seguida feita a diferença com a leitura anterior, e, por fim, comparar com o resultado calculado na função ajustada dentro do PIController.



O desafio está em ajustar as constantes que colocam os valores lidos pelo sensor analógico no mesmo sistema de medidas que as calculadas pelo método teórico internamente. Grande tempo e esforço foram dedicados a isso e, em paralelo, a resolver o problema da falta de encoders.

Faltando poucos dias para a competição, a equipe havia conseguido encoders suficientes e o código que utilizaria os Halls se encontrava em um empasse com relação às constantes citadas.

```
float getMotorSpeed(int n_wheel){
    int count=0;
    switch(n_wheel){
        case 1:
            count = XGpio_ReadReg (XPAR_BLDCMOTOR_1_BASEADDR, 1);
            break;
        case 2:
            count = XGpio_ReadReg (XPAR_BLDCMOTOR_2_BASEADDR, 1);
            break;
        case 3:
            count = XGpio_ReadReg (XPAR_BLDCMOTOR_3_BASEADDR, 1);
            break;
        case 4:
            {
                count = XGpio_ReadReg (XPAR_BLDCMOTOR_4_BASEADDR, 1);
            }
            break; }
    // se a contagem tem um valor muito alto, isso porque esta no sentido inverso
    // remover o fundo de escala (0xFFFF ou 65535) obter o valor no sentido oposto (negativo)
    if (count > 0x8000)
        count -= 0xFFFF;
    //Calcula velocidade da roda
    float speed = 0; // rad/s
    // Frequencia de amostragem = 200Hz
    // Pulsos do Hall por volta = 48
    float aux=-(count - prev_odom[n_wheel-1]);
    speed = aux * 2.0f * PI * 200.f / 48.0f;
    //Salva pulsos atuais
    prev_odom[n_wheel-1] = count;
    return speed;
}
```

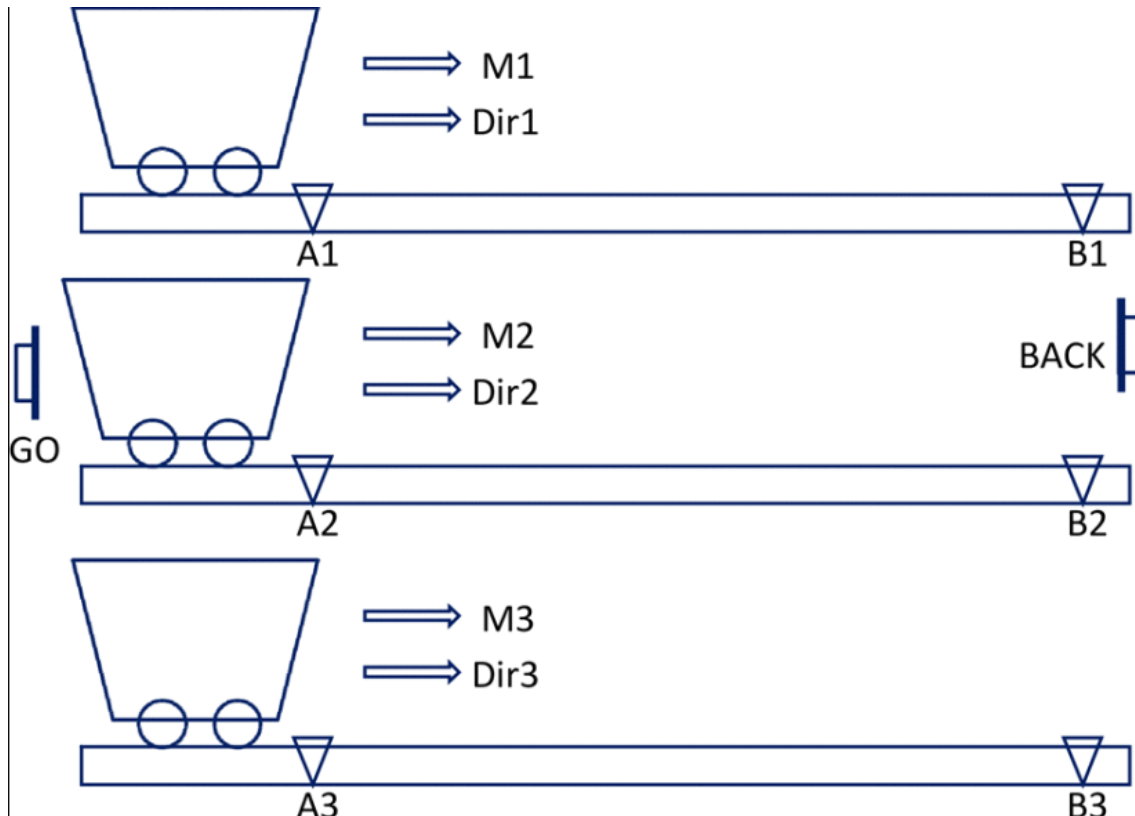
Figura 19 - Função desenvolvida para ler o valor dos sensores Hall e controlar os motores.

## 5.6 Modelagem com Redes de Petri

Utilizando os conceitos apresentados na revisão teórica, o passo final para a conclusão do projeto proposto seria remodelar o código desenvolvido utilizando as Redes de Petri. Para isso, o primeiro passo se deu pela aprendizagem no desenvolvimento de uma rede simples em um software de modelagem.

Em primeiro momento, foi proposto o desenvolvimento de um exemplo simples para a implementação do código em linguagem C. O projeto em

questão irá desenvolver um controlador para a automação de um sistema composto por três carros que irão se mover entre dois pontos para coletar itens. Eles irão se mover juntos quando um botão GO for pressionado e deverão retornar quando todos se encontrarem em suas posições finais e o botão BACK ativado, como pode ser observado na figura abaixo.



*Figura 20 - Sistema com três vagões utilizado para a coleta de itens.*

De acordo com R. Nunes et al (2007), para utilizar as Redes de Petri para a resolução do problema proposto, um modelo de Input-Output Place-Transition (IOPT) foi produzido utilizando o software Snoopy específico para modelagens desse tipo de rede, figura 21. Este modelo pode ser utilizado para gerar o código que irá controlar o sistema.

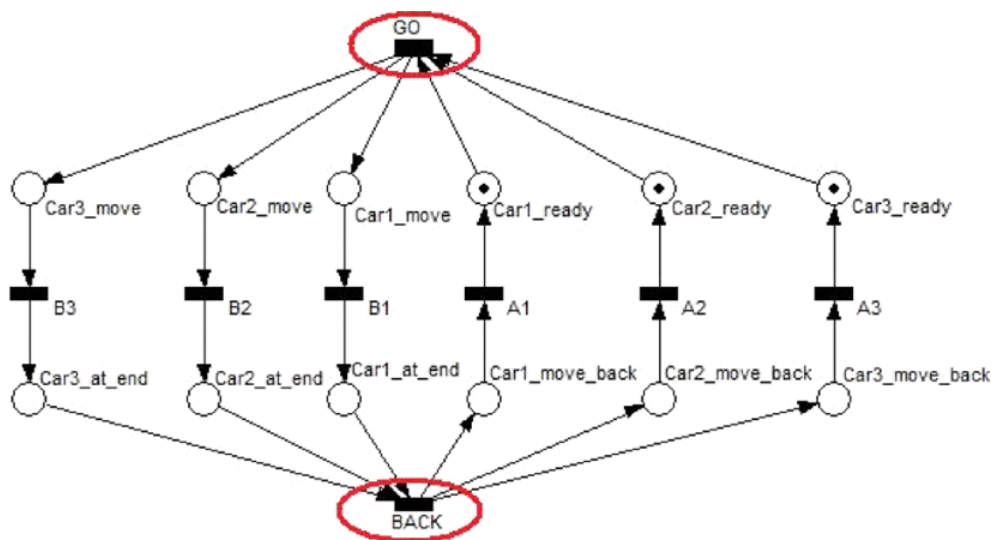


Figura 21 – Modelo IOPT gerado utilizando o software Snoopy.

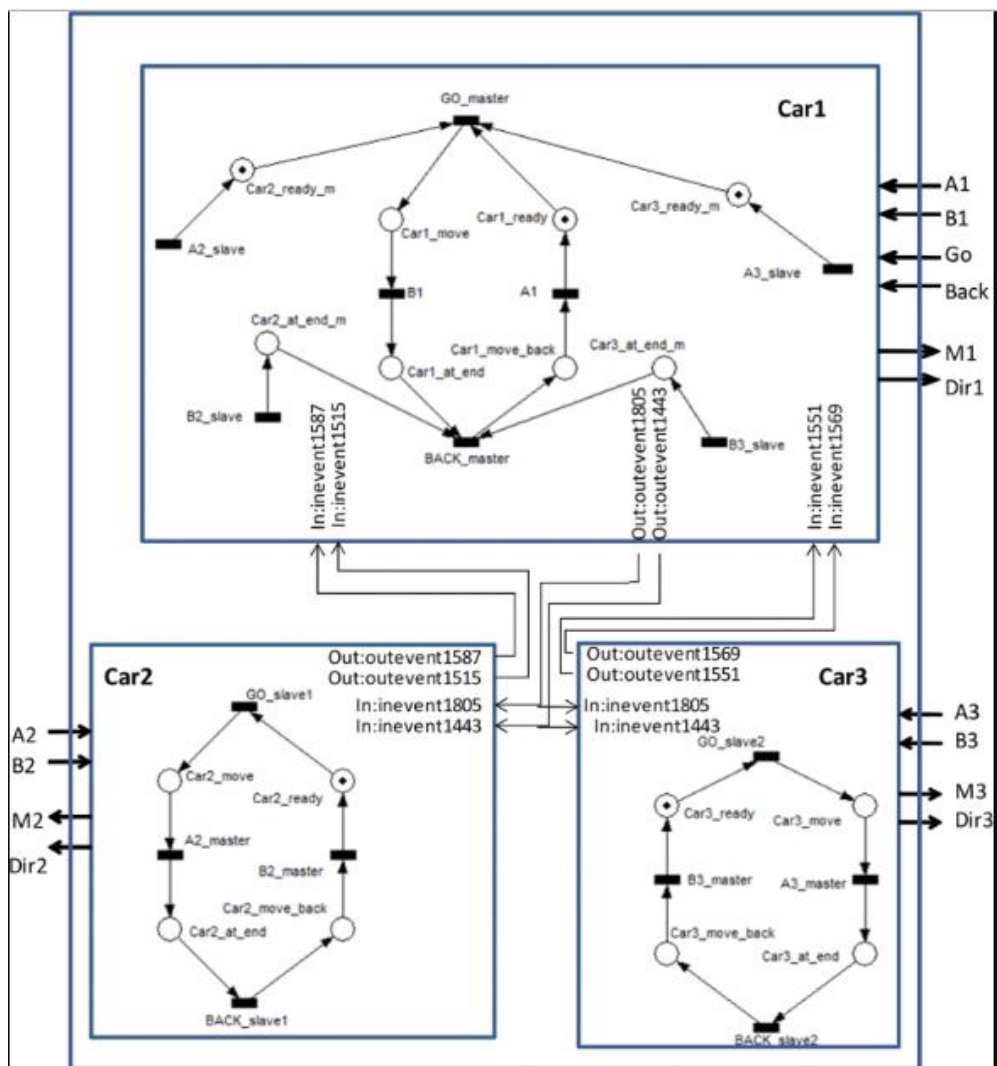


Figura 22 - Submodelos gerados a partir da ferramenta de divisões presente no Snoopy.

Contudo, como o objetivo é gerar um controlador individual para cada vagão, será necessário separar o modelo IOPT em três submodelos, um para cada carro. Segundo A. Costa et al (2009), para isso, pode-se utilizar a ferramenta de divisão e associação de redes (*Split tool*) presente no software utilizado. Na figura 22 é possível notar que os nós de transição GO e BACK foram utilizados como pontos de corte quando aplicada a *Split tool*.

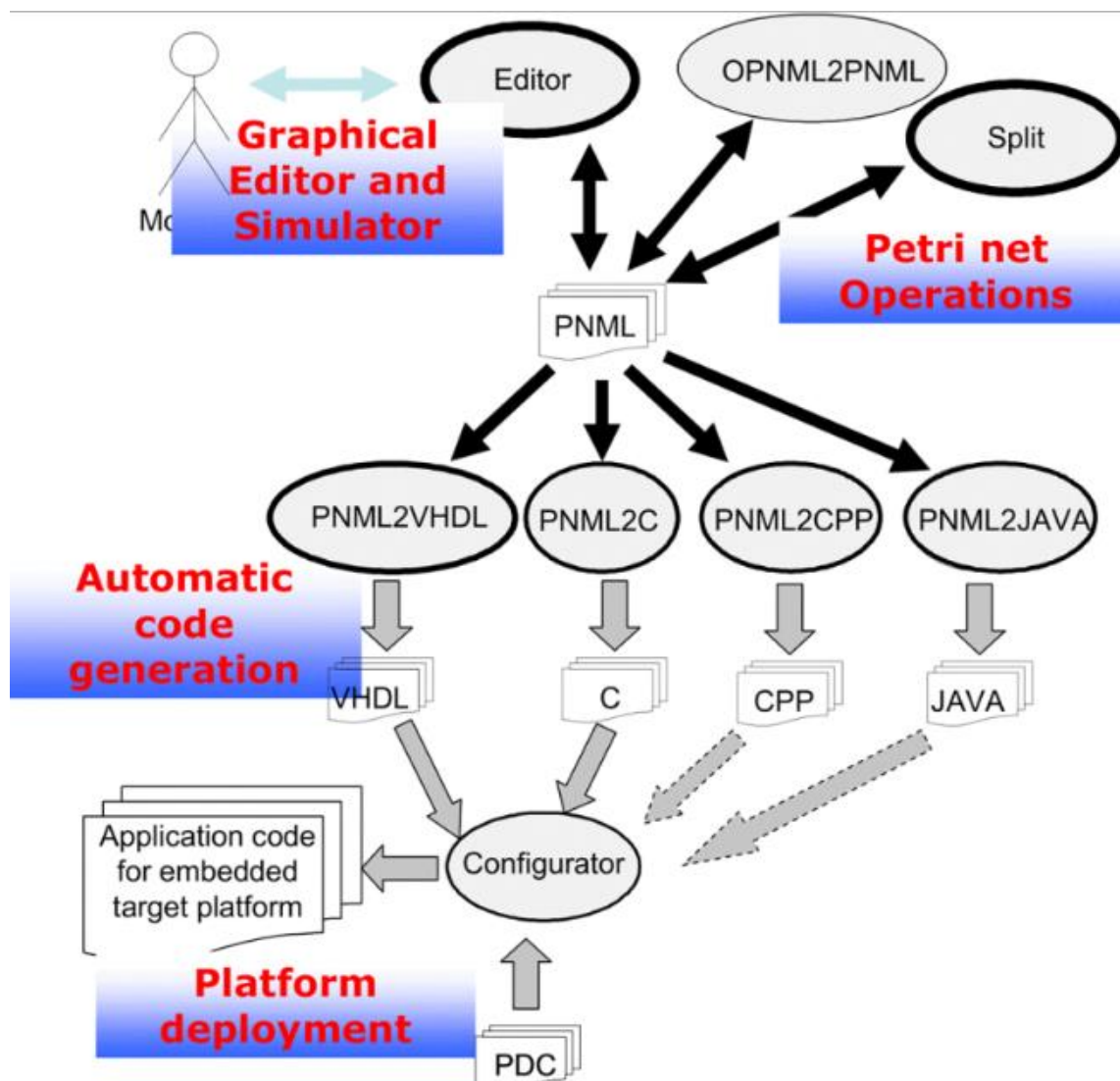


Figura 23 - Passos para o desenvolvimento de um projeto utilizando Redes de Petri dentro do software Snoopy.

Após a separação das redes, o próximo passo se dá por exportar o código gerado na linguagem desejada. O software utilizado possibilita essa exportação nas linguagens VHDL, C, CPP e Java, como mostra a figura 23.

Porém, ao exportar o código que atuaria como controlador para os vagões verificou-se que a sua implantação para a proposta de iniciação não traria muita vantagem. Para um projeto simples, como o detalhado acima, o código gerado na linguagem de programação C possuía mais de 1500 linhas e apresentou todo o seu desenvolvimento atrelado a estruturas de decisão do tipo if e while.

Projetos modelados com este tipo de ferramenta de estruturação se mostram mais uteis e diretos para uma descrição de designs de hardware em outro tipo de linguagem, como VHDL. Por motivos de comparação, o código em VHDL também foi gerado para este caso e ele apresentou por volta de 400 linhas, uma diminuição de 73% com relação ao anterior. Por isso, para um controle que precisa ser feito em uma linguagem de alto nível, neste caso a linguagem C, não existem vantagens evidentes na utilização das Redes de Petri.

## 6. CONSIDERAÇÕES FINAIS

Um dos grandes problemas existentes no firmware atual é a falta de documentação. Graças a isso, não se tem acesso à boa parte dele, como a parte de debug e indexação de variáveis. Esta falta de informações faz com que uma parte significativa deste código não seja entendida pelos programadores, fazendo com que essas funções sejam excluídas.

Com a reformulação apresentada, além de o entendimento do código ter se tornado mais intuitivo, o acesso no caso de alguma modificação ou correção transformou-se em algo direto e rápido, pois, basta ler os nomes das componentes do firmware e identificar onde a ação necessária deve ser efetuada.

Além disso, outro benefício adicionado com a execução do projeto foi o fato de existir toda a documentação necessária tanto para a criação de uma área de trabalho (workspace) quando para a utilização da ferramenta de debug. Mesmo que, por algum acaso, todo o código novo desapareça, será possível à geração de um novo workspace para a adição dos arquivos do firmware.

O fato da implementação do controle das rodas utilizando o sensor de Efeito Hall não ter sido concretizada faz com que a compra de encoders ainda seja necessária. Contudo, isto não é um grande problema no momento, já que todos os motores adquiridos pela equipe nos últimos anos o utilizam. Além disso, a opção de usar o Hall ainda não foi descartada, uma vez que para utiliza-los como mecanismo de controle basta ajustar as constantes necessárias para isso. Porém, isso demandará certo tempo e esforço.

Com relação às Redes de Petri, como mencionado anteriormente, a sua implantação neste projeto traria mais problemas do que soluções. Para que o modelamento completo do projeto fosse feito, todas as possíveis condições para o funcionamento do firmware deveriam ser levadas em consideração. Além de inviável, isso excederia o tempo estipulado para a realização do projeto proposto.

Além do mais, ao realizar a exportação dessas redes utilizando o software específico para a sua modelagem, o código em C/C++ gerado utiliza

muito de estruturas de decisão do tipo if e/ou while. Com isso, para que ocorra uma passagem completa do código, todas as essas estruturas presentes devem ser verificadas. Portanto, tratar as informações da maneira que estava sendo feita pelo código já existente, utilizando funções, se mostrou mais eficiente considerando que cada pacote deve ser tratado e enviado aos seus devidos locais antes do recebimento dos próximos dados, aproximadamente 16 milissegundos.

Após algumas semanas de funcionamento do novo firmware, a placa utilizada para os testes foi deixada de lado devido à competição estar se aproximando. Após a volta da competição, foi verificado que esta placa não estava mais fazendo a comunicação com o rádio. Este problema ainda está pendente.

Como um trabalho futuro, pode-se considerar a modelagem do loop principal utilizando as Redes de Petri. Além de pequeno, lá já existem as chamadas das funções necessárias para executar cada comando. Por isso, mesmo que o código gerado se baseie em estruturas de decisão, isso não afetará o desempenho do código como um todo.

Além disso, também é possível fazer a geração, utilizando o conceito das Redes, do *hardware design* que foi deixado de lado devido à complexidade da placa utilizada pelo robô em questão. Com o software Snoopy, pode-se criar toda a lógica de processamento do hardware desejado e, com isso, gerar a estrutura VHDL necessária para que seja possível produzir um workspace, utilizando o *software* ISE 14.7.

## 7. REFERÊNCIAS BIBLIOGRÁFICAS

A. Costa, L. Gomes. **Petri net partitioning using net splitting operation**. INDIN'2009 - 7th IEEE International Conference on Industrial Informatics, 24-26 Junho 2009, Cardiff

BITBUCKET Repository. **RoboFEI Electronics**. 2014. Disponível em: <<https://bitbucket.org/robofei/electronics/overview>>. Acesso em 10 abr. 2019.

CARDOSO, Janette; VALETTE, Robert. **Redes de Petri**. Florianópolis, 1997.

HI Tecnologia. **O que é Encoder? Para que serve? Como escolher? Como interfacear?** Disponível em: <<https://www.hitecnologia.com.br/blog/o-que-%C3%A9-encoder-para-que-serve-como-escolher-como-interfacear/>>. Acesso em 12 nov. 2019.

ISHIKAWA, Akeru at el. **RoboDragons 2010 Extended Team Description**. Disponível em: <[http://wiki.robocup.org/images/b/b4/Small\\_Size\\_League\\_-\\_RoboCup\\_2010\\_-\\_ETDP\\_RoboDragons.pdf](http://wiki.robocup.org/images/b/b4/Small_Size_League_-_RoboCup_2010_-_ETDP_RoboDragons.pdf)>. Acesso em 02 out. 2018.

LARC/CBR. **Competição Brasileira de robótica**. Disponível em: <<http://www.cbrobotica.org/?lang=pt>>. Acesso em 01 set. 2018.

LARC/CBR. **IEEE Very Small Size Soccer (VSSS)**. Disponível em: <[http://www.cbrobotica.org/?page\\_id=81](http://www.cbrobotica.org/?page_id=81)>. Acesso em 01 set. 2018.

MIRAHY, Benny at el. **ITAndroids Small Size Soccer Team Description 2019**. Disponível em: <[https://ssl.robocup.org/wp-content/uploads/2019/03/2019\\_TDP\\_ITAndroids\\_Small\\_Size.pdf](https://ssl.robocup.org/wp-content/uploads/2019/03/2019_TDP_ITAndroids_Small_Size.pdf)>. Acesso em 27 nov. 2019.

OLIVEIRA, André Schneider de; ANDRADE, Fernando Souza de. **Sistemas Embarcados: Hardware e Firmware na Prática**. São Paulo: Érica Ltda., 2006.



R. Nunes, L. Gomes, J.P. Barros. **A Graphical Editor for the Input-Output Place-Transition Petri Net Class**. ETFA'2007-12th IEEE Conference on Emerging Technologies and Factory Automation, Setembro 25-28, 2007.

ROBOCUP @Home. **About @Home**. Disponível em: <<http://www.robocupathome.org/>>. Acesso em 02 set. 2018.

ROBOCUP oficial site. **RoboCupSoccer - Small Size**. Disponível em: <<https://www.robocup.org/leagues/7>>. Acesso em 01 out. 2018.

ROBOCUP. The Robocup Federation. **A Brief History of RoboCup**. Disponível em: <[https://www.robocup.org/a\\_brief\\_history\\_of\\_robocup](https://www.robocup.org/a_brief_history_of_robocup)>. Acesso em 01 out. 2018.

ROBOCUP. The Robocup Federation. **Humanoid Robot League**. Disponível em: <<https://www.robocuphumanoid.org/>>. Acesso em 01 set. 2018.

ROBOCUP. The Robocup Federation. **Página oficial da competição RoboCup**. Disponível em: <<https://www.robocup.org/>>. Acesso em 02 set. 2018.

ROBOCUP. The Robocup Federation. **RoboCup Rules**. Disponível em: <<http://www.robocup.org/aboutrobocup/regulations-rules/>>. Acesso em 02 set. 2018.

ROBOFEI Centro Universitário da FEI. **Página do time de futebol de robôs da FEI**. Disponível em: <<https://portal.fei.edu.br/Pagina/robo-fei>>. Acesso em 01 set. 2018.

SMAR Automação Industrial. **Sensor Hall - A tecnologia dos Posicionadores Inteligentes de última geração**. Disponível em: <<http://www.smar.com/brasil/artigo-tecnico/sensor-hall-a-tecnologia-dos-posicionadores-inteligentes-de-ultima-geracao>>. Acesso em 14 out. 2019.

WASUNTAPICHAIKUL, Piyamate at el. **Skuba 2010 extended team description.** Disponível em:  
<[http://wiki.robocup.org/images/6/6b/Small\\_Size\\_League\\_-\\_RoboCup\\_2010\\_-\\_ETDP\\_Skuba.pdf](http://wiki.robocup.org/images/6/6b/Small_Size_League_-_RoboCup_2010_-_ETDP_Skuba.pdf)>. Acesso em 02 out. 2018.

YAKOVLEV, Alex; GOMES, Luis; LAVAGNO, Luciano. **Hardware Design and Petri Nets.** The Netherlands, Klumer Academic Publishers, 2003.