



GOVERNO DO ESTADO DO RIO DE JANEIRO  
SECRETARIA DE ESTADO DE CIÊNCIA E TECNOLOGIA  
FUNDAÇÃO DE APOIO À ESCOLA TÉCNICA  
CENTRO DE EDUCAÇÃO PROFISSIONAL EM TECNOLOGIA DA INFORMAÇÃO  
FACULDADE DE EDUCAÇÃO TECNOLÓGICA DO ESTADO DO RIO DE JANEIRO  
FAETERJ/PETRÓPOLIS

*Construção de um time de futebol de robôs para a  
categoria IEEE Very Small Size Soccer*

**Johnathan Fercher da Rosa**

Petrópolis - RJ

Março, 2015

**Johnathan Fercher da Rosa**

***Construção de um time de futebol de robôs para a categoria IEEE Very Small Size Soccer***

Trabalho de Conclusão de Curso apresentado à Coordenadoria do Curso de Tecnólogo em Tecnologia da Informação e da Comunicação da Faculdade de Educação Tecnológica do Estado do Rio de Janeiro Faeterj/Petrópolis, como requisito parcial para obtenção do título de Tecnólogo em Tecnologia da Informação e da Comunicação.

Orientador:

Eduardo Krempser da Silva

Co-orientador:

Alberto Torres Angonese

Petrópolis - RJ

Março, 2015

# *Folha de Aprovação*

Trabalho de Conclusão de Curso sob o título “*Construção de um time de futebol de robôs para a categoria IEEE Very Small Size Soccer*”, defendida por Johnathan Fercher da Rosa e aprovada em 05 de Março de 2015, em Petrópolis - RJ, pela banca examinadora constituída pelos professores:

---

D.Sc. Eduardo Krempser da Silva  
Orientador

---

M.Sc Alberto Torres Angonese  
Coorientador

---

M.Sc Fábio Silveira Vidal  
Instituto Federal de Educação, Ciência e  
Tecnologia do Tocantins

# *Declaração de Autor*

Declaro, para fins de pesquisa acadêmica, didática e tecnico-científica, que o presente Trabalho de Conclusão de Curso pode ser parcial ou totalmente utilizado desde que se faça referência à fonte e aos autores.

---

Johnathan Fercher da Rosa  
Petrópolis, em 05 de Março de 2015

# *Dedicatória*

Ao Laboratório de Sistemas Inteligentes e Robótica, que me proporcionou um grande acúmulo de conhecimento e que sem a existência do mesmo esse trabalho não poderia ser desenvolvido.

# *Agradecimentos*

Agradeço primeiramente a meus pais José e Simone pelo apoio dado durante toda minha vida e por me ensinarem desde cedo a importância dos estudos. Agradeço também a meu irmão Johnny pelo apoio, conversas e brincadeiras. Agradeço a meus amigos Asafe, Luan, Matheus, Thais, Jennifer e Laura, cujo a companhia apesar de não ser continua são muito importantes para mim. Agradeço a meus amigos do Laboratório de Robótica Oscar, Lucas Borsatto, Lucas Marques, Hebert, Marlon e Alexandre por dividir comigo seus conhecimentos e agradeço a meus professores e amigos Eduardo e Alberto por toda a orientação e por servirem de inspiração.

# *Epígrafe*

"Falar é fácil, mostre-me o código."

---

(Linus Torvalds)

# *Resumo*

Este trabalho aborda o problema da construção de um time de robôs autônomos para a competição de futebol de robôs *IEEE Very Small Size* com um olhar voltado para um curso de computação, onde buscou-se simplificar a criação do *hardware* e focar-se na programação do todo. Tal problema é dividido em vários pequenos problemas como, por exemplo, localização dos robôs em campo com utilização de visão computacional construída com o auxílio da biblioteca OpenCV, navegação dos robôs em campo com utilização da técnica campos potenciais, controle aplicado a robótica com utilização da técnica de controle PID, transmissão dos dados via *wireless*, construção dos robôs com descrição de todos os artefatos físicos presentes nos mesmos, recepção dos dados pelos robôs e execução dos mesmos e definição da arquitetura utilizada no *software*. Esse trabalho demonstra parte da teoria utilizada na maioria dos pedaços do algoritmo criado, além de demonstrar também as dificuldades, os erros, e as soluções aplicadas ou ainda não a plataforma criada. Por fim, esse trabalho retrata como a equipe *SirSoccer* do Laboratório de Sistemas Inteligentes e Robótica (SIR Lab) abordou o problema que além de ser multidisciplinar acabou-se tornando bastante verticalizado em algumas partes do mesmo, vale ressaltar também que tal trabalho conquistou o 4º lugar da *Latin American Robotics Competition* em sua segunda participação em um evento do tipo.



# *Abstract*

This work describe the problem that is building a team of autonomous robots to a football competition of robots IEEE Very Small Size with a focused look at the computer course, which was searched simplify the hardware building with focus on the programming that were used. This problem is divide in many little problems, as for an example, localization of the robots in a field with the utilization of Computer Vision, built with the assistance of the OpenCV library, navegation of the robots in a camp using the idea of Potential Field, applied control to the robotics using the technical control PID, transmission o f the data by wireless, building of the robots with the description of all physical artifacts present in it, reception of the data and execution by the robots and the definition of architecture used in the software. This work demonstrates a part from the theory used in the most pieces of the algorithm created, moreover, it demonstrates the difficulties, the mistakes, and the solutions applied or didn't was a platform built. Finally, this work portrays how the equip SirSoccer from the Laboratory of Intelligent System and Robotics (SIR Lab) aborted the problem that is multidisciplinary, eventually becoming a large vertical in some of your divisions, and so conquesting the forth place of the Latin American Robotics Competition of 2014 in your second participation in an event like this.

# *Lista de Figuras*

1.1	Logomarca da competição. . . . .	p. 16
1.2	Logomarca do Laboratório de Sistemas Inteligentes e Robótica. . . . .	p. 17
1.3	Estrutura de funcionamento da categoria VSS. . . . .	p. 18
1.4	Campo da categoria com suas marcações de chute-livre(FB), Faltas(FK) e Pênaltis(PK). . . . .	p. 19
1.5	Área máxima da bola que pode ser coberta. . . . .	p. 20
1.6	Partida entre SirSoccer (FAETERJ) e Autobotz (UFMG). . . . .	p. 21
2.1	Fluxograma do sistema da equipe SirSoccer. . . . .	p. 23
2.2	Imagem do sistema em funcionamento. . . . .	p. 24
3.1	Logomarca da biblioteca OpenCV. . . . .	p. 26
3.2	Representação de matriz de uma imagem. . . . .	p. 26
3.3	Imagem explicativa das cores. . . . .	p. 27
3.4	Imagem dos cones humanos. . . . .	p. 27
3.5	Imagem que exemplifica o comportamento da luz no ar. . . . .	p. 28
3.6	Imagem que exemplifica a queda da amplitude da onda. . . . .	p. 28
3.7	Imagem que exemplifica a dispersão da luz. . . . .	p. 29
3.8	Imagem que exemplifica o gradiente de luminosidade criado a partir de uma única fonte de luz. . . . .	p. 29
3.9	Ilustração do Sistema de Cor HSV. . . . .	p. 30
3.10	Ilustração do Sistema de Cor RGB. . . . .	p. 31
3.11	Exemplo de cores em RGB. . . . .	p. 31
3.12	Método de calibragem. . . . .	p. 32
3.13	Imagem da visão funcionando. . . . .	p. 33

3.14	Exemplo de uma técnica baseada em marcos, SIFT em questão. . . . .	p. 33
3.15	Webcam utilizada durante a competição (Microsoft HD 5000) . . . . .	p. 34
3.16	Parte do código de visão. . . . .	p. 34
3.17	Representação do campo. . . . .	p. 35
3.18	Resultado do filtro inRange. . . . .	p. 35
3.19	Resultado do filtro medianBlur. . . . .	p. 36
3.20	Resultado do filtro findcontours. . . . .	p. 37
3.21	Ilustração os pontos referentes ao um retângulo no OpenCV. . . . .	p. 37
4.1	Comportamento de um Campo elétrico. . . . .	p. 40
4.2	Campo vetorial. . . . .	p. 40
4.3	Campo vetorial de repulsão ao redor de um obstáculo. . . . .	p. 41
4.4	Campo vetorial de atração ao redor de uma meta. . . . .	p. 42
4.5	Ilustração da força empregada nas rodas esquerda e direita respectivamente. . . . .	p. 43
4.6	Casos definidos pelo método de Goodrich para força repulsiva. . . . .	p. 44
4.7	O resultado do método de Goodrich para força repulsiva. . . . .	p. 44
4.8	Casos definidos pelo método de Goodrich para força atrativa. . . . .	p. 45
4.9	O resultado do método de Goodrich para força atrativa. . . . .	p. 45
4.10	Ilustração do resultado dos cálculos empregados no método de Goodrich. . . . .	p. 46
4.11	Situação de gol contra. . . . .	p. 47
4.12	Ilustração do funcionamento dos campos localmente orientados. . . . .	p. 47
5.1	Ilustração de um sistema de malha aberta. . . . .	p. 48
5.2	Ilustração de um sistema de malha fechada. . . . .	p. 48
5.3	Ilustração do comportamento de um sistema em malha aberta. . . . .	p. 49
5.4	Planta de um sistema com o controle PID. . . . .	p. 50
5.5	Fenômeno do Overshoot. . . . .	p. 51
5.6	Comparação de diferentes técnicas. . . . .	p. 52

5.7	Erros considerados no sistema. . . . .	p. 52
5.8	Respostas do sistema a diferentes parâmetros. . . . .	p. 54
6.1	Robôs criados para a categoria VSS. . . . .	p. 56
6.2	Arduino Uno. . . . .	p. 57
6.3	Controlador usado. . . . .	p. 58
6.4	Pwm. . . . .	p. 59
6.5	Explorer utilizado para a transmissão. . . . .	p. 60
6.6	Shield utilizado para a comunicação entre o XBee e o Arduino. . . . .	p. 60
6.7	Mensagem transmitida para os robôs. . . . .	p. 60
6.8	Motor e rodas utilizadas. . . . .	p. 61
7.1	Chave de torneio. . . . .	p. 64
9.1	Controle contra travamentos dos robôs em campo. . . . .	p. 67
9.2	Representação dos jogadores em campo. . . . .	p. 68
9.3	Cálculo da projeção da bola e verificação de gol contra. . . . .	p. 68
9.4	Cálculo da reta para o gol. . . . .	p. 69
10.1	Método reescrito na biblioteca padrão do Arduino SoftwareSerial . . . . .	p. 70
10.2	Código referente a recepção dos dados pelo Arduino. . . . .	p. 71

## *Lista de Tabelas*

5.1	Tabela Ziegler-Nichols . . . . .	p. 55
7.1	Grupo A . . . . .	p. 62
7.2	Grupo B . . . . .	p. 62
7.3	Grupo C . . . . .	p. 62
7.4	Grupo D . . . . .	p. 63
7.5	Resultados grupo A . . . . .	p. 63
7.6	Resultados grupo B . . . . .	p. 63
7.7	Resultados grupo C . . . . .	p. 63
7.8	Resultados grupo D . . . . .	p. 63
7.9	Resultados quartas de final . . . . .	p. 64
7.10	Resultados semifinais . . . . .	p. 65
7.11	Resultado final . . . . .	p. 65
7.12	Resultado disputa terceiro lugar . . . . .	p. 65

# *Sumário*

<b>1</b>	<b>Introdução</b>	p. 16
1.1	RoboCup e SIR Lab . . . . .	p. 16
1.2	IEEE Very Small Size . . . . .	p. 17
1.2.1	Campo . . . . .	p. 18
1.2.2	Robôs . . . . .	p. 19
1.2.3	Partida . . . . .	p. 20
1.3	Objetivos . . . . .	p. 21
<b>2</b>	<b>Aspectos Gerais do Desenvolvimento</b>	p. 22
2.1	Divisão do Problema . . . . .	p. 22
2.2	Tecnologias Utilizadas . . . . .	p. 23
<b>3</b>	<b>Visão Computacional</b>	p. 25
3.1	Implementação do sistema de Visão . . . . .	p. 25
3.1.1	OpenCV . . . . .	p. 26
3.2	Tipo de Dados . . . . .	p. 26
3.3	O que é Cor . . . . .	p. 27
3.4	Sistemas de Cor . . . . .	p. 28
3.4.1	Gradiente de Luminosidade . . . . .	p. 28
3.4.2	HSV . . . . .	p. 30
3.4.3	RGB . . . . .	p. 30
3.5	Calibragem do Sistema . . . . .	p. 32

3.6	Algoritmo de Visão . . . . .	p. 33
<b>4</b>	<b>Navegação</b>	p. 39
4.1	Campos Potenciais . . . . .	p. 39
4.1.1	Campo Elétrico . . . . .	p. 39
4.2	Khatib . . . . .	p. 41
4.2.1	Campo Repulsivo . . . . .	p. 41
4.2.2	Campo Atrativo . . . . .	p. 42
4.2.3	Força Resultante . . . . .	p. 42
4.3	Goodrich . . . . .	p. 43
4.3.1	Campo Repulsivo . . . . .	p. 43
4.3.2	Campo Atrativo . . . . .	p. 45
<b>5</b>	<b>Controle</b>	p. 48
5.1	Controle PID . . . . .	p. 49
5.1.1	Proporcional . . . . .	p. 50
5.1.2	Integral . . . . .	p. 51
5.1.3	Derivativo . . . . .	p. 51
5.2	Implementação . . . . .	p. 52
5.3	Parametrização . . . . .	p. 54
5.3.1	Ziegler-Nichols . . . . .	p. 55
<b>6</b>	<b>Robôs</b>	p. 56
6.1	Construção . . . . .	p. 56
6.1.1	Arduino . . . . .	p. 57
6.1.2	Controlador de Motor . . . . .	p. 58
6.1.3	XBee . . . . .	p. 59
6.1.4	Outros componentes . . . . .	p. 61

<b>7 Resultados</b>	p. 62
7.1 Equipes participantes . . . . .	p. 62
7.2 Fase de grupos . . . . .	p. 63
7.3 Fase Eliminatória . . . . .	p. 64
<b>8 Conclusões e Trabalhos Futuros</b>	p. 66
<b>9 APÊNDICE A - Núcleo da Rotina de Estratégia</b>	p. 67
<b>10 APÊNDICE B - Núcleo da Rotina de Transmissão</b>	p. 70
<b>Referências</b>	p. 73



# 1 *Introdução*

A importância de fomentar conhecimento é indiscutível, o investimento em pesquisa vem aumentando consideravelmente como pode ser visto em [1], esse aumento se deve ao fato de que a ciência e a tecnologia movem o mundo. Uma das maneiras de fomentar conhecimento são as competições, que por meio da interação entre vários indivíduos que visam um ganho seja de competências e ou crescimento conceitual de suas instituições de ensino, indiretamente acabam apoiando o desenvolvimento tecnológico.

## 1.1 **RoboCup e SIR Lab**

Uma competição que podemos citar que contribui para o fomento da robótica e inteligência artificial em âmbito global é a RoboCup[2], a competição possui diversas categorias, por exemplo: *Simulation 2D, Simulation 3D, Small Size, Middle Size, Standard Platform, Humanoid, Rescue Simulation, Rescue Robots, Home, Work, Logistics League Sponsored by FESTO, Junior Dance, Junior Soccer, Junior Rescue, Junior CoSpace*. Na figura 1.1 encontra-se a logomarca da competição, a qual possui a seguinte visão:

"Em meados do século 21, uma equipe totalmente autônoma de robôs humanoides jogadores de futebol deve vencer um jogo contra o time de humanos campeão da última Copa do Mundo da FIFA, utilizando as regras da FIFA."

(RoboCup Federation)



Figura 1.1: Logomarca da competição.

A RoboCup, além de ser uma competição mundial, possui outras competições em âmbito Estadual, Nacional e Continental, como a Olimpíada Brasileira de Robótica (OBR)[3], Competição Brasileira de Robótica (CBR)[4] e *Latin American Robotics Competition* (LARC)[5]. Visando a participação nesses eventos surgiu o Laboratório de Sistemas Inteligentes e Robótica (SIRLAB)[6] da FAETERJ-Petrópolis[7], que participou pela primeira vez de um evento nacional em 2013 competindo na CBR categoria IEEE *Very Small Size*[8] com a equipe SirSoccer e competindo pela segunda vez na mesma categoria no ano de 2014 na LARC. Na figura 1.2 encontra-se a logomarca do SIR Lab.



Figura 1.2: Logomarca do Laboratório de Sistemas Inteligentes e Robótica.

## 1.2 IEEE Very Small Size

A categoria *Very Small Size*(VSS) trata-se de uma competição de futebol de robôs, que possui o mesmo objetivo de um jogo de futebol normal que é fazer gols e vencer a partida, porém com algumas adaptações. Essa categoria é regulamentada pelo Instituto de Engenheiros Eletricistas e Eletrônicos (IEEE)[9] e possui regras baseadas na MiroSot[10] que orientam desde a construção do campo e robôs até situações de jogo como falta, tiro livre, pênalti e início de partida. De todos os aspectos de um robô, deve-se entender como é possível que um sistema robótico execute alguma função, qualquer sistema robótico é derivado de três primitivas básicas[11] que constituem a inteligência de um robô, são elas: Sentir, Planejar e Atuar, por exemplo, imagine que deva-se criar um robô com o objetivo de retirar as roupas do varal quando começasse a chover, para isso o robô deve perceber que começou a chover com a ajuda de algum sensor (sentir), deve planejar uma ação e deve executar a ação, o futebol de robôs também está preso nessas três primitivas, porém estão contidas dentro de um problema diferente, a categoria VSS define que o nosso sensoriamento deve ser feito a partir de uma câmera posicionada a 2 metros de altura no centro do campo, o nosso planejamento está todo voltado em como os robôs devem se comportar durante uma partida e também há a execução do que

foi planejado. Na figura 1.3 encontra-se a estrutura de funcionamento da categoria. Imagem retirada de [8].

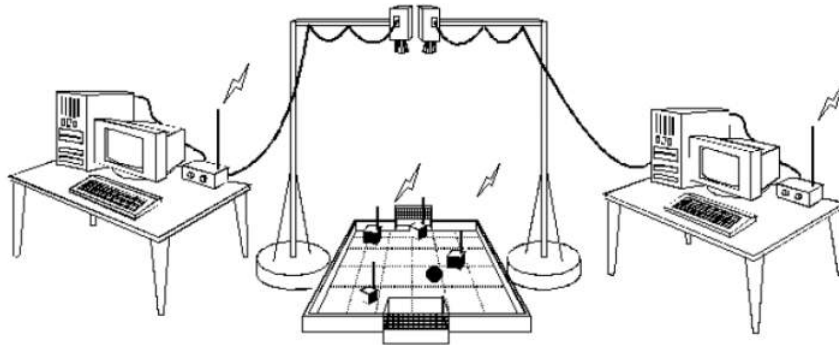


Figura 1.3: Estrutura de funcionamento da categoria VSS.

A estrutura deixa claro o funcionamento da categoria, onde existe uma câmera ligada a uma central de processamento que define os comandos a serem executados pelos robôs e transmite essas ações para os mesmos. Com base nas regras definidas para a competição pode-se agrupá-las em três áreas, regras de construção do campo, regras de construção dos robôs e regras de partida.

### 1.2.1 Campo

As regras de construção do campo são importantes, pois é necessário padronizar o campo utilizado dentre todos os anos de competição para que todas as equipes estejam preocupadas em resolver os mesmos problemas.

- As dimensões do campo devem ser 150x130cm e as paredes desse campo devem possuir 5cm de altura e 2,5cm de espessura;
- O campo deve ser da cor preta fosco não reflexivo;
- O campo deve conter 6 marcações que servem para orientar o posicionamento dos robôs em casos específicos;
- As paredes por dentro devem ser da cor branca;
- As marcações no campo devem ser da cor branca;
- Nos cantos das paredes devem haver pentaedros cortados de maneira que a câmera os veja como triângulos;

- A bola deve ser da cor Laranja;

Na figura 1.4 encontra-se o campo do VSS. Imagem retirada de [8].

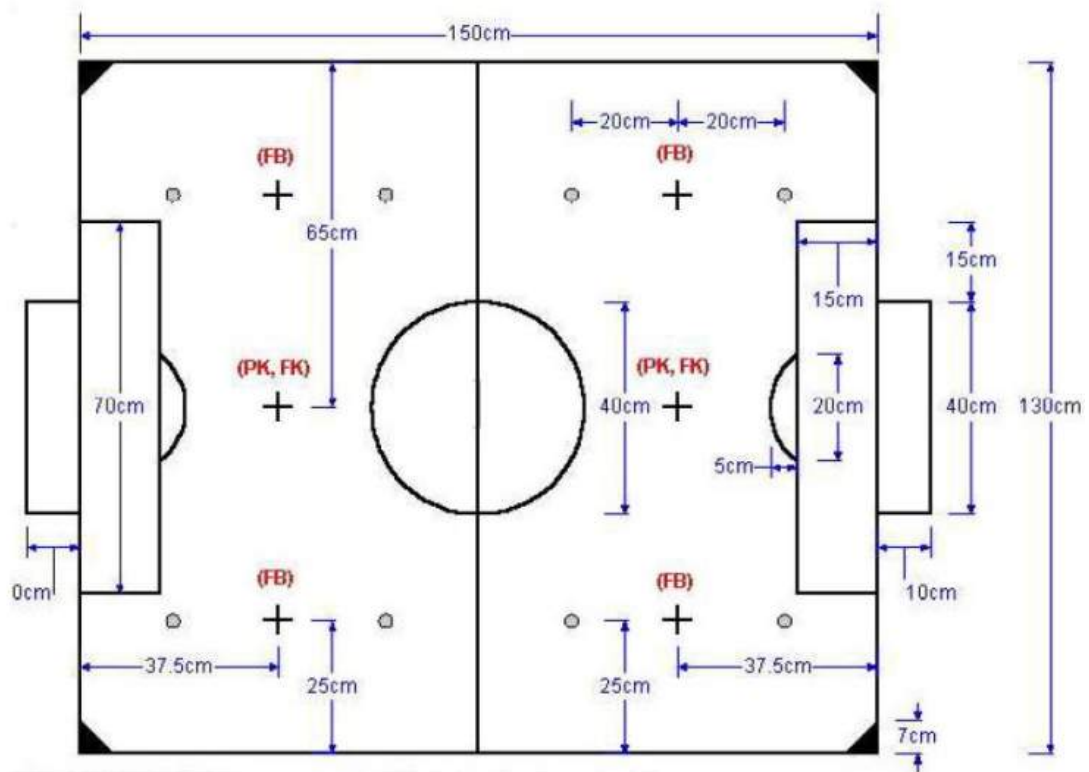


Figura 1.4: Campo da categoria com suas marcações de chute-livre(FB), Faltas(FK) e Pênaltis(PK).

Mais informações sobre as marcações do campo podem ser encontradas em [8].

## 1.2.2 Robôs

As regras referentes à construção dos robôs servem para limitar alguns aspectos importantes em relação ao espaço disponível, pois esse espaço define exatamente quais componentes devem e podem ser utilizados.

- Os robôs não devem possuir mais de 7,5cm de lado;
- Uma proteção pode ser utilizada, desde que os lados do robô em sua totalidade não ultrapasse 8cm;
- Os robôs podem possuir pernas, braços, dispositivos de chute e uma cova para prenderem a bola, desde que no máximo 30 por cento da área da bola seja coberta, como na figura 1.5;

- O topo dos robôs deve ser preto;
- No topo de cada robô deve haver uma etiqueta para identificação de um time que deve ser obrigatoriamente Azul ou Amarelo e devem possuir pelo menos 9cm<sup>2</sup>;
- No topo dos robôs pode haver apenas mais uma etiqueta para a identificação desde que não seja das cores Branca, Laranja, Azul ou Amarela;
- Não podem haver fios ligados aos robôs e os mesmos devem ser autônomos;

Na figura 1.5 encontra-se a exemplificação da área coberta da bola pelo robô. Imagem retirada de [8].

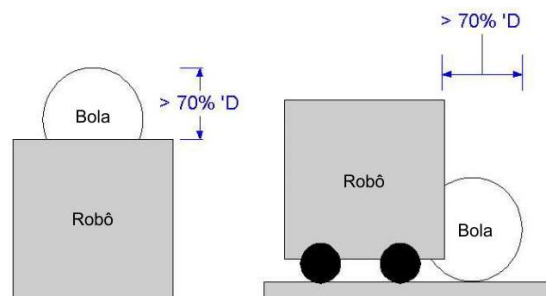


Figura 1.5: Área máxima da bola que pode ser coberta.

### 1.2.3 Partida

As regras de partida são importantes para orientar como o jogo deve acontecer, definindo as regras básicas que devem ser respeitadas.

- A partida dura 10 minutos com dois tempos de 5 minutos;
- Há um intervalo de 10 minutos entre um tempo e outro;
- Cada time tem direito a dois tempos de 2 minutos que podem ser pedidos a qualquer momento;
- Caso a diferença de gols entre os dois times chegue a 10 a partida é encerrada;
- Uma falta ocorre quando há mais de um robô de um mesmo time dentro de sua própria área de gol ou quando um robô empurrar outro robô de outro time;
- Um pênalti ocorre quando a bola fica mais de 10 segundos dentro de alguma das áreas;

- Um chute-livre ocorre quando os robôs ficam travados por mais de 10 segundos, caso ocorra, o juiz posiciona a bola na marca de chute-livre mais próxima de onde ela ficou parada e posiciona os robôs de cada time equidistantes a bola;
- A cada início de partida ou gol feito a bola deve ser posicionada no centro do campo e os robôs devem ser posicionados de acordo com a posse de bola.

Mais informações sobre as regras podem ser encontradas em [8].

### 1.3 Objetivos

A categoria VSS é um ambiente dinâmico, imprevisível e que possui uma grande quantidade de variáveis que podem interferir no sistema, por isso é improvável que crie-se uma solução única que seja capaz de se comportar bem contra qualquer time e em qualquer situação, o que faz com que uma partida de futebol de robôs autônomos seja uma plataforma adequada para testar diversas teorias, algoritmos, arquiteturas e desempenhos como descrito em [12]. Sabendo que a construção de um time de robôs autônomos para a categoria VSS não é algo inédito, a finalidade desse trabalho é que o mesmo possa vir a servir como referência para outras equipes e apresentar os algoritmos propostos para cada exigência do problema, esses sim de caráter inovador. A figura 1.6 retrata o primeiro jogo da equipe da SirSoccer durante a LARC de 2014 contra a equipe Autobotz da UFMG.



Figura 1.6: Partida entre SirSoccer (FAETERJ) e Autobotz (UFMG).

## 2 *Aspectos Gerais do Desenvolvimento*

Nesse capítulo será apresentada uma visão mais superficial do problema tratado em questão e descrito como o sistema da equipe SirSoccer foi construído.

### 2.1 **Divisão do Problema**

A primeira abordagem feita sobre o problema o dividiu em três áreas de estudos, Visão Computacional, Estratégia e Criação dos Robôs. Porém existem diversos artefatos camuflados por baixo dessa abordagem, por exemplo: Técnicas de Controle, Navegação, Persistência de Dados e Comunicação. Optou-se por separar o programa em duas rotinas para que fosse mais fácil lidar com o sistema, a rotina de calibragem e a rotina de jogo que possui várias sub-rotinas a fim de modularizar o código, totalizando as seguintes rotinas.

**Calibragem:** Rotação de imagem, corte de imagem, retificação de imagem, calibragem das cores dos robôs, da bola e de pontos estratégicos.

**Persistência:** Responsável por salvar a calibragem feita e carregar os dados para a rotina de jogo.

**Visão Computacional:** Localização dos robôs e da bola em campo.

**Estratégia de Jogo:** Comandos programados para execução em determinadas situações.

**Campos Potenciais:** Técnica usada para evitar colisões entre robôs, colisões com paredes e ditar pontos para onde os robôs devem ir.

**Controle PID:** Movimentação dos robôs.

**Transmissão de Dados:** Transmissão dos comandos do computador para os robôs.

**Recepção de Dados:** Recepção dos dados transmitidos pelo computador.

**Execução:** Transferência dos comandos recebidos para ações no motor.

O sistema segue o fluxograma presente na figura 2.1:

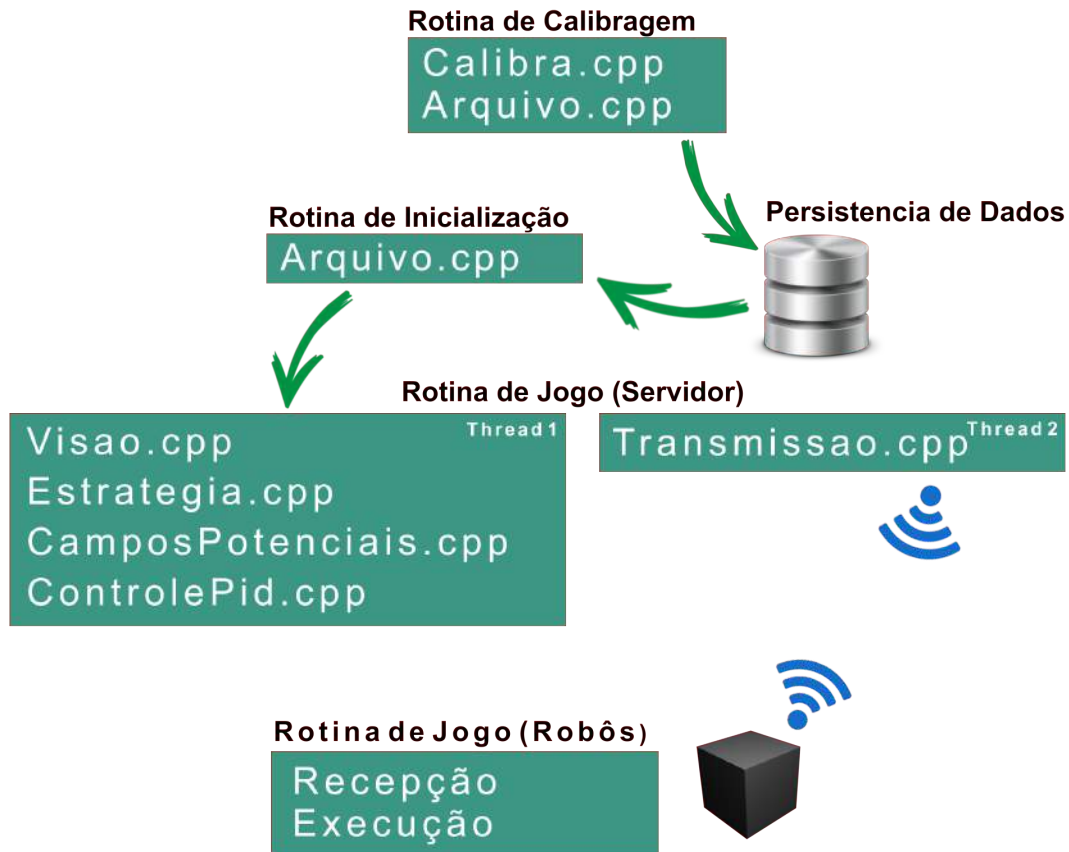


Figura 2.1: Fluxograma do sistema da equipe SirSoccer.

## 2.2 Tecnologias Utilizadas

Após possuir o planejamento do sistema optou-se por utilizar a linguagem de programação C++ para implementar-se a parte da solução do computador, pois além de ser rápida e possuir compatibilidade com várias bibliotecas que seriam usadas na construção do sistema, ainda haveria a possibilidade utilizar bibliotecas de C, optou-se ainda por separar o núcleo de transmissão em uma *thread* diferente de todo o resto, pois verificou-se que o micro-transmissor usado necessitava de um *delay* para garantir um bom funcionamento e isso poderia fazer com que todo o sistema fosse prejudicado, já o código responsável pelo funcionamento do Robô é C, pois a plataforma Arduino que foi usada trabalha unicamente com essa linguagem. Foram implementados também vários modos de execução no projeto, como:

**Interativo/Debug:** Onde é possível os usuários controlarem os robôs.



**Treinamento:** Onde é possível fazer com que o sistema teste alguns valores de parametrização, como, relação angulação robô x bola, para definição da potencia nas rodas do robô, assim escolhendo a melhor relação.

**Otimização:** Simplifica a exibição das informações na tela, fazendo com que o sistema torne-se mais "leve".

**Espectro:** Mostra tudo que a Visão Computacional retorna durante a partida.

**Imagem:** Trabalha com imagens salvas para o desenvolvimento do sistema mesmo que não seja possível ter toda a estrutura montada.

Utilizou-se o paradigma de orientação a objetos para desenvolver todo o sistema, foi criado ainda um *namespace* para o laboratório que guarda todas as *structs*, *enums* e cálculos matemáticos comuns a várias partes do programa. Os detalhes do funcionamento de cada sub-rotina serão explorados mais a frente e a disposição dos capítulos encontra-se na ordem de execução do algoritmo criado. Na figura 2.2 encontra-se o sistema da equipe SirSoccer.

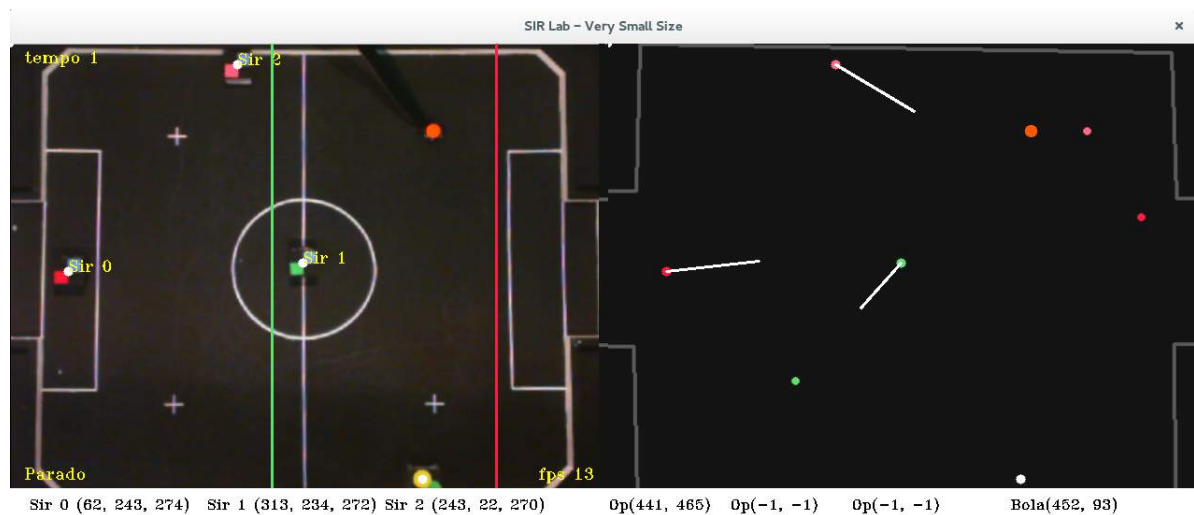


Figura 2.2: Imagem do sistema em funcionamento.

## 3 *Visão Computacional*

Segundo [13], Visão Computacional é o conjunto de métodos e técnicas através dos quais sistemas computacionais podem ser capazes de interpretar imagens, ou seja, extrair informações de imagens. Um programa de Visão Computacional segue a mesma lógica de um programa normal de computador que é formado por um conjunto de instruções de ordem logicamente definida onde dados são inseridos, processados e retornados. Observando o problema encarado na construção do time de robôs, onde sempre é requisitado a distância entre os robôs e a bola, pode-se calcular a distância entre os elementos em campo com uma fita métrica, inserir as informações no programa e fazer com que ele execute alguma ação, porém como estamos lidando com um problema que requer trabalho em tempo real isso é inviável. Para executar o trabalho de extrair as informações do jogo é utilizado uma câmera que capta todo o campo, pois pode-se criar um programa que reconheça os robôs automaticamente e calcule a distância entre os elementos. A única diferença entre um programa de Visão Computacional e um programa normal é o dado utilizado para processar que no nosso caso é uma imagem.

### 3.1 **Implementação do sistema de Visão**

Para a implementação da rotina de Visão Computacional foi utilizado a biblioteca *open-source* OpenCV[14], pois a mesma trabalha com processamento de imagens em tempo real e é possível desenvolver soluções utilizando linguagens de programação como C, C++, CSharp, Java, Python dentre outras. Apesar de existir a possibilidade de trabalhar com linguagens de mais alto nível em relação a C++, o que possivelmente aceleraria o processo de criação houve preferência de utilizar a mesma, pois além de ser leve[15] é extremamente robusta[16] e por isso não haveria problemas em relação a velocidade de processamento.

### 3.1.1 OpenCV

A biblioteca OpenCV foi criada em 2000 pela empresa Intel com o propósito de ser utilizada no desenvolvimento de programas na área de Visão Computacional, a biblioteca, além de contar com implementações voltadas para áreas de Estrutura de Dados, Álgebra Linear, Interface de Usuário, possui mais de 350 algoritmos de visão computacional que forneceram auxílio na criação do sistema. Na figura 3.1 encontra-se a logomarca da biblioteca.



Figura 3.1: Logomarca da biblioteca OpenCV.

## 3.2 Tipo de Dados

Olhando para o nosso tipo de dados[17] em questão, uma imagem nada mais é do que uma matriz que pode ter diversas resoluções como, por exemplo: 640x480, 720x480, 1920x1080. A resolução de uma imagem indica o número de linhas e colunas que a imagem possui, e para obter a quantidade de pixels presentes na imagem basta multiplicarmos os valores, no caso de uma imagem 1080p temos 2073600 pixels, cada pixel guarda valores referentes a uma cor. Na figura 3.2 encontra-se a exemplificação de como um computador interpreta uma imagem.

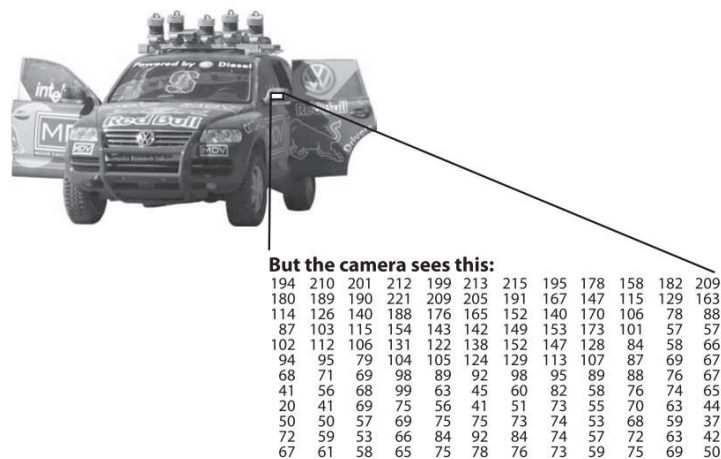


Figura 3.2: Representação de matriz de uma imagem.

### 3.3 O que é Cor

Antes do início do desenvolvimento buscou-se entender com o que o estariasse lidando, já que a localização dos robôs em campo é dada pela cor de suas etiquetas. Segundo [18], cor nada mais é do que a percepção humana dos feixes de luz que são emitidos em diferentes frequências e a diferença dessas frequências também diferem na nossa percepção. Na figura 3.3 encontra-se uma exemplificação da diferença dessas frequência e na figura 3.4 encontra-se o motivo.

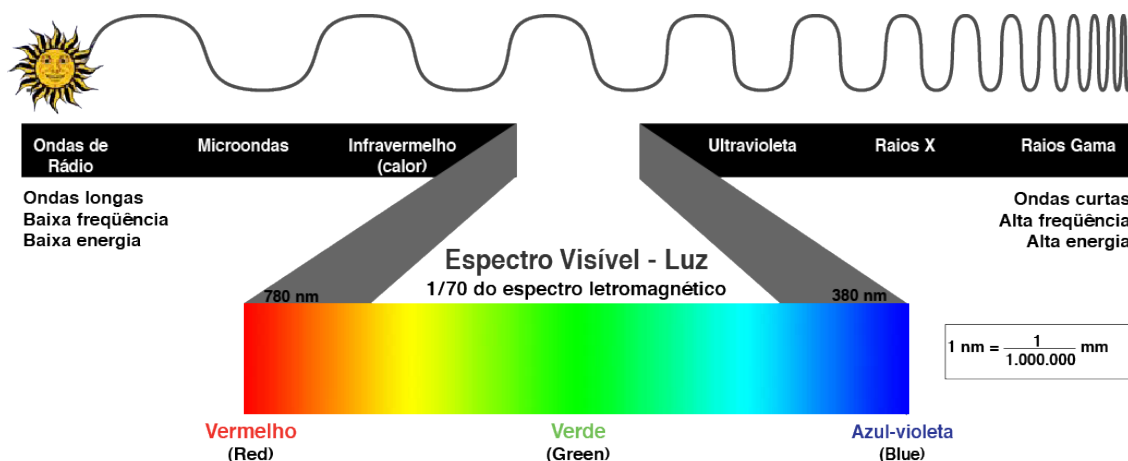


Figura 3.3: Imagem explicativa das cores.

O ser humano possui células especiais na retina responsáveis por reconhecer a luminosidade e diferentes tons de cor, em geral possuísse três tipos de cones responsáveis por reconhecer as frequências de ondas vermelhas, verdes e azuis, e bastonetes responsáveis por reconhecer a luz. Na figura 3.4 encontra-se a ilustração das células especiais.

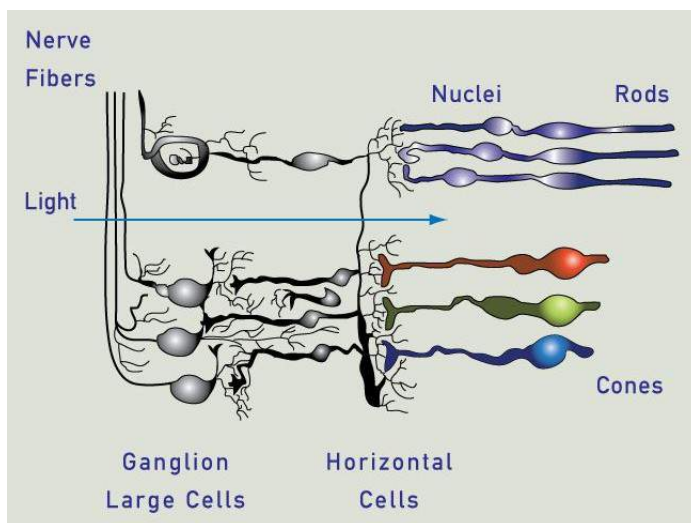


Figura 3.4: Imagem dos cones humanos.

## 3.4 Sistemas de Cor

Uma vez que é necessário trabalhar com cores em um ambiente computacional, é preciso representá-las de alguma forma, essa representação é feita com auxílio de algum sistema de cor[19], o conteúdo de cada pixel depende do sistema de cor utilizado, o padrão mais utilizado é o RGB (*Red, Green and Blue*) que foi criado com o propósito de retratar as cores de maneira próxima a realidade, mas existem diversos sistemas como, por exemplo: HSV (*Hue, Saturation and Value*), YUV, SCT e etc. Durante o período que passou-se estudando esses sistemas houve trabalho principalmente com o RGB[20] e o HSV[21]. Um dos problemas que influenciaram diretamente na escolha do sistema de cor é o gradiente de luminosidade.

### 3.4.1 Gradiente de Luminosidade

Pelo fato da luz deslocar-se como onda e possuímos um fluido no qual ela sofre influência que é o ar, a medida que ela se distânciava de sua fonte sua amplitude cai, como se jogasse uma pedra em um lago. Nas figuras 3.5 e 3.6 encontram-se a exemplificação da queda de amplitude.



Figura 3.5: Imagem que exemplifica o comportamento da luz no ar.

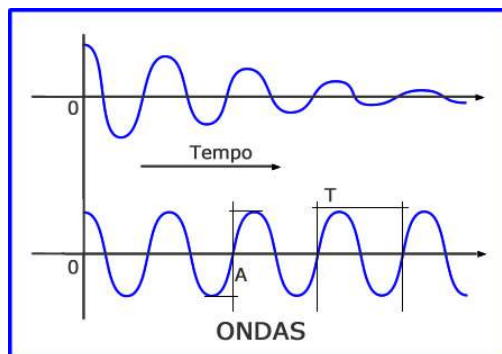


Figura 3.6: Imagem que exemplifica a queda da amplitude da onda.

Imaginando que exista uma única fonte de luz no centro do campo, como a luz choca-se

com os objetos e é refletida de volta para a câmera, o fato de o campo ser plano o meio do campo recebe uma quantidade de luz maior do que suas extremidades, pois quanto mais ao extremo do campo maior é a distância percorrida pela luz fazendo com que ela perca energia no caminho. Na figura 3.7 encontra-se a exemplificação da disposição da luz no sistema.

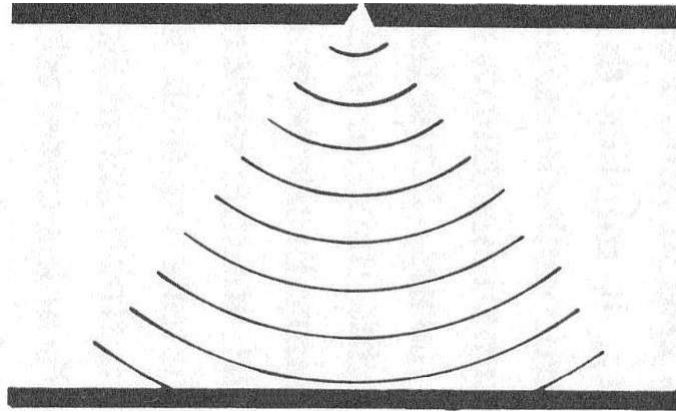


Figura 3.7: Imagem que exemplifica a dispersão da luz.

Após chocar-se com o objeto há ainda a reflexão onde a luz refletida do centro é mais captada pela a câmera do que suas extremidades, assim cria-se um gradiente de luminosidade no campo. Na pratica, nota-se uma diferença no brilho das cores captadas pela câmera, onde as cores do centro da imagem são muito mais claras do que suas extremidades. Na figura 3.8 encontra-se a diferença de luminosidade percebida pela camera caso a fonte de luz encontra-se no centro do campo.

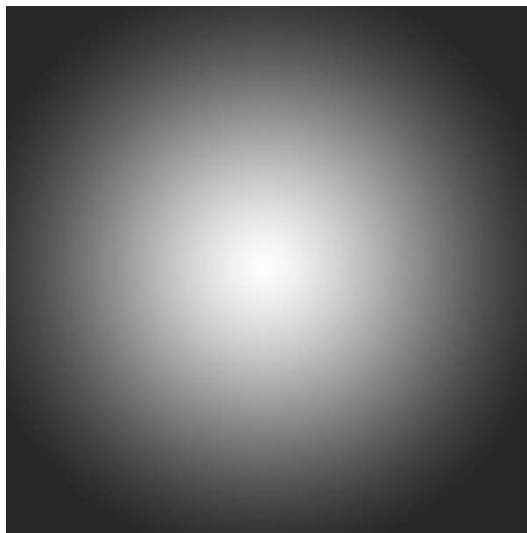


Figura 3.8: Imagem que exemplifica o gradiente de luminosidade criado a partir de uma única fonte de luz.

### 3.4.2 HSV

O padrão HSV foi o primeiro padrão a ser cogitado para o uso do sistema devido a fácil representação que é dividida em três parâmetros que são *Hue*, *Saturation* e *Value*. Na figura 3.9 encontra-se a representação do sistema HSV.

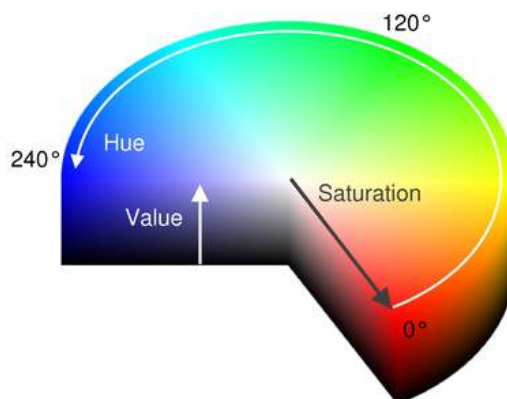


Figura 3.9: Ilustração do Sistema de Cor HSV.

Como podemos ver a representação acima o padrão é bastante intuitivo. O valor do *Hue* é a cor propriamente dita que possui a representação de uma circunferência, o valor de *Saturation* demarca a intensidade daquela cor no pixel, onde o valor mínimo (0) é a cor sem vida praticamente cinza e o máximo (255) a cor em sua maior tonalidade, já o *Value* demarca o brilho presente daquela cor que também varia de 0 a 255. A vantagem de trabalhar-se com o sistema HSV é que o mesmo é mais independente da variação de luminosidade, pois seguindo a lógica de funcionamento, utilizando toda a gama de valores presentes de Saturação e Brilho e apenas trabalhando-se o valor da cor, não haverá problemas para rastrear um objeto, pois o intervalo de cor abrangerá todo o espectro da cor, assim contornando o empecilho das variações de luminosidade no campo.

### 3.4.3 RGB

Embora o sistema de cor HSV possua vantagem em relação ao gradiente de luminosidade, após o estudo de outros padrões como o RGB foi decidido utilizá-lo, pois além do mesmo ser o padrão mais utilizado, após vários testes no laboratório notou-se que tal diferença de luminosidade não era tão brusca e não haveria chance de confusão de cores em todos os cantos do gradiente, apenas haveria a necessidade de calibrar um intervalo referente aos valores máximos e mínimos do gradiente, o padrão RGB foi criado inspirado na percepção humana das cores, ele é formado por três atributos o R, G e B que representam respectivamente os cones humanos

responsáveis por captar as frequências de vermelho, verde e azul. Na figura 3.10 encontra-se a representação do sistema RGB.

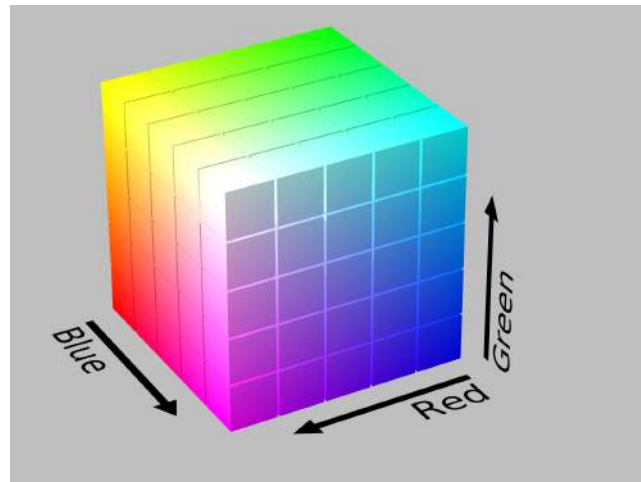


Figura 3.10: Ilustração do Sistema de Cor RGB.

Diferente do HSV o RGB não trata o brilho da cor com base em um único parâmetro e sim com todos os parâmetros e a tonalidade de uma cor é dada com base na mistura das cores, logo se os valores de RGB são os mesmos a cor resultante pertencerá a uma escala de cinza, pois é a mistura de todas ou nenhuma cor, e se possuir uma cor cuja o RGB possua apenas um atributo o brilho daquela cor dependerá do valor daquele atributo. Na figura 3.11 encontra-se exemplos de cores para determinados valores de RGB.












Color	HTML/CSS Name	Hex Code #RRGGBB	Decimal Code (R,G,B)
	Black	#000000	(0,0,0)
	White	#FFFFFF	(255,255,255)
	Red	#FF0000	(255,0,0)
	Lime	#00FF00	(0,255,0)
	Blue	#0000FF	(0,0,255)
	Yellow	#FFFF00	(255,255,0)
	Cyan / Aqua	#00FFFF	(0,255,255)
	Magenta / Fuchsia	#FF00FF	(255,0,255)
	Silver	#C0C0C0	(192,192,192)
	Gray	#808080	(128,128,128)
	Maroon	#800000	(128,0,0)

Figura 3.11: Exemplo de cores em RGB.



## 3.5 Calibragem do Sistema

A rotina de calibragem trata de vários aspectos das imagens retornadas da câmera, calibra-se as seguintes cores para rastreamento: laranja, amarelo e azul obrigatoriamente, e as outras cores referentes aos jogadores em campo. Caso a câmera esteja posicionada a mais de 2 metros de altura o que faria com que ela captasse o campo e mais coisas ao redor devido ao ângulo de abertura da lente usada, existe a opção de cortar a imagem e rotacioná-la. Na figura 3.12 encontra-se a tela de calibragem do sistema.

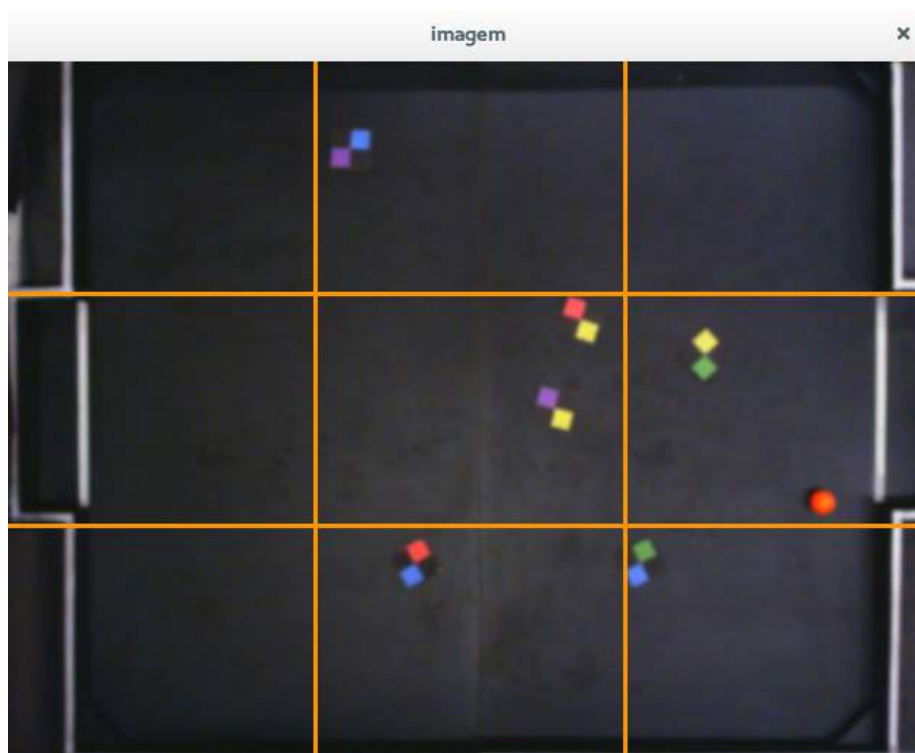


Figura 3.12: Método de calibragem.

Foi criado um método de calibragem onde o usuário clica em cima da cor que gostaria de calibrar com auxílio da função `setMouseCallback()` e assim pega os valores RGB do pixel salvando um intervalo de cor tratado como RGB máximo daquela cor e o mínimo, a medida que vão havendo os cliques o sistema verifica para cada atributo se ele é maior que o atributo máximo salvo ou menor que mínimo salvo, caso seja, o mesmo assume o lugar de menor ou maior, esse processo deve ser repetido para os nove cantos assinalados na figura 3.12, e para todas as cores que devem ser calibradas. Calibra-se também um valor de rotação da imagem, pois durante os preparativos de um jogo há sempre a chance de alguém esbarrar na câmera e desalinha-la por isso houve a preferência de rotacionar a imagem via *software* com auxílio da função `warpAffine()`. Todos os valores são guardados em textos planos(TXTs) com o uso da classe `Arquivo` criada.

## 3.6 Algoritmo de Visão

Quanto maior a resolução mais detalhes a imagem carrega, mais espaço ela ocupa e mais custoso é seu processamento, é necessário escolher muito bem a resolução usada, pois imagens com resoluções muito altas são custosas para se processar e pelo fato dos robôs estarem em constante movimento durante a partida é necessário que a visão trabalhe de maneira rápida o suficiente para que se consiga ter poder de reação, todas as funções citadas dentro dessa seção podem ser encontradas em [22]. Na figura 3.13 encontra-se o sistema de visão funcionando.



Figura 3.13: Imagem da visão funcionando.

A rotina de Visão Computacional é a mais complexa de todo o sistema, foram testadas técnicas que poderiam ser utilizadas no rastreamento como, por exemplo: Blob Detection[23], Surf[24] e Sift[25], porém após vários testes foi decidido que não iria se empregar nenhuma dessas técnicas e sim que criaria-se um algoritmo próprio, pois foi constatado que o custo computacional para executar técnicas baseadas em marcos é extremamente alto.

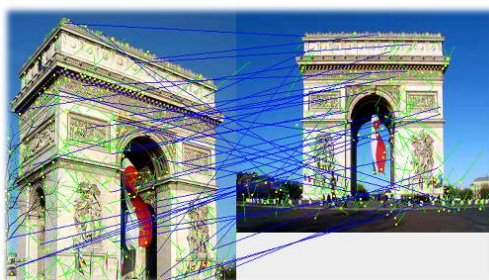


Figura 3.14: Exemplo de uma técnica baseada em marcos, SIFT em questão.

A câmera utilizada foi uma *Webcam* USB comum modelo Microsoft HD 5000 que possui capacidade de fornecer imagens com resolução de 720p a 30 Fps, além de possuir foco automático. Porém a mesma foi utilizada com a resolução 640x480 e foi utilizado o *software* Uvcdynctrl[26] para calibrar algumas funções como Contraste, Saturação e o desligamento do foco automático. Na figura 3.15 encontra-se a câmera utilizada durante a competição de 2014.



Figura 3.15: Webcam utilizada durante a competição (Microsoft HD 5000)

No algoritmo de Visão Computacional são utilizados alguns filtros e funções em uma determinada ordem para que haja a localização dos robôs e da bola em campo. Na figura 3.16 encontra-se a parte essencial do algoritmo criado.

```

estado.cor.clear();
for(int j = 0 ; j < 6 ; j++){
    vector<Point> jogo;

    //GaussianBlur(in, in, Size(3, 3), 3, 3);
    inRange(in,
            Scalar(config.cor[j].rgb[cMin][b]*(1-crang), config.cor[j]
            Scalar(config.cor[j].rgb[cMax][b]*(1+crang), config.cor[j]
            out);

    medianBlur(out, out, 3);
    findContours(out, contours, hierarchy, CV_RETR_TREE, CV_CHAIN_APPROX_SIMPLE);

    vector<vector<Point> > contours_poly( contours.size() );
    vector<Rect> boundRect( contours.size() );

    for( int i = 0; i < contours.size(); i++ ){
        approxPolyDP( Mat(contours[i]), contours_poly[i], 0, true );
        boundRect[i] = boundingRect( Mat(contours_poly[i]) );
    }

    for( int i = 0; i < contours.size(); i++ ){
        if(eUmaEtiqueta(boundRect[i])){

```

Figura 3.16: Parte do código de visão.

Dada a seguinte imagem para processamento considere que a calibragem da câmera está feita para reconhecer a cor amarela. Na figura 3.17 encontra-se a representação das cores básicas.

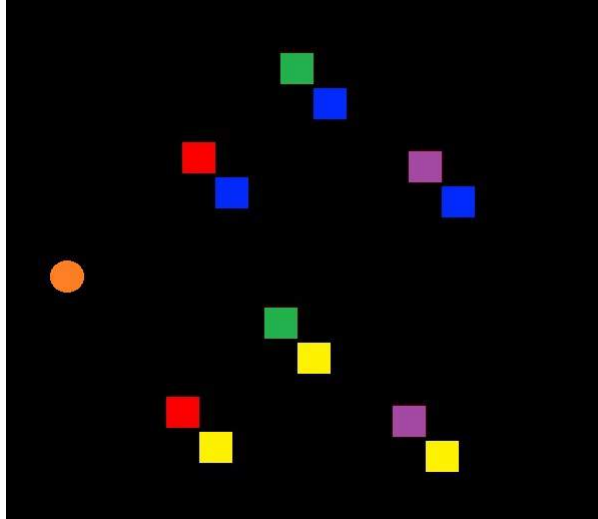


Figura 3.17: Representação do campo.

O primeiro filtro aplicado é o `inRange()`, ele responsável por varrer a imagem a procura de pixels contidos em um determinado intervalo de cor em qualquer sistema de cor e a partir disso gerar uma imagem binarizada onde os pixels contidos no intervalo tornam-se brancos e os que não estão dentro tornam-se pretos. Na figura 3.18 encontra-se o resultado do filtro `inRange()`.

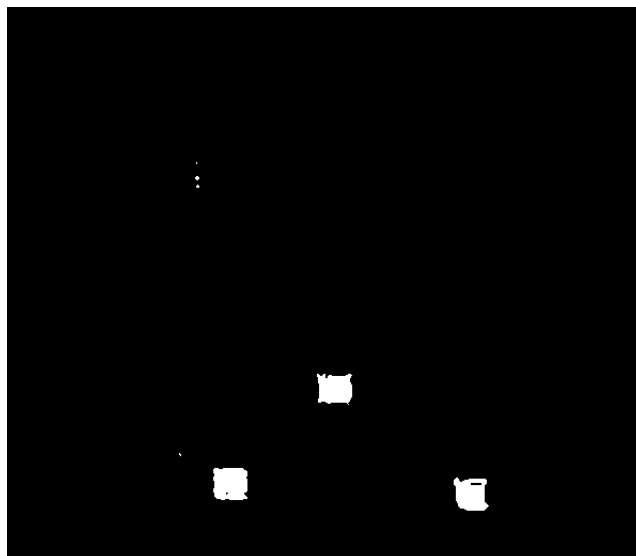


Figura 3.18: Resultado do filtro `inRange`.

Após a execução do `inRange()`, aplicasse o filtro da mediana para retirar o ruído da imagem binarizada e torna-la mais uniforme, assim facilitando o trabalho exercido pelo próximo filtro. Na figura 3.19 encontra-se o resultado do filtro `medianBlur()`.

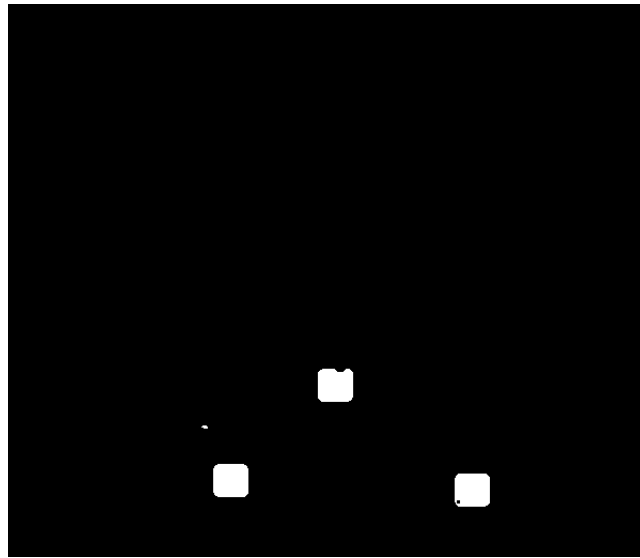


Figura 3.19: Resultado do filtro medianBlur.

O filtro da mediana funciona da seguinte forma. *Dada a seguinte matriz que deseja-se aplicar o filtro da mediana 3x3.*

$$\begin{pmatrix} 12 & 23 & 33 \\ 41 & 55 & 31 \\ 12 & 22 & 21 \end{pmatrix}$$

Primeiro é necessário ordenar os elementos da matriz, após ordenados identifica-se o elemento central e o substitui nos outros elementos da matriz.

12, 12, 21, 22, 23, 31, 33, 41, 55

Assim a matriz resultante é:

$$\begin{pmatrix} 23 & 23 & 23 \\ 23 & 23 & 23 \\ 23 & 23 & 23 \end{pmatrix}$$

Após aplicado o filtro da mediana, utiliza-se o filtro `findContours()` para achar os contornos, esse filtro a medida que identifica os contornos altera a imagem de saída e salva os pontos em um vetor(*Vector*). Todo esse processo é repetido para todas as cores utilizadas nas etiquetas dos robôs e para a bola, assim no final de cada iteração do algoritmo cria-se seis vetores que possuem pontos de um espectro de cor achado, após se conhecer os pontos há a necessidade de agrupa-los, para isso utiliza-se as funções `approxPolyDP()` e `boundingRect()` para criar-se um retângulo ao redor dos contornos achados pelo algoritmo. Na figura 3.20 encontra-se o resultado do filtro `findContours()`.

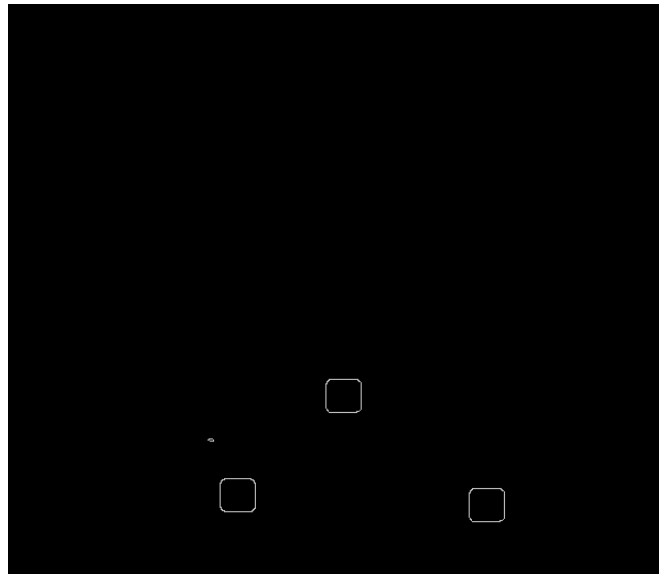


Figura 3.20: Resultado do filtro findcontours.

Na figura 3.21 encontra-se a representação de um retângulo empregada pela biblioteca OpenCV.

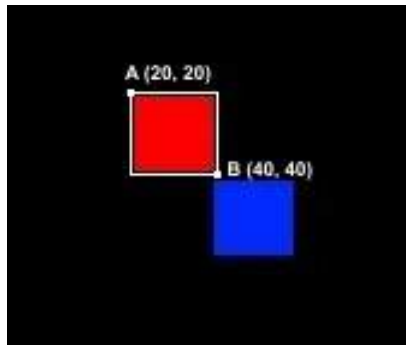


Figura 3.21: Ilustração os pontos referentes ao um retângulo no OpenCV.

Após conhecer-se a localização das etiquetas coloridas no topo dos robôs em campo, há a necessidade de encontrar o centro do retângulo que é dado pelo ponto médio.

$$P_m = \left( \frac{a_x + b_x}{2}, \frac{a_y + a_y}{2} \right) \quad (3.1)$$

Como a visão computacional está sujeita a inúmeras interferências, há a necessidade de filtrar esses retângulos com base em alguns parâmetros que são a área e a proporção Lado x Lado, como as etiquetas possuem um tamanho padrão, todo espectro que possuir uma área maior do que a área máxima delimitada no programa, ou menor do que a área mínima, ou uma diferença no comprimentos dos lados maior do que a considerada aceitável é ignorado pelo

algoritmo. Após a filtragem das informações com base no formato e área, verifica-se utilizando a distância euclidiana as cores conjuntas, por exemplo, a etiqueta amarela mais próxima a uma vermelha.

Para os pontos bidimensionais,  $P = (p_x, p_y)$  e  $Q = (q_x, q_y)$  a distância  $D$  é computada como:

$$D = \sqrt{(p_x - q_x)^2 + (p_y - q_y)^2} \quad (3.2)$$

Os retângulos que tiverem a menor distância são considerados pelo programa como sendo um robô, após a identificação de cada robô calcula-se o ângulo  $\theta$  um a um utilizando o arco tangente dado por:

$$\theta = \arctan\left(\frac{\Delta y}{\Delta x}\right) \quad (3.3)$$

Após o cálculo da angulação de cada robô, repete-se o cálculo do ponto médio, assumindo os pontos P e Q como sendo o centro dos retângulos que representam um robô, assim achando o ponto central do mesmo. A localização e orientação dos robôs em campo são importantes para que comande-se as ações dos mesmos.

## 4 *Navegação*

Um dos problemas básicos que devem ser muito bem pensados no *IEEE Very Small Size* é a navegação dos robôs em campo, uma vez que estamos trabalhando com um ambiente dinâmico, torna-se muito difícil descrever matematicamente e programar todos os possíveis casos que podem ocorrer durante um jogo, por exemplo, dado um gráfico de uma função qualquer que represente uma trajetória em um plano bidimensional (imagem retornada da câmera), imagine que possui-se em determinados pontos desse gráfico um robô e a bola, fazer com que nosso robô siga perfeitamente aquela trajetória gerada a fim de acertar a bola seria o ideal, porém na prática existem diversas coisas que podem fazer com que a ação definida não seja tangível como, por exemplo, um robô adversário interpor-se ao caminho gerado, o robô não seguir a trajetória gerada ou a direção da bola sofrer uma mudança de repente, situações que repetem-se o tempo todo durante uma partida.

### 4.1 Campos Potenciais

Partindo do princípio que não há uma solução de movimentação única que resolva todos os problemas contidos nessa categoria, foi pesquisado técnicas alternativas para a resolução da questão de navegação. Houve foco em uma técnica que generalizasse a movimentação dos robôs em campo e por isso foi estudada a técnica campos potenciais[27] para a resolução desse problema, pois além de possuir um baixo custo computacional era de conhecimento prévio que a mesma geraria bons resultados[28].

#### 4.1.1 Campo Elétrico

A técnica de campos potenciais baseia-se no princípio de campo elétrico, onde a interação de várias cargas elétricas criam um campo de forças que atuam sobre os elementos fazendo com que os elementos que possuam mesma carga independente que sejam ambas positivas ou negativas repelem-se e partículas com cargas opostas atraem-se. Na figura 4.1 encontra-se a



representação do campo elétrico.

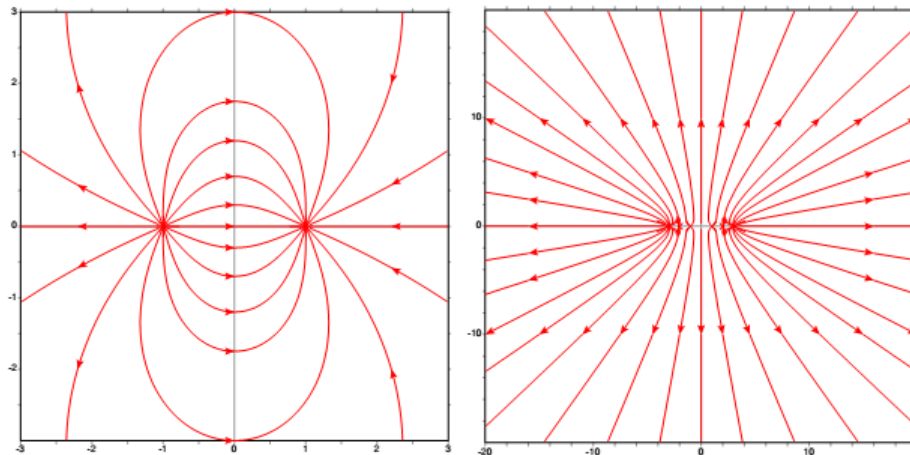


Figura 4.1: Comportamento de um Campo elétrico.

As forças aplicadas às partículas possuem direção, sentido e módulo o que faz com que sejam grandezas vetoriais, e por isso, representa-se o campo potencial como um campo vetorial. Na figura 4.2 encontra-se o campo vetorial que é a representação das forças no campo.

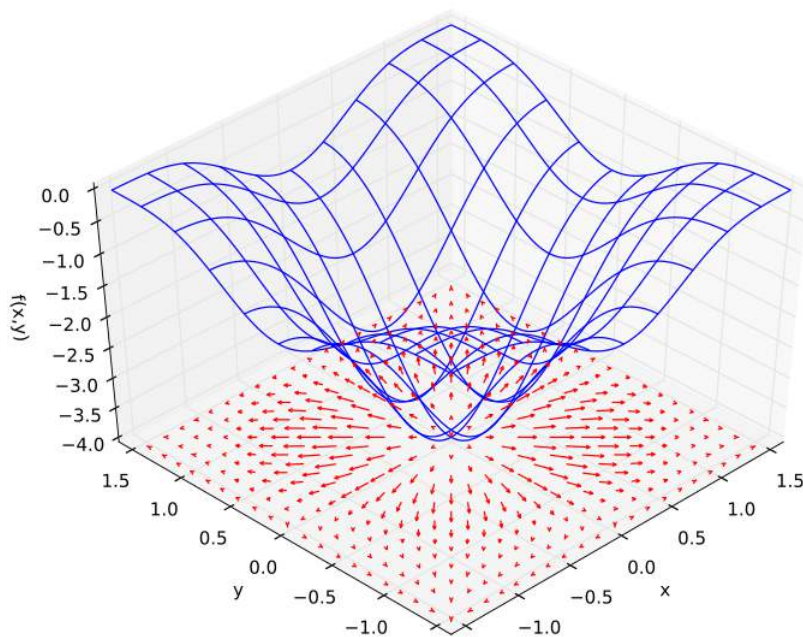


Figura 4.2: Campo vetorial.

A rotina de navegação foi implementada considerando que obstáculos e o robô em questão possuem uma carga igual e as metas geradas possuem uma carga oposta a do robô, assim com o decorrer do tempo o robô direciona-se para sua meta ao mesmo tempo que evita colisões, mais detalhes podem ser encontrados em [29].

## 4.2 Khatib

A ideia de se basear nos campos elétricos foi proposta pela primeira vez por Khatib[30], e sua implementação é dividida em três partes: cálculo do campo repulsivo, cálculo do campo atrativo e cálculo da força resultante.

### 4.2.1 Campo Repulsivo

Para o cálculo do campo repulsivo considera-se uma área mínima de influência ao redor de um obstáculo onde o objeto somente sofrerá força repulsiva estando dentro dessa área. Na maioria das vezes essa área é testada empiricamente e deve ser bem parametrizada para garantir um bom funcionamento do sistema. A força repulsiva ( $\vec{F}_r$ ) é definida como um vetor cujo módulo é inversamente proporcional ao quadrado da distância ( $d$ ) entre o robô e um obstáculo, onde ( $K$ ) é uma constante escolhida, e possui um ângulo resultante igual ao arco tangente da variação negativa entre robô e obstáculo dos eixos X e Y. Na figura 4.3 encontra-se a representação do campo repulsivo. Imagem retirada de [27].

$$|\vec{F}_r| = \frac{K}{d^2} \quad (4.1)$$

$$\theta_{\vec{F}_r} = \arctan\left(\frac{-\Delta y}{-\Delta x}\right) \quad (4.2)$$

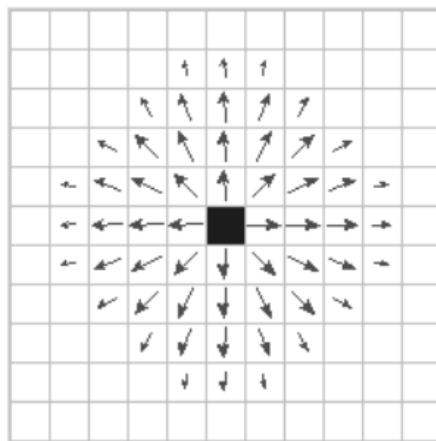


Figura 4.3: Campo vetorial de repulsão ao redor de um obstáculo.

Caso o robô esteja sofrendo repulsão de dois ou mais obstáculos executamos a soma entre os vetores de repulsão e a força repulsiva passa a ser dada pela equação 4.3.

$$\vec{F}_R = \sum_n (\vec{F}_r)_n \quad (4.3)$$

### 4.2.2 Campo Atrativo

A diferença do cálculo do campo atrativo em relação ao campo repulsivo está no fato de que, além da atração gerar influência em todo o campo sem uma área mínima, a força atrativa é constante e a angulação resultante não baseia-se na variação dos eixos X e Y como negativa, mas sim positiva. Na figura 4.4 encontra-se a representação do campo atrativo. Imagem retirada de [27].

$$|\vec{F}_a| = C \quad (4.4)$$

$$\theta_{\vec{F}_a} = \arctan\left(\frac{\Delta y}{\Delta x}\right) \quad (4.5)$$

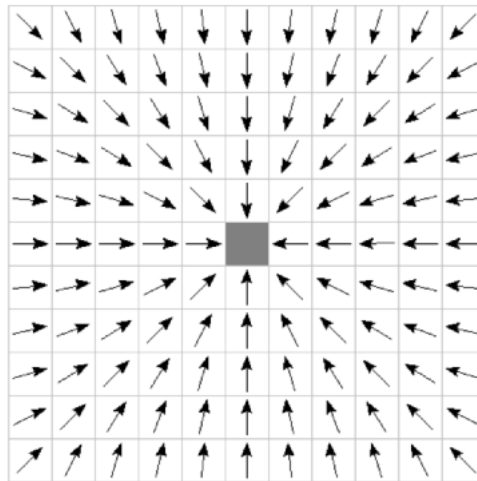


Figura 4.4: Campo vetorial de atração ao redor de uma meta.

### 4.2.3 Força Resultante

Após conhecer-se a força atrativa e a repulsiva, a força resultante é dada pela soma de ambas as forças.

$$\vec{F} = \vec{F}_R + \vec{F}_a \quad (4.6)$$

Essa força resultante dita exatamente para onde o robô deve ir, com base no módulo do vetor resultante há o cálculo a quantidade de força empregada nas rodas e com base na direção e sentido há o cálculo da diferença de força empregada nas rodas para que o robô possa andar em linha reta, executar curvas fechadas, executar curvas abertas e girar. Na figura 4.5 encontra-se a representação do funcionamento das forças aplicadas nas rodas.

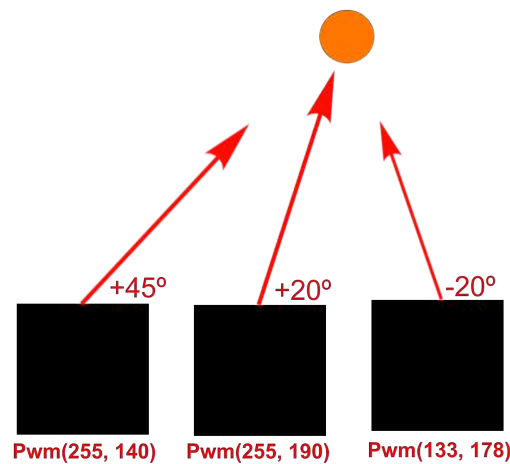


Figura 4.5: Ilustração da força empregada nas rodas esquerda e direita respectivamente.

Apesar dessa técnica funcionar bem e ser de baixo custo computacional ela não é a melhor das opções para se aplicar em nosso problema, pois está sujeita a mínimos locais (áreas intangíveis por característica do método) e não há um controle de velocidade para atingir as metas e parar em uma área desejada, com a observação dessas limitações foi estudado o método de Goodrich[31] que foi utilizado durante a competição.

## 4.3 Goodrich

A diferença entre a implementação sugerida por Khatib e Goodrich é que o modelo de Goodrich separa o campo repulsivo e o campo atrativo em casos e também é resolvido o problema do modelo sugerido por Khatib da força atrativa que passa a ser variável com a distancia dada entre o robô e sua meta.

### 4.3.1 Campo Repulsivo

O campo repulsivo segue a mesma lógica do sugerido por Khatib onde também define-se uma área de influência ( $s$ ) e calcula-se o ângulo do robô em relação ao obstáculo( $\theta$ ), com a diferença de que para a implementação desse modelo é necessário outras informações como

qual o raio ( $r$ ) dos obstáculos e que haja a separação dos possíveis acontecimento em três casos que são referentes a distancia ( $d$ ) do robô para os obstáculos, definisse também uma constante de ganho para a força repulsiva ( $\beta$ ), para um dos casos criados utiliza-se a função signum que nada mais é do que uma função de ativação cuja recebe um valor real de entrada e retorna um valor de saída que é verdadeiro ou falso, o valor de entrada utilizado por essa função é o ângulo ( $\theta$ ) do robô em relação ao obstáculo, caso a função retorne verdadeiro a repulsão aplicada no robô é maior possível. Caso o robô sofra influência de mais de uma força de repulsão com essa implementação também deve-se somar os vetores repulsivos. Na figura 4.6 encontra-se os casos definidos pelo método de Goodrich para o cálculo da força repulsiva. Imagem retirada de [31].

$$\begin{array}{ll} \text{if } d < r & \Delta x = -\text{sign}(\cos(\theta))\infty \text{ and } \Delta y = -\text{sign}(\sin(\theta))\infty \\ \text{if } r \leq d \leq s + r & \Delta x = -\beta(s + r - d) \cos(\theta) \text{ and } \Delta y = -\beta(s + r - d) \sin(\theta) \\ \text{if } d > s + r & \Delta x = \Delta y = 0 \end{array}$$

Figura 4.6: Casos definidos pelo método de Goodrich para força repulsiva.

Os três casos que são verificados são:

- se o robô está encostado no obstáculo, se sim ele recebe um vetor repulsivo de valor máximo programado.
- se o robô está dentro da área de influência, se sim ele recebe um vetor repulsivo de valor inversamente proporcional a distancia vezes um ganho pré-definido.
- se o robô está fora da área de influência, se sim ele não recebe nenhum vetor.

Na figura 4.7 encontra-se a representação do campo repulsivo de Goodrich.

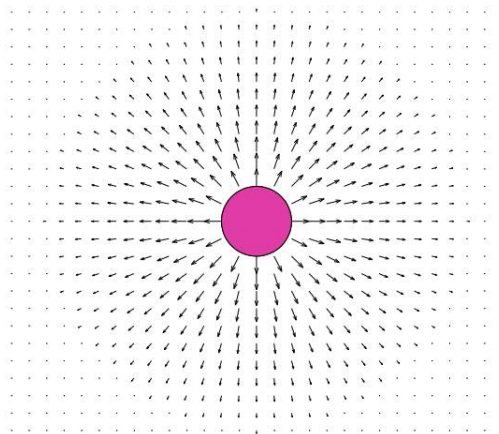


Figura 4.7: O resultado do método de Goodrich para força repulsiva.

### 4.3.2 Campo Atrativo

O campo atrativo segue a mesma divisão do repulsivo, também precisamos saber o raio ( $r$ ) da meta, a distancia ( $d$ ), definir um ganho de atração ( $\alpha$ ) e definir uma área de influência ( $s$ ), a área de influência trabalha junto com o raio da meta, o robô possui uma força de atração continua fora dessas áreas, quando ele entra na área de influência começa a desacelerar e quando entra no raio da meta ele para. Na figura 4.8 encontra-se os casos definidos pelo método de Goodrich para o cálculo da força atrativa. Imagem retirada de [31].

$$\begin{array}{ll} \text{if } d < r & \Delta x = \Delta y = 0 \\ \text{if } r \leq d \leq s + r & \Delta x = \alpha(d - r) \cos(\theta) \text{ and } \Delta y = \alpha(d - r) \sin(\theta) \\ \text{if } d > s + r & \Delta x = \alpha s \cos(\theta) \text{ and } \Delta y = \alpha s \sin(\theta) \end{array}$$

Figura 4.8: Casos definidos pelo método de Goodrich para força atrativa.

Os três casos que são verificados são:

- se o robô está encostado no obstáculo, se sim a força empregada nos motores é 0.
- se o robô está dentro da área de influência, se sim ele começa gradativamente a diminuir sua velocidade.
- se o robô está fora da área de influência, se sim ele trabalha com os valores máximos possíveis

Na figura 4.9 encontra-se a representação do campo atrativo de Goodrich.

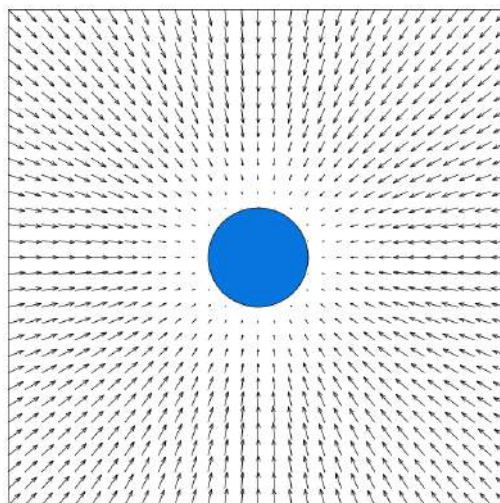


Figura 4.9: O resultado do método de Goodrich para força atrativa.

Na figura 4.10 encontra-se uma representação do comportamento do robô respeitando o método de Goodrich, pode-se notar a trajetória não uniforme ao redor de obstáculo em verde, devido ao efeito gerado pela função *signum* que apenas aplica a força repulsiva caso nosso robô esteja com uma orientação que possivelmente possa levar a uma colisão, pode-se também notar a força de atração reduzindo-se a medida que se entra nas áreas do campo atrativo.

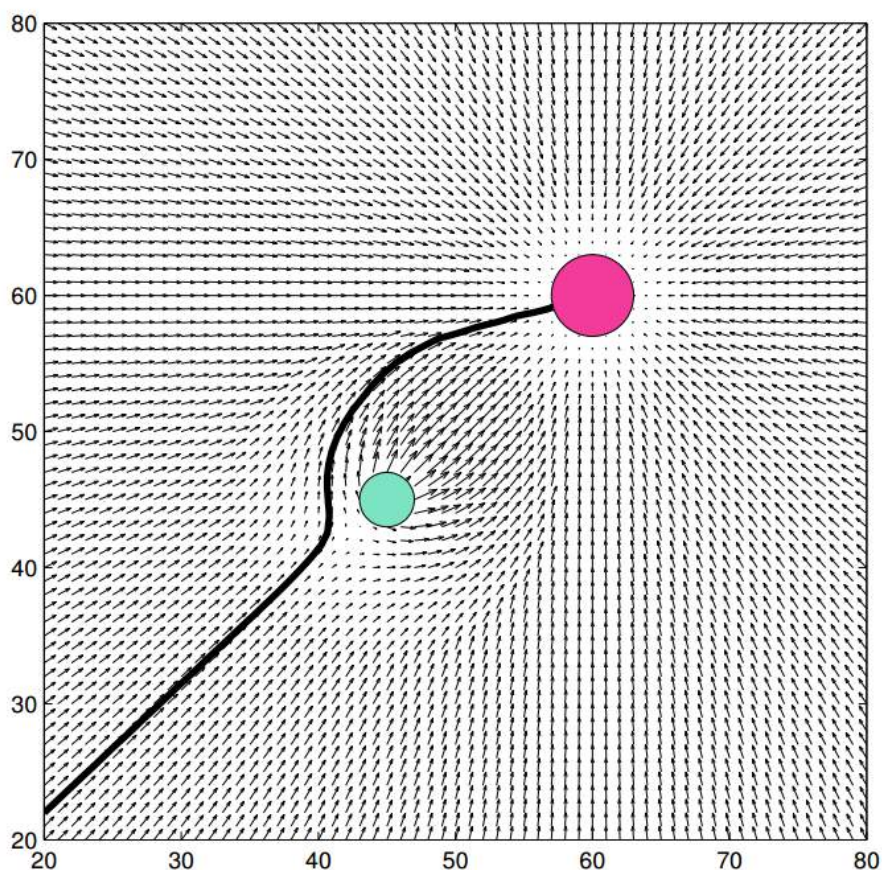


Figura 4.10: Ilustração do resultado dos cálculos empregados no método de Goodrich.

Apesar do método atender nossas expectativas de movimentação, há o empecilho de que não há orientação na movimentação dos robôs em campo, pois a técnica aplicada possui apenas o propósito de levar o robô para uma meta desviando dos obstáculos e fazendo com que o mesmo pare de maneira precisa em um ponto desejado ou chute a bola, porém a técnica não prevê a direção de passagem do robô pelo ponto desejado o que poderia resultar em gol contra, esse problema foi contornado na classe de estratégia projetando um ponto a frente da bola com um descolamento no eixo Y sempre que é detectado perigo de gol contra. Apesar da solução ser funcional não é a ideal, pois o ponto gerado nesse caso possui valores fixos o que nem sempre é o ideal dependendo do ângulo do robô em relação ao gol adversário. Na figura 4.11 encontra-se a parte do código referente a verificação feita.

```

Point Estrategia::projetaJogador(Reta retaTime, int id){
    Point projecao;
    if(perigoGolContra(estado.time[id], estado.bola)){
        projecao.x = projecaoBola.x + 60;

        if( calculaDistancia(estado.time[id], Point(estado.bola.x, estado.bola.y-70))
            < calculaDistancia(estado.time[id], Point(estado.bola.x, estado.bola.y+70)))
            projecao.y = projecaoBola.y-70;
        else
            projecao.y = projecaoBola.y+70;

    }else{
        projecao = destinoJogador(projecaoBola,retaTime);
        //verificaChute();
    }
}

```

Figura 4.11: Situação de gol contra.

Com isso notou-se que a orientação dada ao robô deveria ser imposta também pela a técnica de navegação, pois além de nos dar uma granularidade maior de execuções que são definidas em tempo real, faz com que a tomada do robô nos casos de gol contra seja mais precisa. Uma das possíveis técnicas a serem implementadas para o próximo ano são os Campos Potenciais Localmente Orientados[27], pois ela resolve o problema da orientação apesar de necessitar um custo computacional mais elevado. Na figura 4.12 encontra-se o funcionamento da técnica proposta. Imagem retirada de [27].

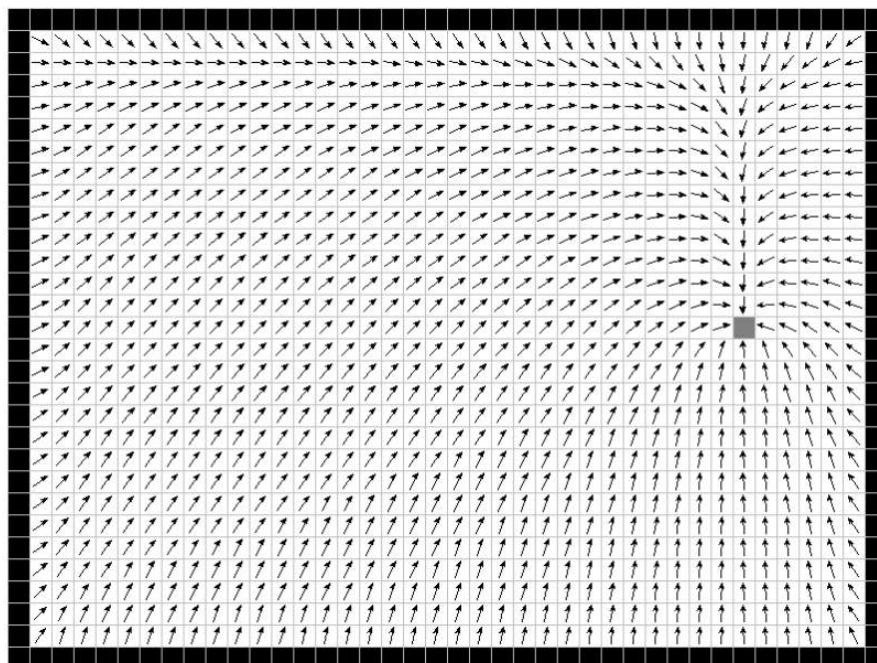


Figura 4.12: Ilustração do funcionamento dos campos localmente orientados.



## 5 Controle

Segundo [32], controle é o ato de comandar, dirigir, ordenar, manipular alguma coisa ou alguém, o controle é responsável por fazer com que os comandos determinados por toda a cadeia de processos sejam executados e corrigidos, imagine o controle de navegação de um carro, onde gostaríamos que ele sempre andasse em uma determinada velocidade, em um terreno plano basta que programe-se uma determinada força exercida nos pedais do carro, porém caso o caminho possua subidas e descidas, o carro irá desacelerar nas subidas e acelerar nas descidas. No caso de determinar-se uma força fixa e essa força atuar durante todo o tempo sem que haja a análise dos acontecimentos e correção estamos lidando com um sistema de malha aberta, esse sistema não está apto a perceber a diferença de velocidade do carro em uma subida e aumentar a pressão dos pedais. Na figura 5.1 encontra-se a representação de um sistema de malha aberta.

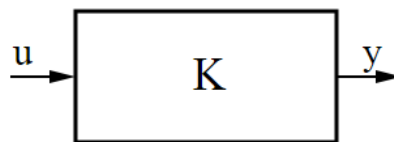


Figura 5.1: Ilustração de um sistema de malha aberta.

Caso desejasse que o sistema compense nas situações de subida ou descida há a necessidade de um sistema de controle de malha fechada que a partir da análise dos acontecimentos no sistema utilizando sensores específicos corrige a execução dos atuadores, assim o sistema estaria apto a acelerar mais em uma subida e desacelerar em uma descida afim de manter um deslocamento constante. Na figura 5.2 encontra-se a representação de um sistema de malha fechada.

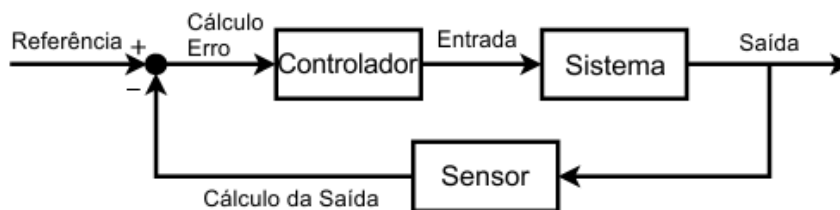


Figura 5.2: Ilustração de um sistema de malha fechada.

Basicamente os sensores encontram um erro referente ao que deveria estar ocorrendo e a partir desse erro calculam uma resposta negativa que é somada ao início da nova interação, assim corrigindo o sistema. Utilizando o mesmo pensamento para o futebol de robôs, imagine que a visão (sensor) detectou em campo um robô e a bola, e a navegação junto a estratégia de jogo determinou uma força a ser exercida em cada roda (atuadores). Caso um sistema de malha aberta seja utilizado, após o cálculo o robô executará aquela ação até encostar na bola, porém caso a bola seja desviada por algum motivo, o robô não irá corrigir a execução, pois não há o sensoriamento do que está acontecendo. Com base nesses problemas utilizamos uma técnica de controle de malha fechada chamado PID[33]. Na figura 5.3 encontra-se uma exemplificação do problema citado.

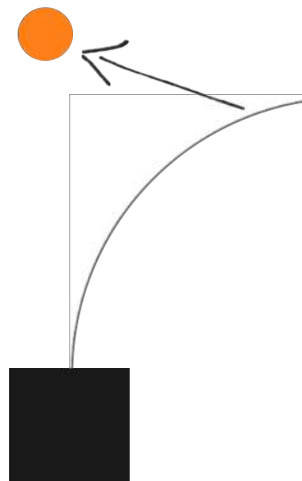


Figura 5.3: Ilustração do comportamento de um sistema em malha aberta.

## 5.1 Controle PID

O controle PID (Proporcional, Integral e Derivativo) faz parte do grupo de técnicas de controle clássicas que trabalha em sistemas de malha fechada e em ciclos contínuos, em geral é o algoritmo de controle mais utilizado na indústria devido ao desempenho robusto em várias condições de funcionamento e ao fato de que sua implementação e sua manutenção ser de fácil execução, segundo Astron[34], entre 90 e 95 por cento dos problemas de controle podem ser solucionados utilizando o PID. O controle PID respeita a seguinte equação:

$$u(t) = K_p e(t) + K_i \int_0^t e(t) dt + K_d \frac{de(t)}{dt} \quad (5.1)$$

Onde ( $u$ ) é o valor a ser executado pelo sistema que é o resultado das somas dos componen-

tes da equação, o único termo comum em ambos os componentes é o erro  $e(t)$  que é mensurado no sistema, pois é a base para o funcionamento do mesmo, esse erro é dado como a diferença entre o *actPoint* e o *setPoint* que são respectivamente o estado onde o sistema encontra-se e o estado que ele deveria estar.

$$e(t) = \text{setPoint} - \text{actPoint} \quad (5.2)$$

Imagine que um robô deveria executar um giro de  $40^\circ$  no próprio eixo, porém por algum motivo o mesmo executou um giro de apenas  $30^\circ$ , nesse exemplo o *actPoint* é  $30^\circ$  e o *setPoint* é  $40^\circ$ , logo o erro é de  $10^\circ$ . Cada termo da equação possui uma constante que dita o peso de cada componente, a constante  $K_p$  dita o peso da componente proporcional,  $K_i$  dita o peso da componente integral e  $K_d$  que dita o peso da componente derivativa. Na figura 5.4 encontra-se a representação do PID.

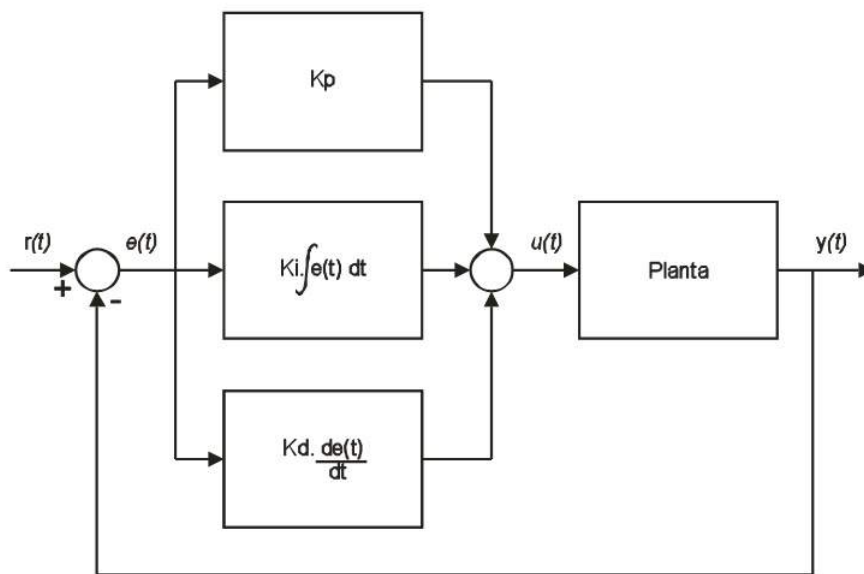


Figura 5.4: Planta de um sistema com o controle PID.

Com o ajuste correto dos pesos das componentes, a medida que o sistema é executado ele é corrigido e a diferença entre o *setPoint* e *actPoint* diminui gradativamente, pois erro é minimizado pela ação proporcional, obtido mais rapidamente pela ação derivativa e zerado pela ação integral.

### 5.1.1 Proporcional

A componente proporcional depende apenas do erro da interação atual do sistema, ele é o erro multiplicado pelo ganho  $K_p$ , em geral o aumento do ganho dessa componente faz com que

a velocidade de resposta do sistema aumente, porém caso o ganho seja muito grande o sistema pode começar a sofrer oscilações.

$$P = K_p e(t) \quad (5.3)$$

### 5.1.2 Integral

A componente integral baseia-se em todo o histórico de erros do sistema, ela é responsável por zerar o erro do sistema, em geral ela serve para compensar as possíveis oscilações geradas no sistema pelo termo proporcional e o aumento do ganho  $K_i$  faz com que o sistema também receba um aumento na velocidade de resposta, porém caso o ganho seja muito grande o sistema estará sujeito ao fenômeno *Overshoot* que acontece quando a resposta passa do valor pretendido. Na figura 5.5 encontra-se um gráfico que demonstra um *overshoot*.

$$I = K_i \int_0^t e(t) dt \quad (5.4)$$

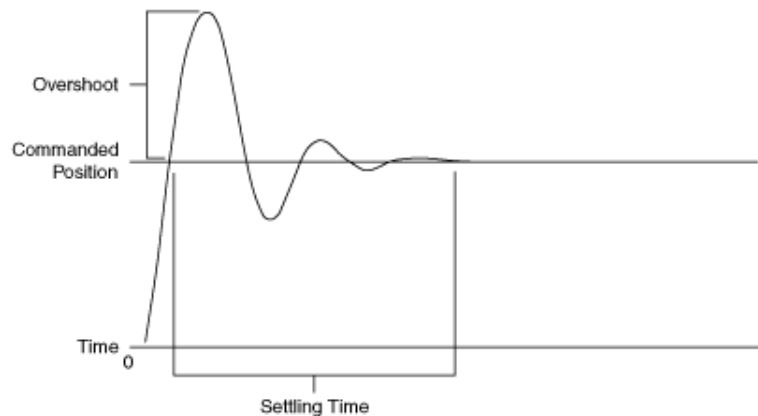


Figura 5.5: Fenômeno do Overshoot.

### 5.1.3 Derivativo

A componente derivativa baseia-se em um intervalo de tempo, geralmente esse intervalo é bem pequeno, pois essa componente é muito sensível a qualquer ruído no sistema, ela é responsável por corrigir o erro antecipadamente, na maioria dos sistemas que utilizam o controle PID essa é a componente que possui o menor ganho  $K_d$ . Na figura 5.6 podemos ver a diferença do tempo de reação que com a técnica que utiliza a componente derivativa é bem menor do que as outras.

$$D = K_d \frac{de(t)}{dt} \quad (5.5)$$

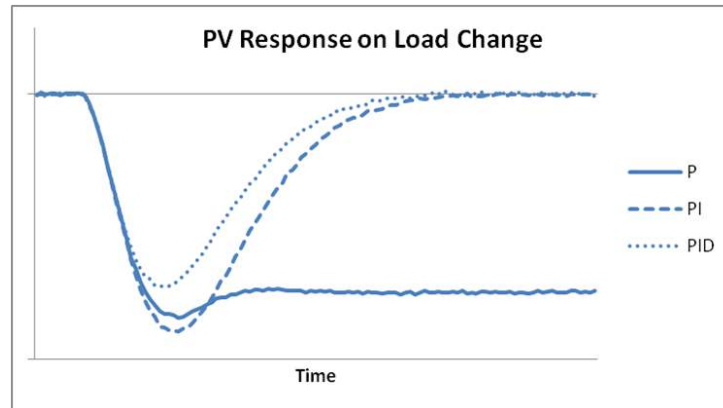


Figura 5.6: Comparação de diferentes técnicas.

## 5.2 Implementação

A implementação do PID no sistema criado considera dois erros, um é a angulação do robô em relação a uma meta ( $e_A$ ) que pode ser a bola ou algum ponto estratégico e outro é a distância do robô para a meta ( $e_D$ ). É feito um cálculo de PID para ambos os erros, foi criado um método que verifica a orientação do robô em relação a uma meta e nesse método é considerado um erro de 0 quando o robô está com a frente para a meta, caso ele esteja rotacionado para a direita ele possui um erro positivo e caso esteja rotacionado para a esquerda, é considerado um erro negativo, com base nesses erros calculamos o  $pid_A$  que serve para ajustar essa angulação. Na figura 5.7 pode-se ver o como o sistema computa os erros durante uma partida.

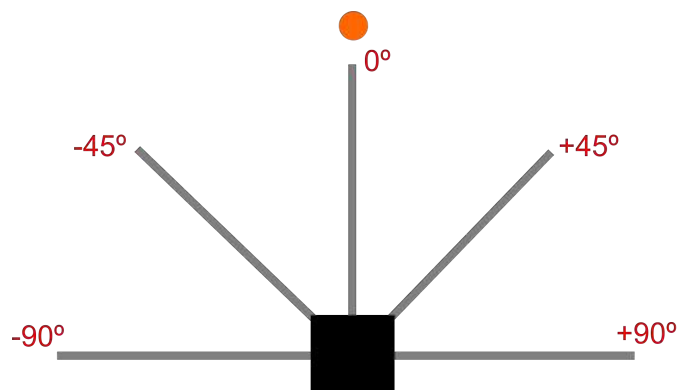


Figura 5.7: Erros considerados no sistema.

A movimentação dos robôs foi dividida em dois casos, caso o robô possua o módulo do erro de ângulo maior do que  $60^\circ$  graus o robô executa um giro, caso contrário o robô executa

uma curva, essa divisão foi feita, pois depois de executados vários testes no laboratório foi verificado que essa configuração nos dava uma movimentação ágil e precisa o suficiente para o jogar. O algoritmo a seguir contempla a implementação do PID referente ao erro de angulação no sistema, onde  $pwmD$  e  $pwmE$ , representam as forças aplicadas nas rodas direita e esquerda respectivamente.

**Algoritmo do  $pid_A$**

```
float pidA = 0;
float erroAtual;
int pwmD, pwmE;
while joga do
    erroAtual = getErroAtual();
    pidA = proporcional() + integral() + derivativo();
    if erroAtual > 60 then
        if pidA > 0 then
            pwmE = pid;
            pwmD = 255 + pid;
        else
            pwmE = 255 + (pid*-1);
            pwmD = pid*-1;
        end
    else
        if pidA > 0 then
            pwmE = 255;
            pwmD = 255 - pid;
        else
            pwmE = 255 - (pid*-1);
            pwmD = 255;
        end
    end
end
```

**Algorithm 1:** Implementação do PID para o ângulo, onde a potência máxima aplicável é de 255.

Após o cálculo do  $pid_A$  deve-se interpretar o valor retornado da técnica que deverá resultar em uma força empregada para as rodas, esse valor pode ser positivo ou negativo, logo o algoritmo foi dividido em casos de giro/curva para esquerda e direita. E o cálculo do  $pid_D$  é feito com base no tamanho do módulo do vetor retornado dos campos potenciais que é o nosso erro considerado, em que a distância máxima da técnica representa os valores sem desacelera-

ção e a medida que esse valor diminui o robô desacelera. O algoritmo a seguir contempla a implementação do PID referente ao erro de distância no sistema.

#### Algoritmo do $pid_D$

```
float pidD = 0, pidDmax, gain;
int pwmD, pwmE;
while joga do
    //Calculo dos valores de PWM pelo pidA
    pidD = proporcional() + integral() + derivativo();
    pwmE *= gain*pidD/pidDmax;
    pwmD *= gain*pidD/pidDmax;
end
```

**Algorithm 2:** Implementação do PID para a distância.

## 5.3 Parametrização

A parametrização do PID é o maior problema enfrentado com o uso dessa técnica, pois pelo fato de sermos obrigados a trabalhar com três valores que podem variar entre si e suas variações afetarem o sistema de maneira diferente, torna-se extremamente trabalhoso a parametrização correta do sistema. Na figura 5.8 pode-se ver o comportamento da técnica com diferentes pesos determinados em um sistema qualquer.

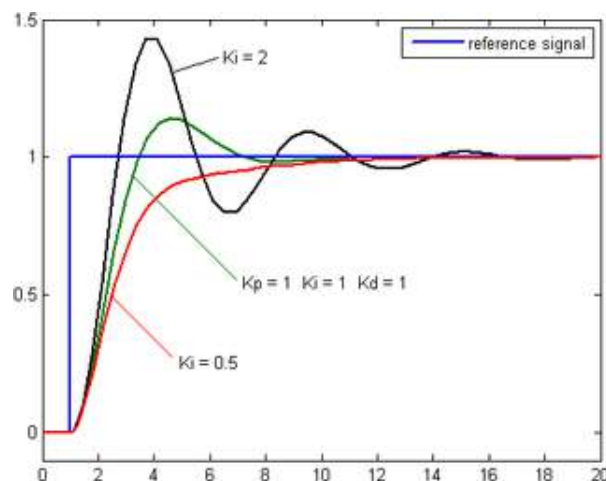


Figura 5.8: Respostas do sistema a diferentes parâmetros.

Podemos testar valores empiricamente no sistema até encontrar os valores ideais ou podemos utilizar alguma técnica de parametrização, em um primeiro momento foi testado valores manualmente, porém notou-se uma grande dificuldade em interpretar o funcionamento da

técnica, por isso foi estudado outros métodos de parametrização e um dos que se mostraram eficientes para nosso problema foi o método de Ziegler-Nichols [35].

### 5.3.1 Ziegler-Nichols

O método de Ziegler-Nichols é usado para parametrizar qualquer técnica que possua a ação proporcional como, por exemplo, controle P, PI, PD e PID. O método segue os seguintes passos:

- Determinar  $K_p$  crítico, considerando apenas o ganho proporcional.
- Obter o máximo de 25 por cento de *overshoot* com a ação proporcional.
- Determinar a frequência de oscilação  $F_o$ .
- Determinar o período crítico  $P_c$ .

Após determinado todos os parâmetros no método utilizamos a seguinte tabela para definir  $K_i$  e  $K_d$ .

Tabela 5.1: Tabela Ziegler-Nichols

Tipo	$K_p$	$K_i$	$K_d$
P	$0.50P_c$	-	-
PI	$0.45P_c$	$1.2K_p/F_o$	-
PD	$0.80P_c$	-	$K_pF_o/8$
PID	$0.60P_c$	$2K_p/F_o$	$K_pF_o/8$

Essa técnica é usada, pois simplifica o trabalho de parametrização do PID, em geral quando se faz todos os passos da técnica o resultado final é um controle com pouco ou nenhum *overshoot* e uma estabilização do erro rápida.



## 6 Robôs

Segundo as regras da competição [8], os robôs criados para essa categoria não podem possuir mais do que 7,5cm de lado com uma tolerância de 8cm para suas capas, essa limitação é o principal fator que influencia na construção dos robôs, pois o pouco espaço livre disponível faz com que seja um desafio construí-los. Na figura 6.1 pode-se ver os robôs criados para a competição que respeitam os limites de tamanho.

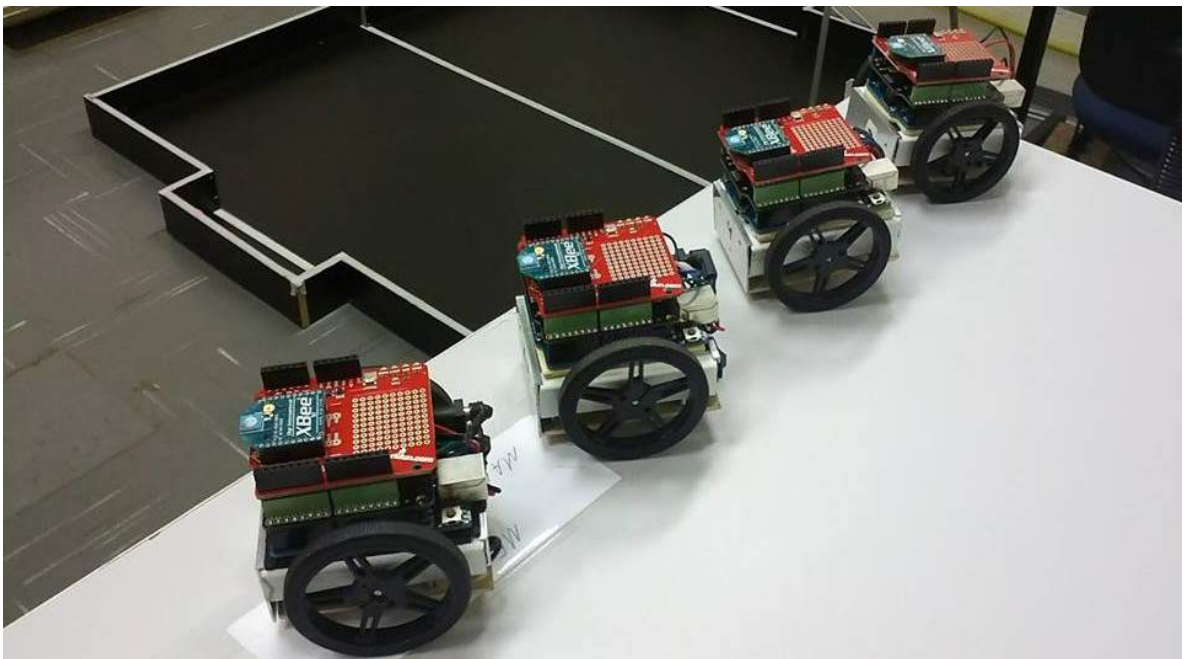


Figura 6.1: Robôs criados para a categoria VSS.

### 6.1 Construção

Para a construção de cada robô foram utilizados:

- 1 Arduino Uno
- 1 Controlador de Motor DRFduino L298P

- 1 Shield XBee
- 1 Modem XBee
- 2 Micromotores Pololu
- 2 Rodas
- 1 Bateria de 2A

### 6.1.1 Arduino

Foi utilizado a plataforma de prototipagem Arduino[36] UNO que possui o microcontrolador Atmega 328P, o microcontrolador que possui um clock de 16Mhz e trabalha a 5V possui 14 pinos digitais onde 6 desses pinos podem ser utilizados como PWM (Modulação por largura de pulso), conexão USB, suplemento alimentar completo que possibilita o uso de baterias, suporte ao protocolo de comunicação ICSP (*In-circuit serial programming*) que possibilita programar o Arduino no circuito e um botão de reset. Na figura 6.2 encontra-se a plataforma Arduino.

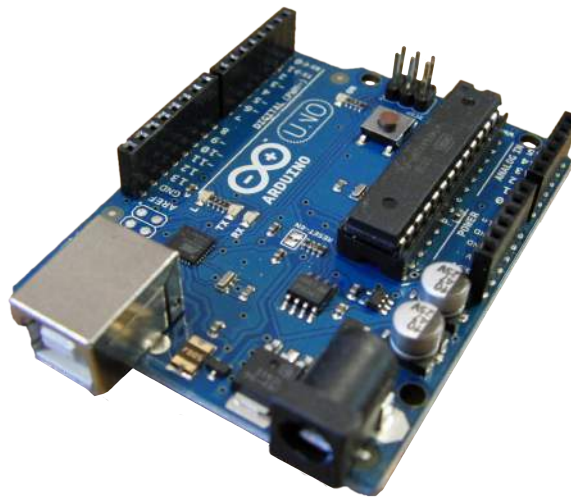


Figura 6.2: Arduino Uno.

O Arduino possui um *software* próprio para que o mesmo possa ser programado, ele utiliza uma linguagem de programação própria que possui muitos aspectos parecidos com C, a plataforma conta com inúmeros códigos prontos que podem ser utilizados como comunicação serial, controle de motor e uma interface pronta que simplifica a implementação do PWM. Esse microcontrolador é responsável por fazer uma interface entre o controlador de motor e o microtransmissor.

## 6.1.2 Controlador de Motor

Foi utilizado o controlador de motor DRFduino L298P, pois o mesmo possui compatibilidade com a plataforma Arduino simplificando o trabalho de fazer com que ambos trabalhem juntos, o controlador que trabalha a 5V possibilita que dois motores de 7 a 12V sejam operados com corrente elétrica de 2A, por padrão esses motores atuam nos pinos 5 e 6 e pinos 4 e 7 do Arduino. Na figura 6.3 encontra-se o shield do controlador de motor utilizado.

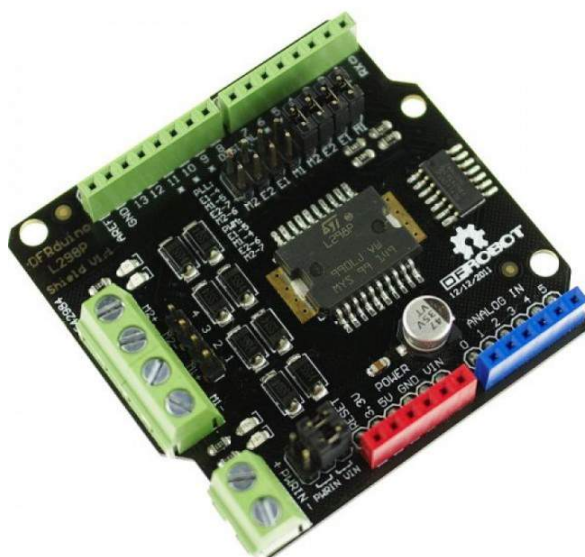


Figura 6.3: Controlador usado.

O principal motivo de utilizar-se um controlador de motor é o fato dele possibilitar a inversão do giro do motor, que junto com a técnica PWM aumenta muito o número de possibilidades de execução do motor. O PWM[37] é utilizado para determinar uma força a ser executada pelo motor, pelo fato de trabalhar-se com sinais digitais apenas é possível utilizar os valores 0 e 1, que no nosso caso representa respectivamente o motor desligado e o motor ligado com a maior potência que o mesmo pode gerar. Para entender-se o PWM imagine um jogo de corrida, onde o jogador possui um controle com dois botões, um para virar o carro para esquerda e outro para direita, caso nenhum dos botões sejam apertados o carro segue em linha reta, diferente da vida real onde o piloto controla o carro com o volante o que lhe dá uma infinidade de execuções, no jogo possui-se apenas 3 possibilidades, o princípio de PWM é o que faz com que seja possível o jogador executar uma infinidade de curvas durante o jogo, analisando as ações do jogador para executar uma curva, ele alterna entre o botão pressionado e não pressionado visualizando a distância do carro para as beiras, isso acontece, pois o jogador não tem a possibilidade de virar o carro com diferentes intensidades e utiliza o tempo para simular o que na vida real um piloto apenas giraria o volante 42 graus, analogamente o valor 0 do motor representa o controle sem nenhum

botão pressionado, o 1 representa o botão pressionado e o valor de 42 graus é a voltagem que gostaria-se de empregar ao motor para gerar diferentes potências. A medida que diminui-se o intervalo de tempo para alternar-se entre ligado e desligado a técnica torna-se mais precisa em suas representações dos valores intermediários. Na figura 6.4, a imagem acima demonstra um sinal analógico que gostaria-se de seguir e a imagem abaixo demonstra a variação do sinal digital com base no tempo para simular o sinal analógico.

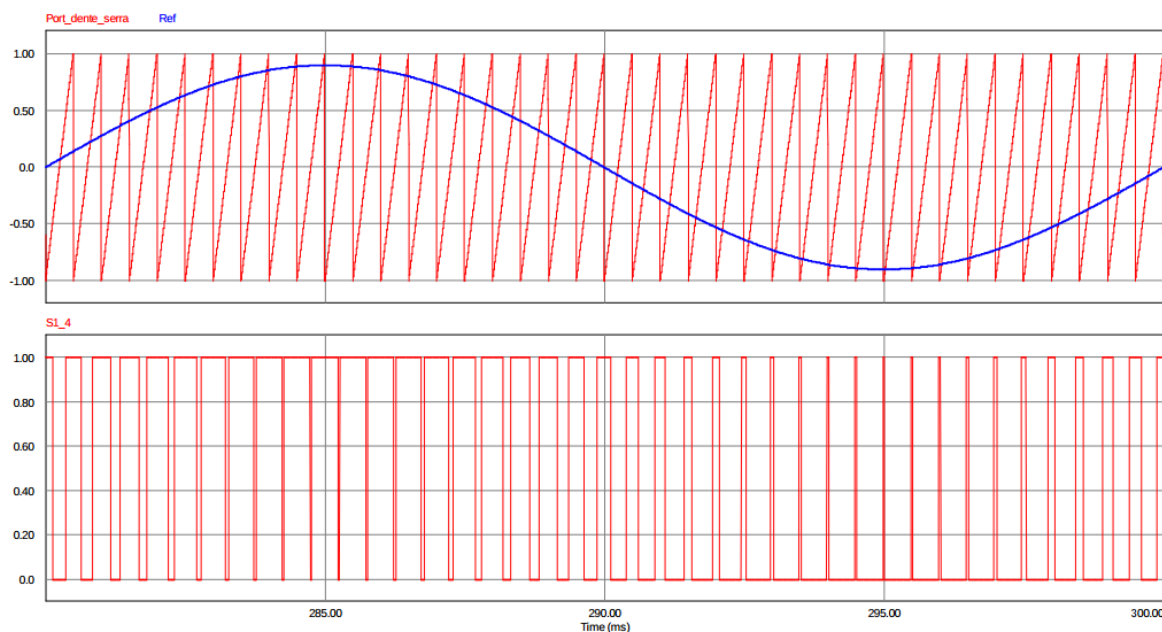


Figura 6.4: Pwm.

### 6.1.3 XBee

XBee é um modem de rádio frequência que é utilizado no mundo todo em sistemas embarcados de tecnologia *wireless*, existem diversos modelos dessa categoria de transmissores com diferentes consumos de energia e frequências de transmissão. O XBee utilizado para a comunicação foi o XBee-PRO DigiMesh 2.4, que opera a 2.4Ghz e transmite dados para até 3200m, apesar do modelo utilizado possibilitar a construção de diversas topologias de redes e criptografiação dos dados enviados, utilizou-se uma transmissão broadcast e os dados foram transmitidos em textos planos, o que foi um erro, pois durante a competição notou-se uma interferência na transmissão dos dados do servidor para os robôs que afetou de maneira negativa o sistema criado, para a próxima competição deve-se criptografar os comandos transmitidos. O módulo XBee foi utilizado tanto para a transmissão do computador quanto para a recepção dos dados no robô. Para utilizar-se o XBee como transmissor é necessário um *explorer* que faz o trabalho de transmitir os dados via USB para os Robôs. Na figura 6.5 encontra-se o *explorer* utilizado.



Figura 6.5: Explorer utilizado para a transmissão.

Para a recepção dos dados no robô é utilizado um *shield* que é responsável por tratar a comunicação do XBee com o Arduino, o principal motivo desse tratamento deve-se ao fato de o Arduino trabalhar a 5V e o XBee a 3,3V, sem esse tratamento o Arduino ignoraria os dados vindos do XBee. Na figura 6.6 encontra-se o *shield* plugado a um Arduino.

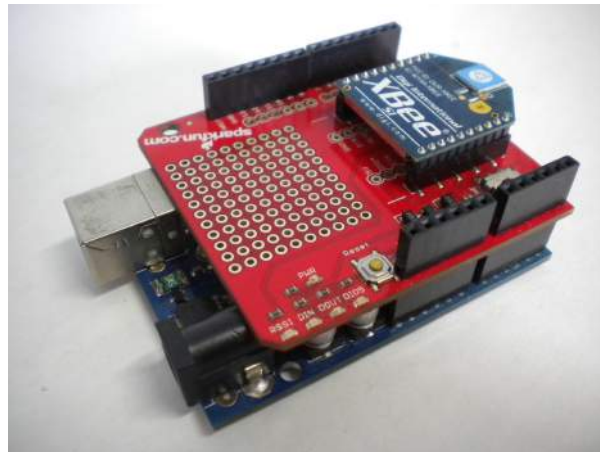


Figura 6.6: Shield utilizado para a comunicação entre o XBee e o Arduino.

A transmissão de dados é feita a uma frequência de 9600 bps onde há a transmissão de uma mensagem broadcast de tamanho variável entre 48 e 54 bits que são divididos e transmitidos de 8 em 8 bits, essa mensagem guarda os valores de PWM dos motores de todos os robôs. Na figura 6.7 encontra-se a exemplificação de uma mensagem transmitida pelo XBee.

Robô 0		Robô 1		Robô 2	
255	255	510	510	100	100

Robô 0		Robô 1		Robô 2	
11111111	11111111	11111110	11111110	1100100	1100100

Figura 6.7: Mensagem transmitida para os robôs.

No caso apresentado na figura 6.7 a mensagem seria transmitida como um único *long long int* com o valor de 255255510510100100. A recepção é feita de modo serial, ou seja, a medida que os dados vão sendo recebidos pelo XBee eles vão sendo alocados em uma fila, na rotina programada dos robôs há um loop de execução que verifica o numero de elementos na entrada serial do Arduino com auxílio da biblioteca padrão `SoftwareSerial`, quando o tamanho da mensagem guardada na fila chega a um determinado valor o algoritmo de recepção entra na rotina de ler os dados da entrada e executar o comando recebido. O método `read()` da biblioteca utilizada por padrão lê inteiros de 8 bits, que era iniciado sempre que existisse pelo menos 6 inteiros de 8 bits na fila do Arduino, quando essa condição era satisfeita o Arduino lia a entrada serial 6 vezes a procura das mensagens referentes aos robôs, apesar de ser funcional esse acesso a entrada serial do Arduino é lento e por isso surgiu um *delay* de execução por parte dos robôs. A solução para o problema de execução foi escrever um novo método de leitura que trabalharia junto à biblioteca `SoftwareSerial`, esse novo método diferente do método padrão que apenas lia 8 bits conseguia ler 64 bits, o que da a possibilidade que com apenas uma lida da entrada serial todos os robôs possuam seus comandos à serem executados. Mais informações sobre a transmissão em [APÊNDICE B - Núcleo da Rotina de Transmissão] figuras 10.2 e 10.1 .

#### 6.1.4 Outros componentes

Foram utilizados motores Pololus de corrente continua de alta potência que trabalham a 6V e possuem caixa de redução com uma relação 1:75, essa configuração consegue entregar um torque de 1.58 kg-cm. As rodas utilizadas possuíam 6 cm de diametro e peso 11 gramas e a bateria utilizada 2000mA. Na figura 6.8 encontra-se o motor e rodas utilizadas durante a competição.



Figura 6.8: Motor e rodas utilizadas.

## 7 Resultados

Ao todo 20 equipes foram aceitas na *Latin American Robotics Competition* (LARC) na categoria *IEEE Very Small Size*, porém apenas 17 compareceram, afetando na divisão dos grupos que foram sorteados. Foram criados 3 grupos com 4 equipes cada e 1 grupo com 5 equipes. Os jogos ocorreram dos dias 20 a 22 de Outubro de 2014, dentre as equipes participantes a equipe SirSoccer conseguiu conquistar o 4º lugar da LARC 2014.

### 7.1 Equipes participantes

Nas tabelas 7.1, 7.2, 7.3 e 7.4 encontram-se os grupos criados para a competição.

Tabela 7.1: Grupo A

Equipe	Localidade	Universidade
Ararabots Azul	Brazil/MS	UFMS
ERUS	Brazil/ES	UFES
Pequi Mecânico	Brazil/GO	UFG
Warthogs	Brazil/SP	USP

Tabela 7.2: Grupo B

Equipe	Localidade	Universidade
Autobotz	Brazil/MG	UFMG
Red Dragons	Brazil/SP	UFSCar
Robok B	Brazil/MG	UNIFEI
SirSoccer	Brazil/RJ	FAETERJ

Tabela 7.3: Grupo C

Equipe	Localidade	Universidade
Drumsters	Brazil/MG	UNIFEI
Hefestos	Brazil/RJ	UFRJ
GER	Brazil/SP	—
Rodetas	Brazil/MG	UFOP
Itandroids	Brazil/SP	ITA

Tabela 7.4: Grupo D

Equipe	Localidade	Universidade
Carrossel Caipira	Brazil/SP	UNESP
Nerds da Fronteira	Brazil/MS	UFMS
Robok A	Brazil/MG	UNIFEI
Umayux	Peru	—

## 7.2 Fase de grupos

Durante a fase de grupos a equipe SirSoccer jogou contra as equipes Autobotz (UFMG) e Red Dragons (UFSCar), vencendo ambas partidas e garantindo o lugar na fase eliminatória.

Resultados da fase de grupos:

Tabela 7.5: Resultados grupo A

Data e Horário					
20/10 - 10:30	Warthogs	0	x	1	ERUS
20/10 - 14:30	Ararabots	0	x	2	Pequi Mecânico
20/10 - 16:30	Ararabots	0	x	2	ERUS
21/10 - 9:30	Warthogs	9	x	1	Pequi Mecânico

Tabela 7.6: Resultados grupo B

Data e Horário					
20/10 - 9:30	SirSoccer	4	x	0	Autobotz
20/10 - 11:30	Robok B	4	x	2	Red Dragons
20/10 - 15:30	Robok B	0	x	0	Autobotz
21/10 - 10:30	SirSoccer	3	x	1	Red Dragons

Tabela 7.7: Resultados grupo C

Data e Horário					
20/10 - 9:00	Itandroids	0	x	2	Rodetas
20/10 - 11:00	GER	0	x	5	Drumonsters
20/10 - 14:00	Hefestos	1	x	4	Rodetas
20/10 - 16:00	Itandroids	0	x	10	Drumonsters
21/10 - 10:00	Hefestos	0	x	0	GER

Tabela 7.8: Resultados grupo D

Data e Horário					
20/10 - 10:00	Robok A	1	x	0	Nerds
20/10 - 15:00	Carrossel	0	x	0	Umayux
20/10 - 17:00	Carrossel	0	x	1	Nerds
21/10 - 9:00	Robok A	3	x	0	Umayux



## 7.3 Fase Eliminatória

Na fase eliminatória a equipe SirSoccer venceu a até então campeã Latino Americana da categoria VSS, a equipe Rodetas (UFOP) nas quartas de finais, perdeu para a equipe Warthogs (USP) na semi-final e perdeu para a equipe ERUS (UFES) na disputa pelo terceiro lugar. O resultado a partir da semifinal foi muito a baixo da média, pois os robôs não executavam as ações comandadas pelo servidor, mais tarde foi constatado que houve uma interferência no sinal recebido pelos mesmos. Na figura 7.1 encontra-se a chave de torneio da fase eliminatória.



Figura 7.1: Chave de torneio.

Tabela 7.9: Resultados quartas de final

Data e Horário				
21/10 - 14:00	Robok A	1	x	6 Warthogs
21/10 - 14:30	SirSoccer	6	x	3 Rodetas
21/10 - 15:00	ERUS	3	x	1 Nerds
21/10 - 15:30	Drumonsters	3	x	0 Robok B

Tabela 7.10: Resultados semifinais

Data e Horário					
21/10 - 17:00	SirSoccer	1	x	11	Warthogs
21/10 - 18:00	ERUS	1	x	6	Drummonsters

Tabela 7.11: Resultado final

Data e Horário					
22/10 - 11:00	Warthogs	0	x	2	Drummonsters

Tabela 7.12: Resultado disputa terceiro lugar

Data e Horário					
22/10 - 9:00	SirSoccer	1	x	2	ERUS

## 8 *Conclusões e Trabalhos Futuros*

Analisando os resultados obtidos durante a LARC 2014, fica evidente que a participação da equipe SirSoccer na categoria *IEEE Very Small Size* foi bem sucedida. Comparando o trabalho feito pela equipe do SIR Lab (FAETERJ) durante o ano de 2014 com outras equipes de grandes universidades brasileiras e latino americanas de renome, pode-se ver um avanço significativo da plataforma criada, plataforma essa que conquistou o quarto lugar latino americano em seu segundo ano de competição.

Apesar dos resultados terem sido satisfatórios, durante a competição notou-se diversos problemas que devem ser resolvidos para se otimizar a plataforma e torna-la uma das melhores da América Latina, tanto em aspecto de velocidade de execução quanto em aspecto de competência do grupo de robôs de resolver situações de jogo, dentre esses problemas pode-se citar, interferência da comunicação e controle empregado nos robôs em campo. Existem aspectos que podem ser melhorados como a estratégia de jogo, calibragem do sistema de visão computacional e navegação dos robôs em campo. Também existem componentes de *hardware* que podem ser incorporados na plataforma que gerariam um ganho para o sistema, como a adoção de *encoders* que possibilitaria o trabalho com uma modelagem matemática do problema do VSS, curva na carcaça do robô que possibilitaria carregar a bola e uma câmera profissional que possua baixo tempo de resposta, alta taxa de fps e maior resolução que possibilitaria o sensoriamento mais preciso dos robôs em movimento.

Em geral há sempre o que se aprimorar em um sistema robótico, uma evidência disso é a disparidade dentre as equipes que participam das CBRs e LARCs para as que competem no mundial MiroSot, disparidade essa que deve ser encarada como motivação para que sempre busque-se aprimoramento, por sua vez essa busca por aprimoramento acarreta no desenvolvimento profissional e acadêmico dos alunos e professores envolvidos no projeto.

## 9 APÊNDICE A - Núcleo da Rotina de Estratégia

Um dos problemas decorrentes durante uma partida são travamentos dos robôs, que podem ocorrer por diversos fatores, para todo caso foi criado um método que verifica se um robô está parado em um mesmo ponto por mais de 1 segundo, como pode ser visto no código apresentado na figura 9.1. Caso o robô esteja parado ele entra no procedimento de saída, que basicamente é tratar todas transmissões dele como ré durante 17 iterações do algoritmo que representam aproximadamente 0,6 segundos.

```

209     if(procedimentoSaida < 0){
210         comando = pid.comando(distanciaObj, angulo, papel, chuta, distancia, e.bola.y);
211
212         if(distanciaObj > 10){
213             frames++;
214         }
215         if(frames > 30){
216             if(papel != goleiro){
217                 if(calculaDistancia(robo, fantasma) < 10 && distanciaObj > 10){
218                     procedimentoSaida = 0;
219                     setCmdSaida(casoDeMinimo(e.time[id]));
220                 }
221             }else{
222                 if(calculaDistancia(robo, fantasma) < 10 && distanciaObj > 10){
223                     //procedimentoSaida = 0;
224                     setCmdSaida(casoDeMinimo(e.time[id]));
225                 }
226             }
227
228             fantasma = robo;
229             frames = 0;
230         }
231     }else{
232         status = "Saida";
233         if(procedimentoSaida < 13){
234             procedimentoSaida++;
235             comando = "510510";
236         }else
237         if(procedimentoSaida < 17){
238             procedimentoSaida++;
239             comando = cmdSaida;
240         }else{
241             procedimentoSaida = -1;
242         }

```

Figura 9.1: Controle contra travamentos dos robôs em campo.

A estratégia foi construída pensando em um time de futebol de verdade, onde existem diversos papéis a serem executados em campo, em geral na estratégia possuíse três papéis definidos: Atacante, Zagueiro e Goleiro. O tipo de atuação de cada robô depende do seu papel, por exemplo, se o papel atual é goleiro ele tende a ficar próximo a área do gol, se o papel é zagueiro ele tende a ficar mais a frente e se é um atacante ele marca a bola o tempo todo, a vantagem desse pensamento é que se a cada rodada for verificado as condições de jogo é possível fazer com que os robôs troquem de posição caso vantajoso.

Na figura 9.2 encontra-se a representação da organização criada para os jogadores.

```
253 Estrategia::Estrategia(){
254     sir[0].setPapel(goleiro);   sir[0].setId(0);
255     sir[1].setPapel(zagueiro);  sir[1].setId(1);
256     sir[2].setPapel(atacante);  sir[2].setId(2);
257     frames = 1;
258     campoAtaque = false;
259     tempoDeTroca = 0;
260     permiteTroca = true;
261 }
```

Figura 9.2: Representação dos jogadores em campo.

Outra situação que devem ser verificada durante a partida é se há chance das ações de um robô resultar em gol contra, por isso há a verificação do caso, que trabalha junto com o código referente a figura 4.11 do capítulo 4, vale atentar também que a bola está em constante movimento e por isso não faz sentido o robô ir no ponto que a bola está naquele momento, pois ele sempre irá chegar atrasado, por isso calculasse uma projeção da bola e o robô direcionasse para onde a bola irá estar. Na figura 9.3 encontra-se o cálculo da projeção da bola e a verificação dos casos de gol contra.

```
60 bool perigoGolContra(Robo a, Point b){
61     if(a.x < b.x)   return true;
62     else            return false;
63 }
64
65 Point calculaProjecao(Point a, Point b){
66     return Point(((a.x - b.x) + a.x), ((a.y - b.y) + a.y));
67 }
```

Figura 9.3: Cálculo da projeção da bola e verificação de gol contra.

Uma estratégia empregada na retomada de bola é traçar uma reta entre a bola e o centro do gol criando uma projeção, assim quando um robô estiver apto a retomar a bola, ele a carrega para o gol adversário com uma rota já traçada. Na figura 9.4 encontra-se as funções utilizadas para fazer com que seja possível calcular o ponto de projeção.

```
69  Reto equacaoReta(Point a, Point b){
70      float inclinacao = ((float)a.y - (float)b.y)/((float)a.x - (float)b.x);
71      int n = a.y - inclinacao*a.x;
72
73      //cout << endl << endl << "INCLINAÇÃO: " << inclinacao << endl;// << "a.;"
74
75      Reto reta;
76      reta.m = inclinacao;
77      reta.n = n;
78
79      return reta;
80  }
81
82  Reto calculaRetaGol(Point bola, bool goleiro){
83      Point gol;
84
85      gol.y = 240;
86
87      if(!goleiro){
88          gol.x = 0;
89          return equacaoReta(bola,gol);
90      }else{
91          gol.x = 640;
92          return equacaoReta(gol,bola);
93      }
94  }
```

Figura 9.4: Cálculo da reta para o gol.

## 10 APÊNDICE B - Núcleo da Rotina de Transmissão

Como foi descrito no capítulo 6, a baixa capacidade de processamento do Arduino faz com que o tempo gasto para ler a entrada serial seja bastante elevado, pelo fato da mensagem ser transmitida de forma serial o Arduino é obrigado a ler toda a fila de mensagens, isto é, o Robô 1 que necessita das duas primeira mensagens enviadas pela central, deveria ler a entrada serial 2 vezes, já o Robô 2 que necessita das mensagens 3 e 4, é obrigado a ler a entrada serial 4 vezes, assim temos que.

$$T = 2i\alpha \quad (10.1)$$

Onde ( $T$ ) é o tempo total levado para a execução do comando, ( $i$ ) é o id do robô e ( $\alpha$ ) é o tempo que o Arduino leva para a leitura da entrada serial.

Na figura 10.1 encontra-se o código fonte do método reescrito na biblioteca SoftwareSerial.

```

423 int SoftwareSerial::read()
424 {
425     if (!isListening())
426         return -1;
427
428     // Empty buffer?
429     if (_receive_buffer_head == _receive_buffer_tail)
430         return -1;
431
432     // Read from "head"
433     uint8_t d = _receive_buffer[_receive_buffer_head]; // grab next byte
434     _receive_buffer_head = (_receive_buffer_head + 4) % _SS_MAX_RX_BUFF;
435     return d;
436 }
437
438 // SIR Read data from buffer
439 uint64_t SoftwareSerial::SIRread()
440 {
441     uint64_t d = *((uint64_t*)_receive_buffer);
442     _receive_buffer_head = (_receive_buffer_head + 8) % _SS_MAX_RX_BUFF;
443     return d;

```

Figura 10.1: Método reescrito na biblioteca padrão do Arduino SoftwareSerial

Visando a resolução desse problema foi criado um novo método dentro da biblioteca SoftwareSerial do Arduino com capacidade de ler todo o comando de uma única vez o `SIRread()`, o método lê 64 bits de uma única só vez diferente do método padrão que lê apenas 8 bits, isso faz com que o tempo que os robôs demoram para ler a mensagem sejam o mesmo.

Na figura 10.2 encontra-se parte do código fonte do robô voltado para a recepção dos dados enviados pela central de processamento.

```

26 bool leEntradaSerial(struct Comando *cmd){
27     bool executa = false;
28     uint64_t leitura;
29     //255255255255255255
30     if(Xbee.available() > 7){
31         leitura = Xbee.SIRread();
32
33         #if(ROBO == 0)
34             int resto1 = (leitura/1000000000000000) % 1000;
35             int resto2 = (leitura/10000000000000) % 1000;
36         #endif
37
38         #if(ROBO == 1)
39             int resto1 = (leitura/1000000000) % 1000;
40             int resto2 = (leitura/10000000) % 1000;
41         #endif
42
43         #if(ROBO == 2)
44             int resto1 = (leitura/1000) % 1000;
45             int resto2 = leitura % 1000;
46         #endif
47
48         if(resto1 < 256){
49             Serial.println("Roda Esquerda-> Frente");
50             Serial.println(resto1, DEC);
51             cmd->direcaoE = 1;
52             cmd->pwmE = resto1;
53         }else{
54             Serial.println("Roda Esquerda -> Tras");
55             Serial.println(resto1 - 255, DEC);
56             cmd->direcaoE = 0;
57             cmd->pwmE = resto1 - 255;
58         }
59
60         if(resto2 < 256){
61             Serial.println("Roda Direita -> Frente");
62             Serial.println(resto2, DEC);
63             cmd->direcaoD = 1;
64             cmd->pwmD = resto2;
65         }else{
66             Serial.println("Roda Direita -> Tras");
67             Serial.println(resto2 - 255, DEC);
68             cmd->direcaoD = 0;
69             cmd->pwmD = resto2 - 255;
70         }
71
72         if(resto1 != 666 && resto2 != 666) executa = true;
73         Xbee.flush();
74     }
75 }

```

Figura 10.2: Código referente a recepção dos dados pelo Arduino.



Com o problema da recepção resolvido, restava somente o problema de fazer com que cada robô extraísse a informação referente ao mesmo, o que acontecia naturalmente pelo método padrão da biblioteca `SoftwareSerial Read()`, para isso o algoritmo foi adaptado, pois pelo fato da transmissão se dar com um único inteiro de 64 bits, temos números do tipo 255255255255255255 ou 100100100100100100, como cada robô necessita de parte desse inteiro foi criado uma divisão onde dependendo da ID do robô ele executa diferentes operações de divisão e de resto para extrair sua parte da mensagem, como pode ser visto na figura 10.2. Outro problema seria definir a direção para onde os motores deveriam girar, para não extrapolar o tamanho de 64 bits, como o PWM que cada motor pode executar varia de 0 a 255, foi definido que qualquer valor acima de 255, representaria o valor de PWM menos 255 para trás, como pode ser visto na figura 10.2, logo temos que 255 representa o motor girando com potência máxima para frente e 510 representa o motor girando com potência máxima para trás.

## *Referências*

- [1] Relatório Unesco Sobre Ciência. Disponível em: <http://unesdoc.unesco.org/images/0018/001898/189883por.pdf>. Acesso em: 26 fev. 2015.
- [2] RoboCup. Disponível em: <http://www.robocup.org.br/objetivo.php>. Acesso em: 27 out. 2014.
- [3] Olimpíada Brasileira de Robótica. Disponível em: <http://www.obr.org.br/>. Acesso em: 01 fev. 2015.
- [4] Latin American and Brazilian Robotics Competition 2014. Disponível em: <http://www.cbrobotica.org/>. Acesso em: 01 fev. 2015.
- [5] Latin American and Brazilian Robotics Competition 2014. Disponível em: <http://www.cbrobotica.org/>. Acesso em: 01 fev. 2015.
- [6] Laboratório de Sistemas Inteligentes e Robótica. Disponível em: <https://www.facebook.com/sirlab.faeterj>. Acesso em: 28 out. 2014.
- [7] FAETERJ. Disponível em: <http://www.cptipetropolis.net.br/joomla/index.php>. Acesso em: 28 out. 2014.
- [8] CBR: IEEE Very Small Size. Disponível em: [http://www.cbrobotica.org/?page\\_id=81&lang=pt](http://www.cbrobotica.org/?page_id=81&lang=pt). Acesso em: 28 out. 2014.
- [9] The world's largest professional association for the advancement of technology. Disponível em: <https://www.ieee.org/index.html>. Acesso em: 01 fev. 2015.
- [10] Mirobot: Micro Robot World Cup Soccer Tournament. Disponível em: [http://www.fira.net/contents/sub03/sub03\\_3.asp](http://www.fira.net/contents/sub03/sub03_3.asp). Acesso em: 28 out. 2014.
- [11] Sacchetin, C. M. Análise e implementação de algoritmos para localização e mapeamento de robôs móveis baseada em computação reconfigurável, 2005. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-22082014-143916/pt-br.php>. Acesso em: 29 out. 2014.
- [12] Costa, A. H. e Pegoraro, R. Construindo Robôs Autônomos para Partidas de Futebol: O Time Guaraná. SBA Controle e Automação, 2000. Disponível em: <http://www.sba.org.br/revista/vol11/v11a259.htm>. Acesso em: 29 out. 2014.
- [13] NASA. Machine Vision Fundamentals: How to Make Robots 'See', 2011. Disponível em: <http://www.techbriefs.com/component/content/article/23-ntb/features/feature-articles/10531>. Acesso em: 04 nov. 2014.
- [14] OpenCV: About. Disponível em: <http://opencv.org/about.html>. Acesso em: 04 nov. 2014.

- [15] Perl, Python, Ruby, PHP, C, C++, Lua, tcl, javascript and Java comparison. Disponível em: <http://raid6.com.au/~onlyjob/posts/arena/>. Acesso em: 05 nov. 2014.
- [16] Stroustrup, B. Why C++ is not just an Object-Oriented Programming Language. Disponível em: <http://www.stroustrup.com/oopsla.pdf>. Acesso em: 05 nov. 2014.
- [17] Joaquim, A. O. e Knopman, J. Tipos de Dados, 2001. Disponível em: <http://equipe.nce.ufrj.br/adriano/algoritmos/apostila/tipos.htm>. Acesso em: 06 nov. 2014.
- [18] Cor: fenômeno ótico. Disponível em: <http://www.ufpa.br/dicas/htm/htm-cor4.htm>. Acesso em: 06 nov. 2014.
- [19] Sistemas de cores. Disponível em: <https://fperrotti.wikispaces.com/Sistemas+de+cores>. Acesso em: 07 nov. 2014.
- [20] Süsstrunk, S., Buckley, R. e Swen, S. Standard RGB Color Spaces. Disponível em: <http://infoscience.epfl.ch/record/34089/files/SustrunkBS99.pdf?version=3>. Acesso em: 09 nov. 2014.
- [21] Smith, A. R. Color gamut transform pairs. Computer Graphics, 1978. Disponível em: <http://infoscience.epfl.ch/record/34089/files/SustrunkBS99.pdf?version=3>. Acesso em: 09 nov. 2014.
- [22] OpenCV API Reference. Disponível em: <http://docs.opencv.org/modules/refman.html>. Acesso em: 09 nov. 2014.
- [23] Lindeberg, T. Feature Detection with Automatic Scale Selection, 2000. Disponível em: <ftp://ftp.nada.kth.se/CVAP/reports/cvap198.pdf>. Acesso em: 09 nov. 2014.
- [24] Bay, H., Tuytelaars, T. e Gool, L. V. SURF: Speeded Up Robust Features. Disponível em: [https://lirias.kuleuven.be/bitstream/123456789/71383/1/Bay\\_Tuytelaars\\_VanGool-surf-eccv06.pdf](https://lirias.kuleuven.be/bitstream/123456789/71383/1/Bay_Tuytelaars_VanGool-surf-eccv06.pdf). Acesso em: 10 nov. 2014.
- [25] Lowe, G. D. Object Recognition from Local Scale-Invariant Features, 1999. Disponível em: <http://www.cs.ubc.ca/~lowe/papers/iccv99.pdf>. Acesso em: 10 nov. 2014.
- [26] A great documentation place for Linux commands. Disponível em: <http://www.linux-commands-examples.com/uvcdynctrl>. Acesso em: 01 fev. 2015.
- [27] Silva, O. M. Campos potenciais modificados aplicados ao controle de múltiplos robôs, 2011. Disponível em: <http://www.teses.usp.br/teses/disponiveis/55/55134/tde-23112011-090055/pt-br.php>. Acesso em: 11 nov. 2014.
- [28] Angonese, T. A., Rosa, F. F. P. Estação de Controle em Solo com Funcionalidades de Voo para Múltiplos VANTs. Simpósio Brasileiro de Automação Inteligente. Disponível em: <http://www.sbai2013.ufc.br/pdfs/3732.pdf>. Acesso em: 12 nov. 2014.
- [29] Só física: Campo Elétrico. Disponível em: <http://www.sofisica.com.br/conteudos/Eletromagnetismo/Eletrstatica/campo.php>. Acesso em: 12 nov. 2014.
- [30] Khatib, O. Real-Time Obstacle Avoidance for Manipulators and Mobile Robots. Disponível em: <http://ijr.sagepub.com/content/5/1/90.short>. Acesso em: 13 nov. 2014.

- [31] Goodrich, A. M. Potential Fields Tutorial. Disponível em: <http://students.cs.byu.edu/~cs470ta/readings/Pfields.pdf>. Acesso em: 14 nov. 2014.
- [32] Ribeiro, E. J. Teoria de Controle Supervisório de Sistemas a Eventos Discretos. Disponível em: <http://www.das.ufsc.br/~cury/cursos/apostila.pdf>. Acesso em: 14 nov. 2014.
- [33] Araki, M. PID Control. Disponível em: [http://www.eolss.net/ebooks/Sample\\_20Chapters/C18/E6-43-03-03.pdf](http://www.eolss.net/ebooks/Sample_20Chapters/C18/E6-43-03-03.pdf). Acesso em: 14 nov. 2014.
- [34] Astron, K, J. e Hagglund, T. "PID Control" The Control Handbook, IEEE Press, 1996. Disponível em: <http://www.ni.com/white-paper/3782/pt/>. Acesso em: 15 nov. 2014.
- [35] PROJETO DE CONTROLADORES PID USANDO MÉTODO DE ZIEGLER NICHOLS. Disponível em: <http://professorgustavo.weebly.com/praacutetica-4---projeto-de-pid-usando-ziegler-nichols-i.html>. Acesso em: 22 jan. 2015.
- [36] Arduino: Overview. Disponível em: <http://arduino.cc/en/Main/ArduinoBoardUno>. Acesso em: 16 nov. 2014.
- [37] Holmes, D.; Lipo, T. Pulse Width Modulation for Power Converters: Principles and Practice, 2003. Disponível em <http://ieeexplore.com/xpl/bkabstractplus.jsp?bkn=5264450>. Acesso em: 17 nov. 2014.