

SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS

RELATÓRIO FINAL

orientador: Flavio Tonidandel
Departamento: Ciência da Computação
Aluno: Erivelton Gualter dos Santos
Número de matrícula: 11.210.368-4

RESUMO

Analizando um jogo de futebol é visível que, quanto mais passes é realizado com eficiência durante o jogo, é mais difícil para o oponente ficar com a posse da bola, resultando em mais chances de realizar o gol. Com essa avaliação, este projeto de iniciação científica, tem com objetivo o aperfeiçoamento do sistema de passe, inserindo o controle de chute, recepção da bola.

Para a conclusão desta atividade, serão estudadas as variáveis do jogo, como os efeitos físicos da bola e do robô durante o jogo. Inicialmente serão realizadas análises do jogo através da gravação de dados, como a velocidade que a bola alcança. Sendo ela o principal objeto de estudo, pois deve ser realizada a leitura da sua posição correta, para assim determinar o controle preciso para chutá-la.

SUMÁRIO

1.	INTRODUÇÃO.....	4
1.1	OBJETIVO	4
1.2	JUSTIFICATIVA	4
2.	REVISÃO BIBLIOGRÁFICA	5
2.1	O FUTEBOL DE ROBÔS	5
2.2	O FUTEBOL DE ROBÔS NA FEI	5
2.3	DESCRIÇÃO DO HARDWARE	5
2.3.1	DISPOSITIVO DE CHUTE	6
2.3.2	FUNCIONAMENTO DO DISPOSITIVO DE CHUTE.....	6
2.3.3	SOLENOÍDE.....	7
2.3.4	INTENSIDADE DA FORÇA DE CHUTE DO SOLENOIDE	8
2.4	VISÃO COMPUTACIONAL.....	8
2.4.1	CÁLCULO DA VELOCIDADE DA BOLA	9
2.4.2	CALIBRAÇÃO DA CÂMERA	10
2.5	DESCRIÇÃO DO SOFTWARE.....	11
2.5.1	MÓDULO DO “STATE PREDICTOR”	12
2.5.2	MÓDULO DA “STRATEGY”	12
2.5.3	MÓDULO DA “SKILL”	12
2.6	GRAVAÇÃO DE DADOS.....	13
2.7	TIPOS DE PASSE	14
3.	METODOLOGIA	15
4.	PROJETO	15
4.1	ANÁLISE DA INTENSIDADE DE CHUTE.....	15
4.1.1	INTENSIDADE DO CHUTE	16
4.1.2	EXTRAÇÃO DAS VARIÁVEIS DA GRAVAÇÃO DE DADOS	16
4.2	ESCOLHER INTENSIDADE PARA CHUTAR A BOLA.....	19
4.3	IMPLEMENTAÇÃO DAS FUNÇÕES AUXILIARES NO NAMESPACE SUPPORT	21
4.3.1	INTERSECÇÃO ENTRE DUAS RETAS	21
4.3.2	INTERSECÇÃO ENTRE UMA RETA E UM CÍRCULO	24
4.3.3	INTERSECÇÃO ENTRE DOIS CÍRCULOS	26
4.3.4	PONTO TANGENTE DE UM CÍRCULO	28
4.4	DETERMINAÇÃO DO MELHOR PONTO PARA EFETUAR O CHUTE AO GOL.....	30
4.5	PONTO EM QUE A BOLA INTERCEPTA O CAMPO	31
4.6	INTERCEPTAR A BOLA.....	32
4.7	CONSTRUÇÃO DO ALGORITMO DE PASSE.....	34
5.	EXPERIMENTOS	38
6.	CONCLUSÃO	41
7.	REFERÊNCIAS BIBLIOGRÁFICAS.....	42
8.	ANEXOS.....	44
8.1	ARTIGOS PUBLICADOS	44
8.2	NAMESPACE COM FUNÇÕES MATEMÁTICAS E TRIGONOMÉTRICAS.....	47
8.3	BESTPOINTTOSHOOT	54
8.4	EXECUTE: PASSTOROBOT	57

1. INTRODUÇÃO

O projeto de Futebol de Robôs no Centro Universitário da FEI, conhecido como ROBOFEI [1], teve iniciativa dos professores Flavio Tonidandel e Reinaldo Bianchi em 2003, inserindo alunos de graduação e pós-graduação a pesquisa na área da robótica e inteligência artificial, com motivação as competições de futebol robótico.

No decorrer desses anos, foram feitas várias pesquisas, resultando na criação de robôs na categoria *Very Small* e *Small Size* [2], através de algumas publicações de artigos de iniciação científica e teses de conclusão de curso na área da Mecânica, Eletrônica e Software.

Muitos alunos participaram deste projeto, realizando pesquisas com o intuito do desenvolvimento e aprimoramento dos robôs. Com isto, possibilitou uma qualidade estável no sistema de visão computacional, controle, hardware e software, que foram sendo aprimorados no decorrer do tempo.

Apesar de longas pesquisas realizadas pelos alunos, e muitas vitórias na categoria *Small Size* [2], ainda há muito por fazer. Nossa colocação no âmbito nacional é satisfatória, porém ainda precisamos de um aperfeiçoamento para obter ótimos resultados em competições internacionais. Com isso, é necessário continuar realizando novos estudos, buscando sempre o crescimento da equipe.

1.1 OBJETIVO

Essa iniciação científica tem como objetivo inserir um sistema de passe mais eficiente no robô da categoria *Small Size* [2], realizando o estudo completo dos fatores, como: força do chute, e recebimento da bola.

1.2 JUSTIFICATIVA

Na categoria *Small Size* [2] a equipe ROBOFEI [1] possui uma classificação satisfatória em âmbito nacional, sagrando-se campeã no campeonato nacional em 2011. Em âmbito internacional, ficou entre os 8 melhores times na *RoboCup* [3] realizada na Cidade do México neste ano. Essas conquistas são os resultados obtidos dos trabalhos de alunos de graduação e pós-graduação, que desenvolveram pesquisas e o desenvolvimento científico dentro do projeto nos últimos anos. Entretanto, existem muitas tarefas a serem feitas e aprimoradas.

Há a necessidade de abordar o aperfeiçoamento do sistema de passe a partir de jogos realizados em campeonatos nacionais e internacionais. Atualmente, durante as partidas, no momento em que a equipe ROBOFEI [1] permanece com a posse de bola, geralmente realiza um chute ao gol, que não é convertido, ou mesmo um passe deficiente, resultando na perda da bola.

Deve-se, portanto, estudar e desenvolver um sistema de passe mais eficiente, com o intuito de aumentar o índice de finalizações ao gol. Além disso, deve-se aprimorar a qualidade de recebimento de bola, para que esta não seja perdida durante o procedimento de passe.

2. REVISÃO BIBLIOGRÁFICA

2.1 O FUTEBOL DE ROBÔS

O futebol de robôs tem como objetivo o desenvolvimento de robôs autônomos, para realizar tarefas sem a intervenção humana. Ela é uma plataforma de pesquisa, que atua principalmente na área da robótica e inteligência artificial.

Existem muitas equipes desenvolvendo pesquisa nesta área, com o principal foco os campeonatos robóticos. O principal organizador desses campeonatos, é a *RoboCup* [3], que foi criada em 1993 pelo Dr Hiroaki Kitano, com a ideia de ampliar Sistemas Autônomos Multi-Agentes.

Temos outras competições de robótica, como a Competição Brasileira de Robótica[4], onde se apresenta a disputa de futebol robótico em diversas categorias. Contém congressos paralelamente, apresentando publicações acadêmicas com a função de difundir o conhecimento entre os pesquisadores do mundo inteiro.

2.2 O FUTEBOL DE ROBÔS NA FEI

O projeto futebol de robôs na FEI iniciou-se no ano de 2003, com pesquisas na categoria Mirosot [5], com robôs baseados nas equipes FUTEPOLI[7] e Guaraná[8], nomeando a equipe como ROBOFEI [1]. Primeiramente, a equipe realizava pesquisa na simulação computacional, porém no mesmo ano surge a motivação de manufaturar seus próprios robôs, durante o 6º SBAI (Simpósio Brasileiro de Automação Inteligente)[6], que ocorreu em conjunto com a *Second IEEE Student Robotics Competition*, realizando a primeira competição do Futebol de Robôs, categoria *Very Small*. Deste ano em diante, o projeto foi se desenvolvendo a ponto de permitir novos robôs e a participação em categorias mais avançadas, como a Small Size da Robocup Federation [3].

2.3 DESCRIÇÃO DO HARDWARE

Os robôs utilizados na categoria *Small Size* [2], são robôs omnidirecionais, mais conhecidos como robôs móveis por se deslocar em qualquer direção e realizar movimentos de translação. Para tal tarefa é feita a combinação da rotação de suas quatro rodas.

As rodas omnidirecionais são projetadas para diminuir o atrito radial no eixo do motor. A fim de alcançar isso, a roda é construída usando pequenas rodas ao longo da periferia. O robô da equipe ROBOFEI [1], possui 4 rodas onde é possível realizar movimentos de translação e deslocamentos na diagonal sem nenhum esforço brusco. Lembrando que o movimento do robô é a soma vetorial das velocidades de cada roda.

Além do sistema de movimentação, o robô possui um dispositivo de drible, que consiste em um motor acoplado a um eixo com uma borracha ao redor, que ao girar prende a bola devido ao atrito. Também temos o sistema de chute, que é composto por dois solenoides, sendo um deles para o chute horizontal, e o outro para o chute no plano tridimensional (resultando uma trajetória parabólica da bola).

2.3.1 DISPOSITIVO DE CHUTE

O dispositivo de chute horizontal (veja a figura 1) é composto por um solenoide de 30 mm de diâmetro, feito de Nylon, enrolado com fio de cobre esmaltado AWG 21. Seu êmbolo contém 14 mm de diâmetro feito de aço SAE1020. O dispositivo de chute parabólico (conhecido também, como *chip-kick*) possui um solenoide retangular localizado embaixo do robô (Veja a figura 2).

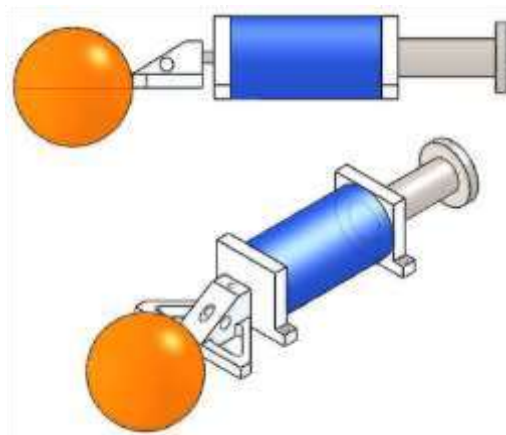


Figura 1 - Sistema de chute horizontal

Fonte: RoboFEI 2010 Team Description Paper, 2010, Gurzoni, José Angelo, p.5

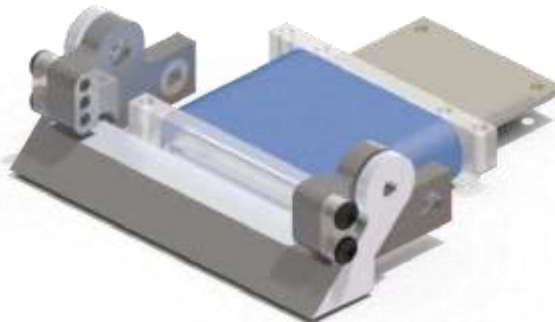


Figura 2 - Sistema de chute parabólico

Fonte: RoboFEI 2010 Team Description Paper, 2010, Gurzoni, José Angelo, p.6

2.3.2 FUNCIONAMENTO DO DISPOSITIVO DE CHUTE

O dispositivo de chute é um dispositivo eletromecânico com excitação simples. Ou seja, basicamente é uma bobina alimentada por uma corrente elétrica, contendo apenas uma indutância própria e neste caso, não existe o efeito da mútua indutância. Ele é denominado por solenoide.

2.3.3 SOLENÓIDE

O solenoide [10] é uma bobina enrolada por um fio com N espiras, e no instante em que percorre uma corrente elétrica é gerado um campo magnético. Ele é constituído por duas partes, sendo uma parte fixa que é enrolada por um fio e outra parte móvel, separado por um entreferro, mais conhecido como êmbolo.

O campo magnético [10] pode ser criado através de um ímã permanente ou por uma corrente elétrica, que é o nosso caso. Pela figura 3, podemos verificar a representação do campo magnético pelas linhas de campo que saem do polo Norte e chegam ao polo Sul. Essas linhas de campo magnético são denominadas como fluxo magnético.

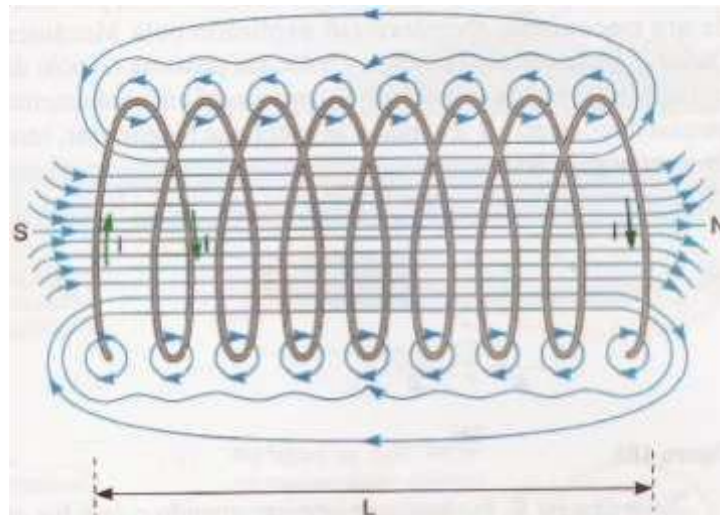


Figura 3 – Campo Magnético por corrente elétrica em solenoides

Fonte: <http://www.mundoeducacao.com.br/fisica/>, CABRAL, Marina. Acesso em: 14 Set. 12

Podemos calcular o campo magnético [9] através da seguinte equação:

$$B = \frac{\mu \cdot N \cdot I}{l} [T] \quad (1)$$

Sabendo que:

- μ : Permeabilidade magnética do material
- N : Número de espiras
- I : Corrente elétrica
- l : comprimento da bobina

O material da parte móvel é fabricado de um material ferromagnético. Este tipo de material é muito usual, pois apresenta valores elevados de permeabilidade relativa.

A permeabilidade magnética (μ) é uma grandeza magnética que permite quantificar o campo magnético de um material. Ela é definida como a relação entre a indução magnética (B) e a intensidade do campo magnético (H) [10].

$$\mu = \frac{B}{H} \left[\frac{H}{m} \right] \quad (2)$$

A permeabilidade relativa do material é obtida através da relação da permeabilidade do material e a permeabilidade magnética do vácuo.

É importante que a parte móvel, ser de material ferromagnético, pois na presença de um campo magnético externo, a sua magnetização ocorre de forma intensa no mesmo sentido do campo magnético [9] aplicado. Com isso podemos definir que o chute é realizado no momento em que é gerado um campo magnético, lembrando que este campo é provocado pela excitação de corrente elétrica no solenoide.

2.3.4 INTENSIDADE DA FORÇA DE CHUTE DO SOLENOIDE

Quando aplicado uma corrente na bobina do solenoide, será criado um fluxo magnético de acordo com o sentido da corrente. Podemos definir o sentido do fluxo, através da regra da mão direita, onde o polegar indica o sentido da corrente e os demais dedos curvarão ao redor do fio, definindo a direção das linhas de campo.

O fluxo magnético [9] gerado pela corrente percorre o material ferromagnético junto com o entreferro, resultando no deslocamento do êmbolo com determinada intensidade de força. Podemos concluir que a força magnética [10] gerada é convertida para uma força mecânica.

Podemos determinar a força através da seguinte equação:

$$F(x) = \frac{i^2}{2} \cdot \frac{dL}{dx} [N] \quad (3)$$

2.4 VISÃO COMPUTACIONAL

O sistema de visão computacional do futebol de robôs tem o objetivo de reconhecer os objetos dentro de campo, como as posições do robô e da bola. E em seguida transmitir os dados do jogo para as equipes.

Até o ano de 2010 a *RoboCup* [3] permitia que os times da categoria *Small Size League* [2], criassem seu próprio sistema de visão. Porém, diversas equipes tiveram problemas durante os jogos, devido a ruídos e predição errada da segmentação de cores. Para solucionar este problema, os comitês responsáveis decidiram migrar para um sistema de visão universal, chamado SSL-Vision [11] (Veja a visão geral do SSL-Vision na figura 4).

O sistema de visão consiste em duas câmeras sobre o campo, uma para cada lado. Estas câmeras são ligadas num servidor central, onde o SSL-Vision está configurado para transmitir a visão do campo.

O SSL-Vision [11] foi criado em linguagem C++ no ambiente Linux, utilizando o Qt toolkit. Também é utilizada uma interface gráfica (GUI) para facilitar a configuração e ajuste de parâmetros de cores dos robôs.

O sistema de visão transmite as posições e rotações do robô e da bola no momento que os objetos são detectados. Porém, esses dados são enviados de ambas as câmeras e não realiza

nenhum filtro para suavizar os objetos. Logo, a equipe deve mesclar os dados das imagens entre as câmeras, e filtrar corretamente as suas posições.

Esses pacotes de dados são codificados utilizando o Google Protocol Buffers e a comunicação entre o sistema de visão e o cliente é feita via UDP Multicast na porta 10002 e endereço de Multicast 224.5.23.2.

Basicamente os dados provenientes da SSL Vision são: □

Número de robôs azuis dentro do campo;

□ Número de robôs amarelos dentro do campo;

□ Número de identificação de cada robô;

□ Coordenada da posição de cada robô no campo;

□ Rotação do robô;

□ Coordenada da posição da bola;

□ Número de bolas;

□ Número de identificação da câmera (câmera 0 ou câmera 1); □ Tempo de leitura da imagem (horário Unix).

2.4.1 CÁLCULO DA VELOCIDADE DA BOLA

O SSL-Vision [11] fornece as posições da bola e dos robôs assim como suas orientações. Porém, durante a leitura e transmissão de dados, pode haver ruídos na detecção das posições e orientação dos objetos. Para tratar possíveis erros, a equipe ROBOFEI utilizou um *Filtro de Kalman* [13] para fazer *tracking* da bola e dos robôs.

Este filtro foi escolhido por ser uma técnica eficaz de realizar a predição e correção da estimativa da localização de agentes em sistemas dinâmicos. Com as posições e orientações corretas, agora só basta determinar a velocidade da bola. Para determinar a velocidade da bola, são utilizadas as seguintes relações. Onde, primeiramente é calculada a componente da velocidade X e Y, e depois é calculado o módulo da velocidade.

$$Velocidade \rightarrow X = \frac{(Posição da bola atual \rightarrow x) - (Posição da bola anterior \rightarrow x)}{Tempo atual - Tempo anterior} \quad (4)$$

$$Velocidade \rightarrow Y = \frac{(Posição da bola atual \rightarrow y) - (Posição da bola anterior \rightarrow y)}{Tempo atual - Tempo anterior} \quad (5)$$

$$Módulo da Velocidade = \sqrt{(Velocidade \rightarrow X)^2 + (Velocidade \rightarrow Y)^2} \quad (6)$$

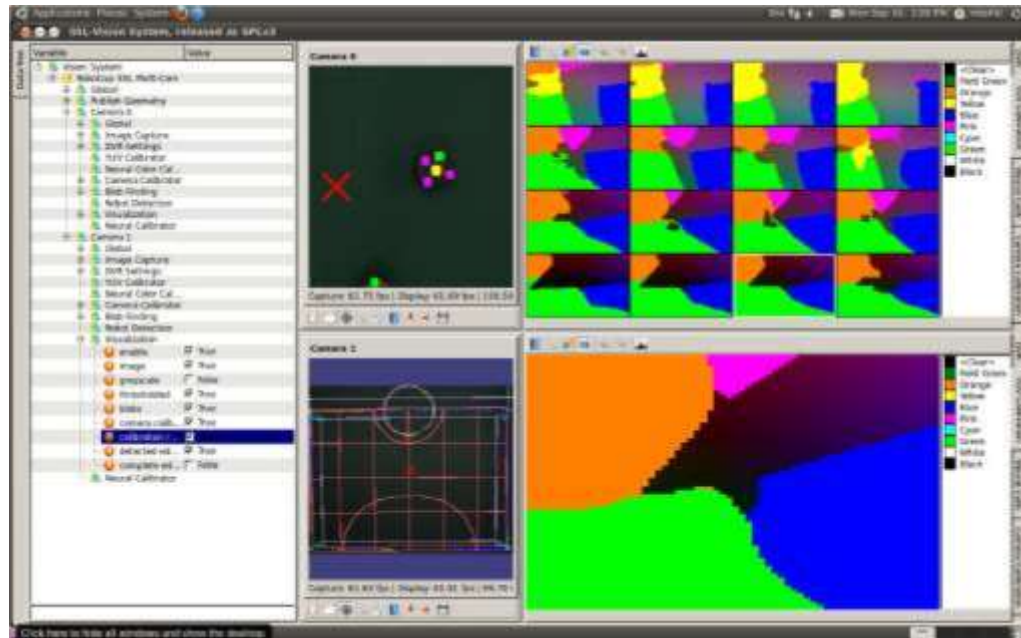


Figura 4 – Visão geral do software SSL-Vision

Fonte: Autor

2.4.2 CALIBRAÇÃO DA CÂMERA

A calibração da câmera [12] deve ser realizada constantemente, pois a luminosidade do ambiente varia ao longo do dia. Portanto é de extrema importância conhecer o procedimento de calibração. Veja a seguir o procedimento de calibração e na figura 5 o resultado da calibração:

- Primeiramente são definidos os *Corners Points*, para isso o usuário deve ativar a *Camera Calibrator* no lado direito. Logo em seguida, tem que colocar os pontos *Left Corner*, *Right Corner*, *Left Centerline* e *Right Centerline* nos devidos lugares, com o intuito de restringir metade do campo.
- Ajustar a altura da câmera, realizando a medição manual da sua altura até o campo.
- Pressione “*Do initial calibration Camera Calibration*” para realizar a calibração inicial com base nos quatro pontos do campo.
- Ajustar a distorção do campo através de um controle deslizante no *Camera Calibrator*.
- Para calibrar o padrão de cores dos robôs, é necessário pressionar sobre a cor desejada, e em seguida preencher com a cor correspondente na tela *YUV Calibrator*.

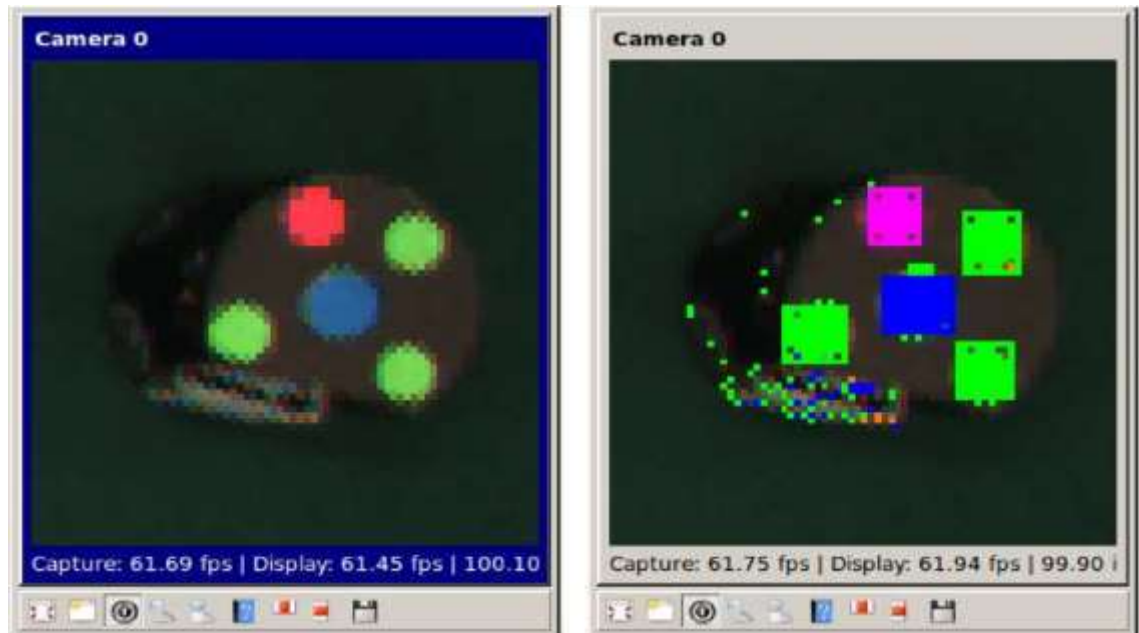


Figura 5 – Resultado da calibração
Fonte: Autor

2.5 DESCRIÇÃO DO SOFTWARE

O software do ROBOFEI [1] consiste em diversos módulos independentes. Cujo objetivo é tornar os robôs independentes e que cooperem entre si como um sistema multiagentes. Ou seja, torná-los autônomos com capacidade de executar determinadas funções de acordo com o ambiente sem comunicação entre eles e ainda facilitar a manutenção e inserção de funções.

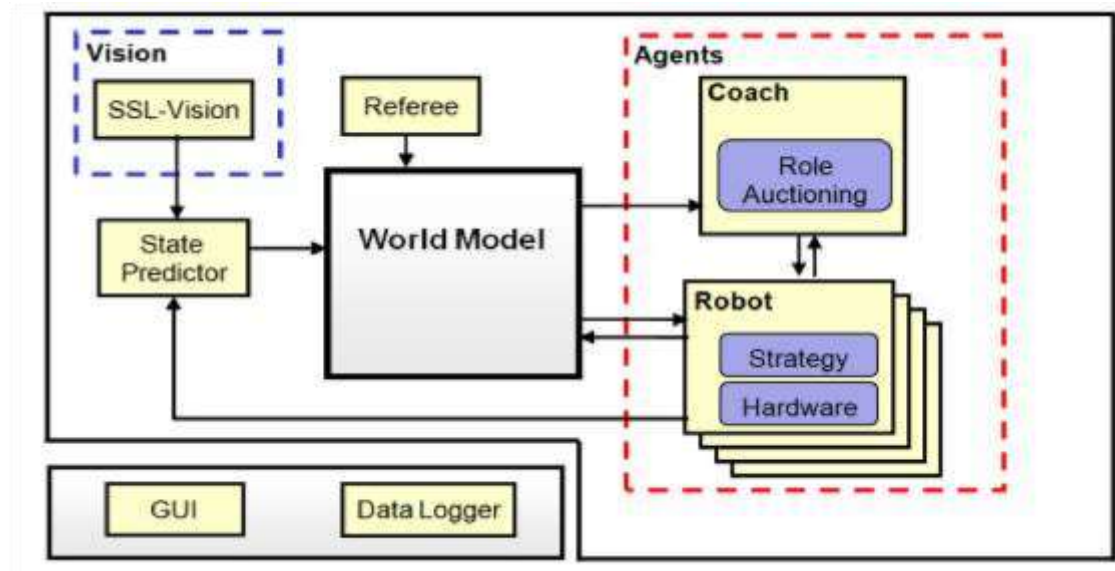


Figura 6 – Diagrama de bloco do software do ROBOFEI [1]
Fonte: RoboFEI 2010 Team Description Paper, 2010, Gurzoni, José Angelo, p.7

Alguns módulos importantes para o estudo do sistema de passe serão abordados a seguir:

2.5.1 MÓDULO DO “STATE PREDICTOR”

O módulo “*State Predictor*” recebe os dados do sistema de visão, chamado de SSLVision, e realiza a previsão das posições dos objetos em campo (robô e bola) do seu tempo de captura para o instante atual (o presente). Para a predição da bola é usado um estimador recursivo, onde estima dados de um sistema dinâmico, conhecido como *Filtro de Kalman* [13].

Para obter o resultado preciso da predição, deve-se efetuar a calibração correta das cores do robô e da bola. A calibração correta está descrita na “Calibração da Câmera”.

2.5.2 MÓDULO DA “STRATEGY”

Para que a estratégia do ROBOFEI [1] adote o critério de multi-agentes, o software foi dividido em camadas, onde cada grupo é composto por certas funções. Temos a camada mais baixa, que é responsável por apenas ativar ou desativar um módulo do hardware, como enviar comando de chute, ativar ou desativar o sistema de drible e se locomover para um determinado ponto.

Outra camada é o que chamamos de *Skills*, que são compostas por um conjunto de funções básicas para que o jogo funcione corretamente, como chutar a bola no gol, passar a bola para um jogador da mesma equipe. Essas habilidades são empregadas em outra camada, onde elas são combinadas com outras habilidades dependendo do papel do jogador durante o jogo. Existem papéis de defensor, atacante e goleiro.

2.5.3 MÓDULO DA “SKILL”

No início do jogo é determinado quais características principais o robô vai possuir. Sendo elas: robôs atacantes, defensores e goleiro. Esses robôs assumem uma responsabilidade durante o jogo, de serem mais ofensivos, defensivos ou simplesmente defender o gol. Porém essas características não são alteradas durante o jogo.

No entanto, todos os robôs possuem as mesmas habilidades, como chutar para o gol, driblar, passar a bola e até mesmo cobrar lateral ou pênalti. Essas funções são conhecidas como *skill*, elas são uma combinação de habilidades e lógicas para que o jogo funcione adequadamente. Veja a seguir a lista de *skills* atualmente:

- ShootToGoal(): Chutar no gol quando estiver com a bola
- FollowOpponent(): Realiza a marcação dos jogadores oponente;
- CatchBall(): Robô irá atrás da bola;
- DeltaDefense(): Os robôs se posicionam estrategicamente, realizando o bloqueio da passagem da bola ao gol e marcando os atacantes adversário;
- FindGoodPosition(): Responsável pelo posicionamento do robô;
- DefendGoal(): Realiza exclusivamente a defesa do gol, evitando que o time adversário marque pontos;

RefereeCircle(): Posicionamento do robô quando o jogo estiver parado;
IndirectFreeKick(): Cobrança do tiro indireto;
IndirectFreeKickOpponent(): Os robôs posicionam estrategicamente no tiro indireto do time oponente;
DirectFreeKick(): Cobrança do tiro direto;
DirectFreeKickOpponent(): Os robôs posicionam estrategicamente no tiro direto do oponente;
KickOff(): Realiza a saída da bola no centro o campo;
KickOffOpponent(): Os robôs posicionam estrategicamente na saída da bola do time adversário.

As seleções das *skill* são totalmente dinâmicas, no qual varia de acordo com a situação do jogo. Essa triagem é realizada em outro trecho do código, sendo ela o bloco *Role*. Ela é composta por diversas combinações de *skills* e depende das coordenada dos robôs e da bola no campo.

2.6 GRAVAÇÃO DE DADOS

A gravação de dados é o registro de eventos relevantes de um software. Esse registro pode ser utilizado para que o programador ou usuário conheça o comportamento detalhado no passado. Esse processo é conhecido como gravação de log de dados.

A aplicação da gravação de dados no projeto ROBOFEI [1], é utilizada para gravar os jogos, ou gravar determinados testes, para que seja reproduzido novamente. Para isso utilizamos o software SSL-LogPlayer 1.1 (Veja a tela principal na figura 7).

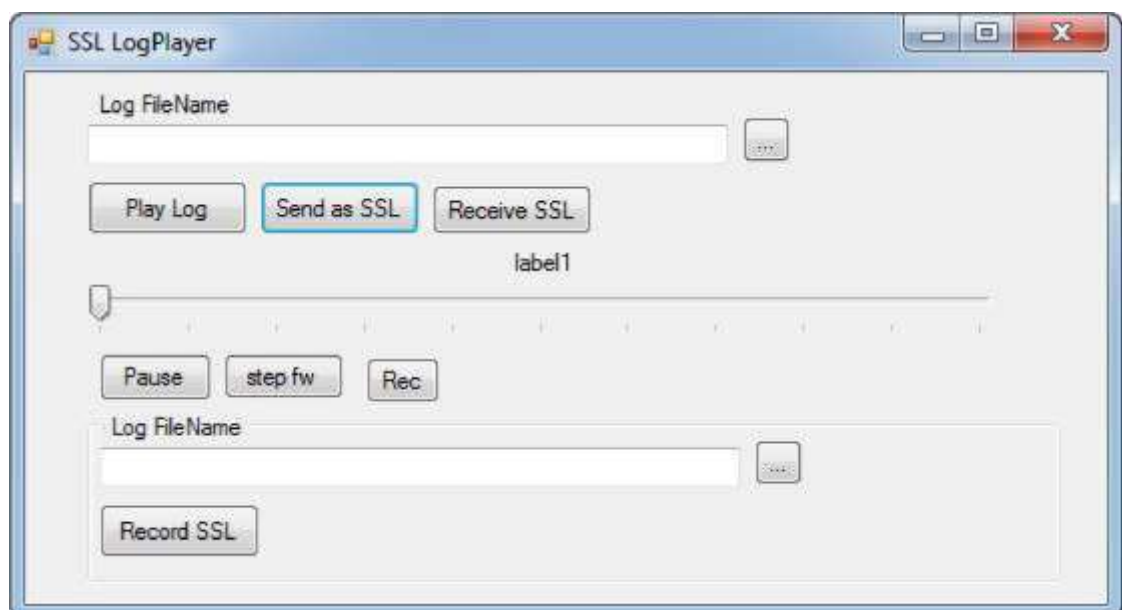


Figura 7 – Tela principal do software SSL LogPlayer

Fonte: Autor

Depois de realizado a gravação de dados, ou popularmente conhecido, como gravação de logs, ele pode ser assistido novamente, ou realizar a extração de algumas variáveis.

2.7 TIPOS DE PASSE

Analizando as equipes que mais se destacam na competição internacional, foram listados alguns tipos de passe mais usual. Essas informações foram encontradas em [14] e observadas nos eventos promovido pela *RoboCup* [3]. Veja a seguir:

- O passe simples consiste em único toque para o robô receptor, que vai efetuar um chute direto ao gol. A dificuldade neste método é realizar um passe preciso, no qual o robô receptor seja capaz de se posicionar para realizar o chute, e efetuá-lo com sucesso;

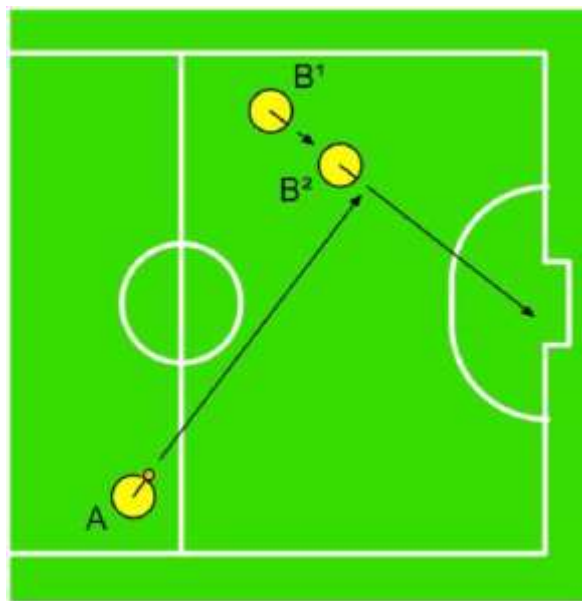


Figura 8 – Passe Simples
Fonte: Autor

- O passe simples com domínio da bola, consiste em um passe para o robô receptor, que receberá a bola de maneira precisa, sem a perda da bola, com o auxílio de um mecanismo mecânico (*roller*). Logo em seguida, será realizada uma avaliação para determinar a próxima jogada, como efetuar o chute ao gol, ou realizar outro passe;

- O passe entre três robôs consiste entre um robô (A) que possui o domínio da bola, outro robô (C) que vai realizar o chute ao gol, e enfim outro robô (B), cuja função é a mediação do passe entre o robô A e o robô C. Este passe é baseado no artigo “*Cooperative 3robot passing and shooting in the RoboCup Small Size League*” [15] do time CMUDragons.

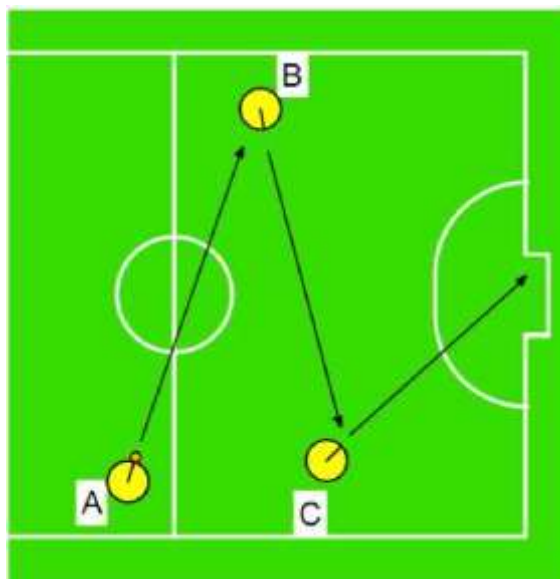


Figura 9 – Passe com três robôs

Fonte: autor

3. METODOLOGIA

Primeiramente, foi feita uma revisão bibliográfica sobre os assuntos relacionados à concretização de passar a bola para outro jogador. Foi estudado o funcionamento do sistema de chute, funcionamento e calibração do sistema de visão, arquitetura do software da equipe ROBOFEI [1] e tipos de passe utilizado pelos times da liga internacional.

Em seguida, foi realizada a gravação de diversos testes, onde foi possível realizar análises para determinar as intensidades de chute e normalizá-las para as competições futuras.

Com a análise das propriedades físicas da bola durante o jogo, foi possível iniciar a implementação do passe simples, e verificar a precisão do passe. Para esta tarefa, foi realizada a construção de outros algoritmos.

4. PROJETO

Neste relatório será apresentada a análise da intensidade do chute através da extração de dados, com objetivo de obter o controle da velocidade da bola durante o jogo. Também apresenta a inserção de alguns algoritmos para auxiliar na elaboração do sistema de passe, como funções para cálculos de trigonométricos e o aperfeiçoamento de chutar a bola ao gol.

4.1 ANÁLISE DA INTENSIDADE DE CHUTE

O objetivo deste experimento é obter as posições do robô e da bola através da gravação de dados de uma simulação real. Esses dados serão utilizados para analisar os efeitos físicos da bola durante o jogo, como a velocidade que ela alcança em cada intensidade de chute.

4.1.1 INTENSIDADE DO CHUTE

Através das características de construção do solenoide e a programação atual do controle do chute. Foi possível dividir a potência dissipada pelo solenoide em 15 níveis, onde para cada nível temos o tempo de descarga de 100 μ s.

Com este experimento foi possível identificar a velocidade máxima que a bola atinge em cada nível de disparo do chute. O procedimento adotado foi:

- Verificar se o sistema de chute do robô está funcionando corretamente ou existe alguma irregularidade;
- Realizar a calibração do software da visão (Descrito em “Calibração da Câmera”);
- Executar o software de gravação de dados (SSL-LogPlayer 1.1); □ Realizar a gravação dos 15 níveis de intensidade do chute.
- Extrair as posições da bola e o tempo de captura. □ Analisar os dados.

4.1.2 EXTRAÇÃO DAS VARIÁVEIS DA GRAVAÇÃO DE DADOS

Para realizar a extração dos dados, primeiramente deve realizar a execução do SSLLogPlayer 1.1 (Veja na figura 11 a tela do software LogPlayer 1.1 no momento de gravação), que é o mesmo software utilizado para realizar a gravação. Porém neste caso, é feito a leitura do log e enviado um pacote de dados para o software da estratégia (Veja a sua tela principal na figura 10), da mesma forma que acontece no jogo real.



Figura 10 – Tela principal do software SSL Futbots Strategy 2.6

Fonte: Autor

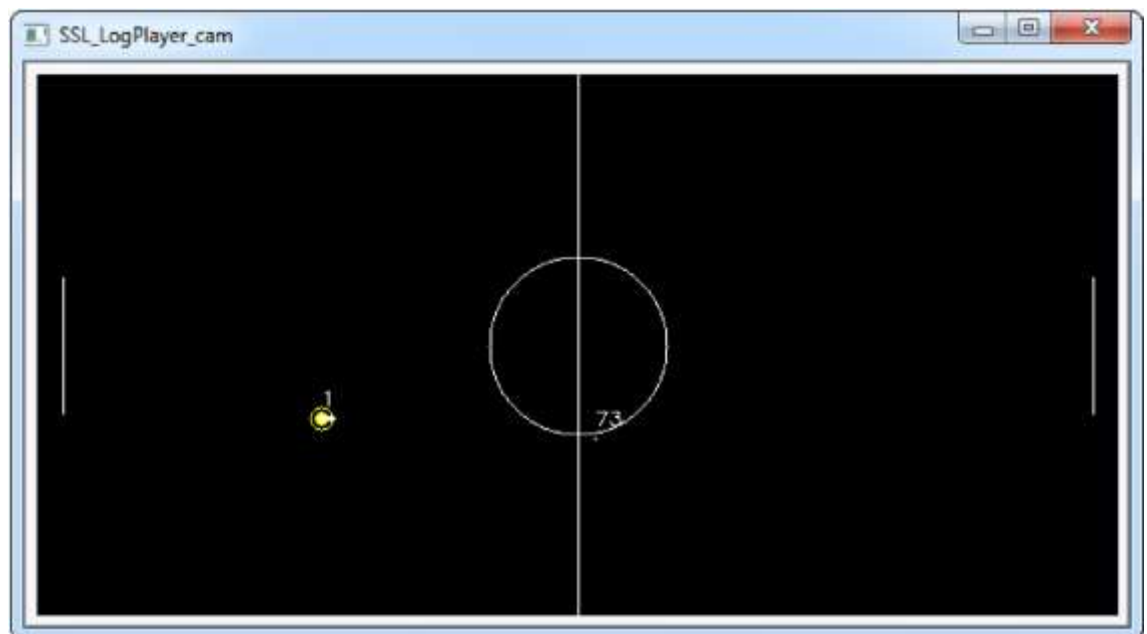


Figura 11 – Tela durante a gravação de dados
Fonte: Autor

Para extrair a posição da bola e o tempo de captura da visão simultaneamente, foi utilizado o método `AppendText()` da classe `File` [16], que tem a função de adicionar um texto codificado em UTF-8 em um arquivo existente no formato `.txt`.

Este método foi inserido no software principal e executado uma vez por ciclo. Onde foram extraídos os seguintes dados:

- Posição da Bola (coordenada X e Y no campo);
- Tempo de leitura;
- Velocidade da bola, obtida pelo *Predictor*;
- Id da câmera que fez a leitura;

Em seguida foram plotados esses dados num gráfico de Velocidade x Tempo. Veja um exemplo do nível 3 na figura 12.

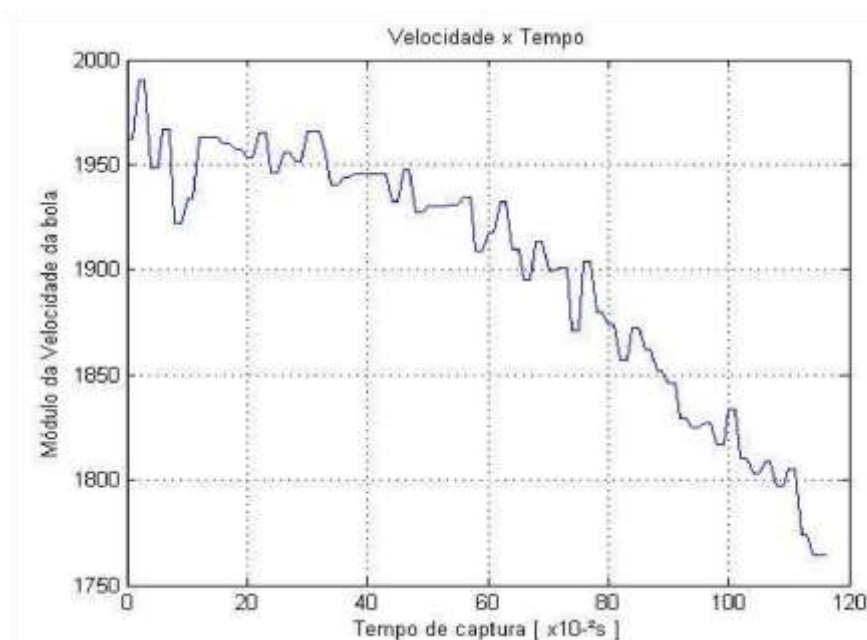


Figura 12 – Gráfico Velocidade x Tempo do nível 3
Fonte: Autor

Observando o gráfico, é possível verificar que existem ruídos durante a leitura da velocidade da bola. No entanto, as velocidades obtidas possuem uma tolerância aceitável, pois a variação dos valores de pico e vale dos ruídos da curva gerada, não ultrapassa a 0,025 m/s, que é relativamente irrelevante durante o jogo.

Também existe uma desaceleração da bola durante o percurso, ela é resultado da força de atrito entre a bola e o carpete. Esta força é contrário ao movimento, e depende da força normal exercida sobre ela.

Depois da análise obtida dos gráficos de cada intensidade, resultou os seguintes valores máximos da velocidade:

Nível de intensidade de Chute	Velocidade Obtida [m/s]
2	0,3
3	2
4	4
5	5
6	6
7	7,8
8	8,1
9	8,5

Tabela 1 – Tabela de Velocidades
Fonte: Autor

É importante observar que, de acordo com as regras da *RoboCup* [3], a velocidade da bola não pode ultrapassar os 8 m/s, logo as velocidades obtidas a partir do nível 9 em diante, não serão implementadas no software da equipe ROBOFEI [1].

4.2 ESCOLHER INTENSIDADE PARA CHUTAR A BOLA

Conhecer a velocidade da bola durante o jogo é importante, pois é necessário saber a força que deve ser aplicada para o robô receber a bola, sem intervenção de um robô oponente. Para determinar a velocidade necessária do passe, para que o robô oponente não intercepte a bola, será utilizado o algoritmo de PAUL BOURKE [17]. Esta solução determina a intersecção (ponto I) da reta formada entre a bola e o robô receptor, com a linha entre o robô oponente a esta reta (Fig. 13).

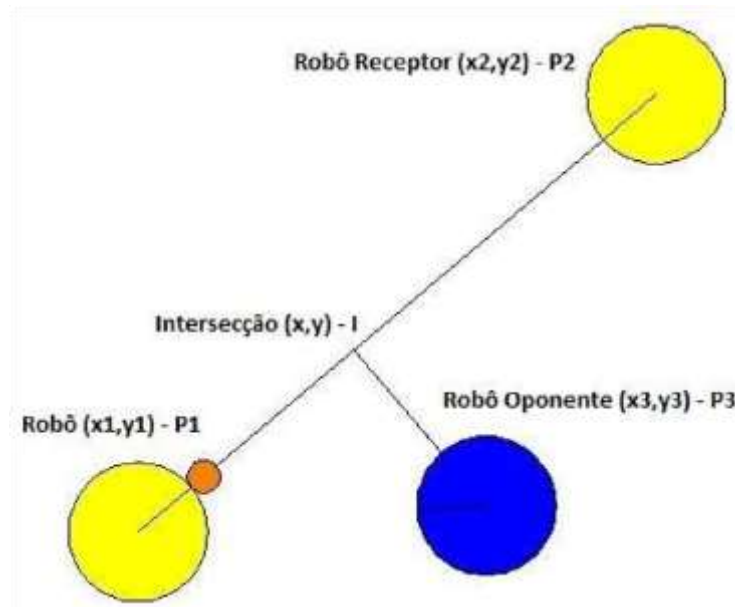


Figura 13 – Diagrama da Intervenção do robô oponente.
Fonte: Autor

O algoritmo de PAUL BOURKE [17] utiliza o segmento de reta entre os dois pontos P1 (x1, y1) e P2(x2, y2):

$$P = P1 + u(P2 - P1)$$

O robô oponente—P3 (x3, y3) está mais próximo ao segmento de reta, exatamente no ponto I, onde a reta formada entre o ponto P3 e a reta forma um ângulo de 90°. Sabendo que as retas entre os pontos P1-P2 e P3-P são ortogonais, logo o produto escalar entre eles é zero:

$$(P3 - P) \cdot (P2 - P1) = 0$$

Substituindo a equação de reta P, temos:

$$(P3 - P1 + u(P2 - P1)) \cdot (P2 - P1) = 0$$

Resolvendo a equação acima e isolando o u resulta em:

$$u = \frac{(x3 - x1)(x2 - x1) + (y3 - y1)(y2 - y1)}{\|P2 - P1\|^2}$$

Logo, para determinar o ponto I, deve-se substituir a expressão de u na equação da reta P, resultando nas seguintes equações paramétricas, onde a coordenada (x, y) corresponde ao ponto I:

$$\begin{aligned}x &= x1 + u(x2 - x1) \\y &= y1 + u(y2 - y1)\end{aligned}$$

Para que não ocorra a interrupção do passe, a bola deve passar pelo ponto I (x, y) antes do robô oponente. Utilizando a equação a seguir temos o tempo ($t1$) que o robô oponente leva para chegar até este ponto. Onde Δs é a distância entre o robô oponente e o ponto I, e Vm é a velocidade máxima do robô oponente.

$$t1 = \frac{\Delta s}{Vm}$$

Logo, o tempo ($t2$) que a bola percorre da sua origem até o ponto I deve ser menor que o tempo $t1$ ($t2 < t1$). Com essa avaliação, podemos determinar a velocidade da bola (Vb), onde ΔS é a distância entre a bola e o ponto I. Veja a equação:

$$Vb = \frac{\Delta S}{t1}$$

Com a extração dos dados da posição da bola durante o chute, foi possível normalizar esses dados, onde foi identificada a velocidade que a bola atinge em cada intensidade do chute. Veja a tabela 1 os dados obtidos da velocidade em cada intensidade de chute.

Com esses dados regularizados, é possível realizar o chute com a velocidade escolhida através da avaliação do tempo que o oponente irá interceptar a bola.

Na ausência de robôs oponentes nas proximidades da linha de passe, não é necessário utilizar o algoritmo apresentado para determinar a intensidade de chute. Para isso, foi analisada força aplicada na bola em função da distância empiricamente e plotadas no gráfico a seguir.



Figura 14 – Intensidade do Chute x Deslocamento
Fonte: Autor

4.3 IMPLEMENTAÇÃO DAS FUNÇÕES AUXILIARES NO NAMESPACE SUPPORT

Namespace permite organizar as classes, objetos e funções numa estrutura nomeada pelo programador. O principal objetivo é facilitar a organização do programa. A necessidade de criar uma estrutura deste tipo, com funções matemáticas e trigonométricas adaptadas para o software do ROBOFEI [1] surgiu devido às repetições de funções ao longo do código, que realizam a mesma tarefa, e também, para modularizar o software para futuros programadores. A princípio são funções simples, porém de grande importância. Veja a seguir:

4.3.1 INTERSECÇÃO ENTRE DUAS RETAS

Para determinar a intersecção entre duas retas temos como variável de entrada os pontos P1, P2, P3 e P4. Formando a reta t, através dos pontos P1 e P2 e a reta s pelos pontos P3 e P4. Veja a figura a seguir:

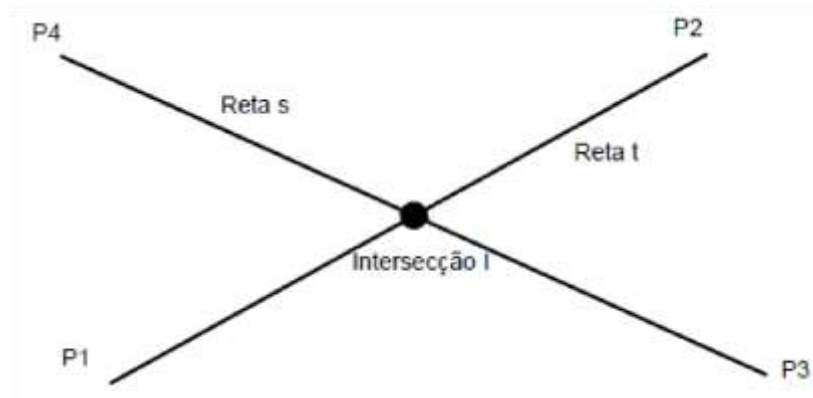


Figura 15 – Intersecção entre duas retas
Fonte: Autor

Sabendo que só existe um ponto de intersecção se as retas forem concorrentes, podemos expressar as retas através da sua equação vetorial: $s: Ps = P1 + ua * (P2 - P1)$; $ua \in \mathbb{R}$ e $t: Pt = P3 + ub * (P4 - P3)$; $ub \in \mathbb{R}$. Para determinar o ponto de intersecção basta resolver o sistema formado pelas suas equações paramétricas.

Equações paramétricas da reta s:

$$\begin{aligned} Ps.x &= P1.x + ua * (P2.x - P1.x) \\ Ps.y &= P1.y + ua * (P2.y - P1.y) \end{aligned}$$

Equações paramétricas da reta t:

$$\begin{aligned} Pt.x &= P3.x + ub * (P4.x - P3.x) \\ Pt.y &= P3.y + ub * (P4.y - P3.y) \end{aligned}$$

Partindo da igualdade $Ps.x = Pt.x$ e $Ps.y = Pt.y$, temos:

$$\begin{aligned} Ps.x &= Pt.x \\ P1.x + ua * (P2.x - P1.x) &= P3.x + ub * (P4.x - P3.x) \\ ua &= \frac{(P3.x - P1.x) + ub * (P4.x - P3.x)}{(P2.x - P1.x)} \\ ub &= \frac{(P1.x - P3.x) + ua * (P2.x - P1.x)}{(P4.x - P3.x)} \end{aligned}$$

$$\begin{aligned} Ps.y &= Pt.y \\ P1.y + ua * (P2.y - P1.y) &= P3.y + ub * (P4.y - P3.y) \\ ua &= \frac{P3.y - P1.y + ub * (P4.y - P3.y)}{(P2.y - P1.y)} \\ ub &= \frac{(P1.y - P3.y) + ua * (P2.y - P1.y)}{(P4.y - P3.y)} \end{aligned}$$

$$\frac{P3.x - P1.x + ub * (P4.x - P3.x)}{(P2.x - P1.x)} = \frac{P3.y - P1.y + ub * (P4.y - P3.y)}{(P2.y - P1.y)}$$

Isolando a variável ub :

$$ub = \frac{(P2.x - P1.x) \cdot (P1.y - P3.y) - (P2.y - P1.y) \cdot (P1.x - P3.x)}{(P4.y - P3.y) \cdot (P2.x - P1.x) - (P4.x - P3.x) \cdot (P2.y - P1.y)}$$

$$\frac{(P1.x - P3.x) + ua * (P2.x - P1.x)}{(P4.x - P3.x)} = \frac{(P1.y - P3.y) + ua * (P2.y - P1.y)}{(P4.y - P3.y)}$$

Isolando a variável ua :

$$ua = \frac{(P4.x - P3.x) \cdot (P1.y - P3.y) - (P4.y - P3.y) \cdot (P1.x - P3.x)}{(P4.y - P3.y) \cdot (P2.x - P1.x) - (P4.x - P3.x) \cdot (P2.y - P1.y)}$$

Logo, o ponto de intersecção entre as reta é determinado por:

$$I.x = P1.x + ua * (P2.x - P1.x)$$

$$I.y = P1.y + ua * (P2.y - P1.y)$$

Algoritmo de Intersecção entre duas retas:

1. Entrar com os valores de entrada: Ponto $p_1(x, y)$, $p_2(x, y)$, $p_3(x, y)$, $p_4(x, y)$;
2. Calcular a expressão do denominador:

$$denom = (P4.y - P3.y) \cdot (P2.x - P1.x) - (P4.x - P3.x) \cdot (P2.y - P1.y)$$

3. Calcular numerador ua e ub :

$$num_a = (P4.x - P3.x) \cdot (P1.y - P3.y) - (P4.y - P3.y) \cdot (P1.x - P3.x)$$

$$num_b = (P2.x - P1.x) \cdot (P1.y - P3.y) - (P2.y - P1.y) \cdot (P1.x - P3.x)$$

4. Calcular ua e ub :

$$ua = \frac{num_a}{denom} \quad ub = \frac{num_b}{denom}$$

5. Caso $denom$ seja '0' não existe intersecção entre elas
6. Caso $denom$ seja diferente de '0' o ponto de intersecção é encontrado através de:

$$Intersection.x = P1.x + ua * (P2.x - P1.x)$$

$$Intersection.y = P1.y + ua * (P2.y - P1.y)$$

4.3.2 INTERSECÇÃO ENTRE UMA RETA E UM CÍRCULO

Para determinar a intersecção entre uma reta e um círculo temos como variável de entrada os dois pontos que formam a reta $P1$, $P2$ e o centro C e raio $Radius$ do círculo. Antes de iniciar a construção do algoritmo veja a figura a seguir que exemplifica o resultado, sendo ele a posição dos círculos pretos:

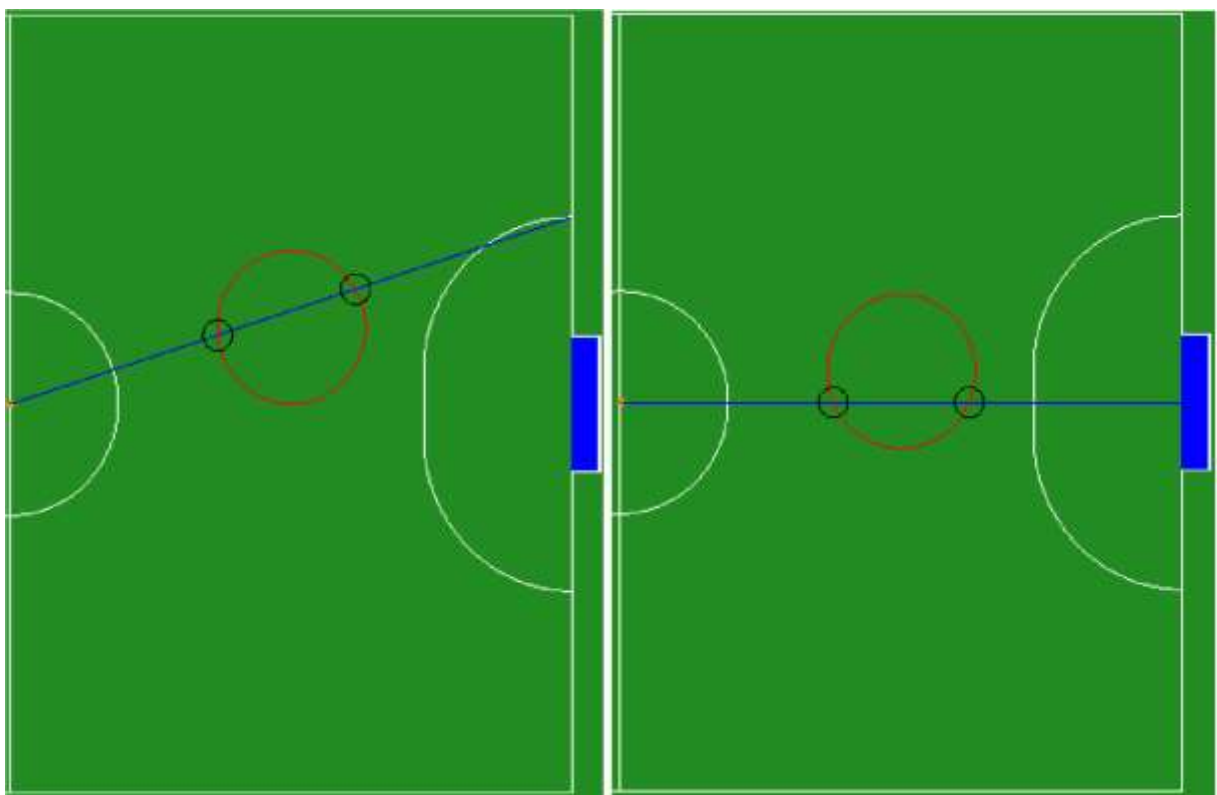


Figura 16 – Exemplos de intersecção entre reta e círculo

Fonte: Autor

Sabendo que dependendo da posição relativa da reta e do círculo, podem não existir intersecção, ou existir apenas um ponto ou dois pontos de intersecção. Este ponto $P(x, y)$ de intersecção esta contido na reta que é formada por dois pontos $P1(x, y)$ e

$P2(x, y)$:

$$\begin{aligned}
 P.x &= P1.x + u * (P2.x - P1.x) \\
 P.y &= P1.y + u * (P2.y - P1.y)
 \end{aligned}$$

O círculo é descrito pela equação da circunferência a seguir:

$$(P.x - C.x)^2 + (P.y - C.y)^2 = radius^2$$

Ao substituir a equação da reta que determina o ponto de intersecção na equação da circunferência, temos uma expressão na forma quadrática:

$$a \cdot u^2 + b \cdot u + c = 0$$

Para determinar a variável u basta apenas utilizar a fórmula de *bhaskara*:

$$u = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

Onde:

$$\begin{aligned} a &= (P2.x - P1.x)^2 + (P2.y - P1.y)^2 \\ b &= 2 \cdot [(P2.x - P1.x) \cdot (P1.x - C.x) + (P2.y - P1.y) \cdot (P1.y - C.y)] \\ c &= C.x^2 + C.y^2 + P1.x^2 + P1.y^2 - 2 \cdot (C.x \cdot P1.x + C.y \cdot P1.y) - r^2 \end{aligned}$$

Algoritmo de Intersecção entre uma reta e um círculo:

1. Entrar com os valores de entrada: Ponto $p_1(x, y)$, $p_2(x, y)$, $C(x, y)$ e *radius*;
2. Calcular os valores das variáveis a , b e c ;
3. Calcular o produto de $b^2 - 4ac$

Caso $b^2 - 4ac < 0$

A reta não intercepta o círculo.

Caso $b^2 - 4ac = 0$

Existe um único ponto de intersecção, pois o valor de $u = \frac{-b}{2a}$, Logo:

Caso $b^2 - 4ac > 0$ Existe dois pontos de intersecção, Logo:

$$u = \frac{-b \pm \sqrt{b^2 - 4ac}}{2a}$$

4. Agora basta determinar o ponto de intersecção P

$$\begin{aligned} P.x &= P1.x + u \cdot (P2.x - P1.x) \\ P.y &= P1.y + u \cdot (P2.y - P1.y) \end{aligned}$$

4.3.3 INTERSECÇÃO ENTRE DOIS CÍRCULOS

Para determinar a intersecção de duas circunferências, primeiro devemos ter a certeza que exista intersecção entre elas. Para isso devemos calcular a distância entre os pontos P_1 e P_0 , no qual chamaremos de “ d ”. Com isso, podemos notar:

- Se $d > |r_0 + r_1|$ não existe intersecção entre os cilindros.
- Se $d < |r_0 - r_1|$ não existem intersecções, porque um círculo está contido dentro do outro.
- Se $d = 0$ e $|r_0 - r_1|$ então os círculos são coincidentes e existe um número infinito de soluções.

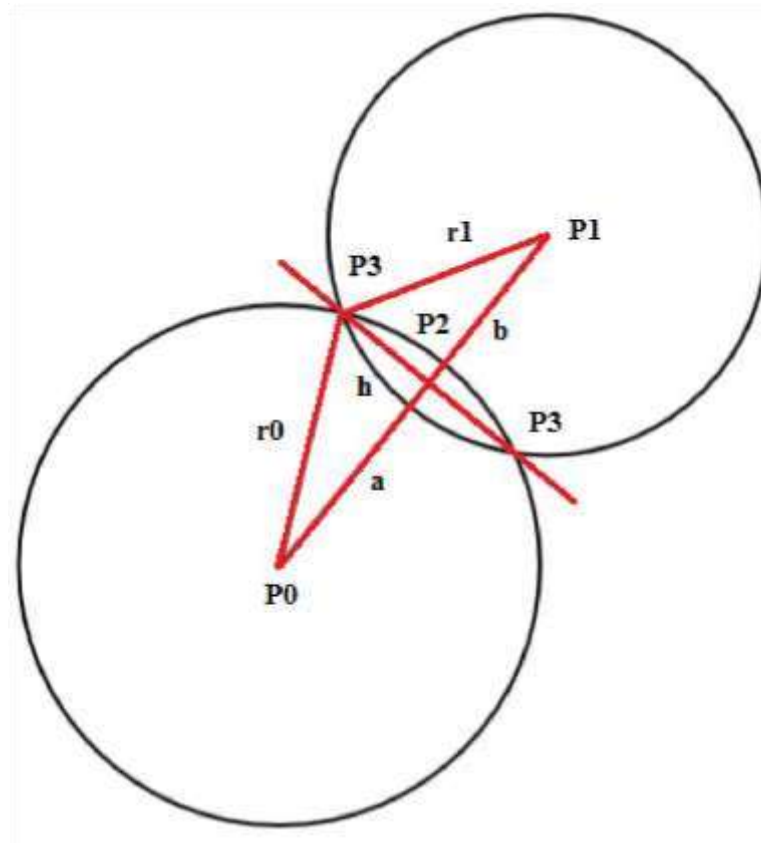


Figura 17 – Intersecção entre dois círculos
 Fonte: Autor

Através dos triângulos $P_0P_2P_3$ e $P_1P_2P_3$ podemos usar o teorema de Pitágoras para relacionar os comprimentos do lado do triângulo. Sendo assim:

$$\begin{aligned}
 a^2 + h^2 &= r_0^2 \\
 b^2 + h^2 &= r_1^2
 \end{aligned}$$

Sabendo que a distância “d” entre os pontos P_0 e P_1 corresponde à soma de “a” e “b”, temos:

$$d = a + b$$

Sabendo que:

$$h = \sqrt{r_0^2 - a^2}$$

$$b = \sqrt{r_1^2 - h^2}$$

Resulta-se em:

$$a = \frac{r_0^2 - r_1^2 + d^2}{2 \cdot d}$$

Portanto, a equação paramétrica de P_2 :

$$P_2 = P_0 + a \cdot (P_1 - P_0) / d$$

E finalmente o ponto P_3 que corresponde à intersecção é:

$$P3.x = P2.x \pm h \cdot (y_1 - y_0) / d$$

$$P3.y = P2.y \mp h \cdot (x_1 - x_0) / d$$

Algoritmo de Intersecção entre dois círculos:

1. Entrar com os valores de entrada: Ponto $C_1(x, y)$, $C_2(x, y)$, $radius_1$ e $radius_2$;

2. Calcular a distancia d:

$$d = |C_2 - C_1|$$

Caso $d = 0$

Existe infinitos pontos de intersecção

Caso $d > (radius_1 + radius_2)$

Não existe ponto de intersecção

Caso $d < (radius_1 + radius_2)$

Existe ponto de intersecção

3. Caso exista ponto de intersecção basta calcular as seguintes expressões:

$$a = \frac{r_0^2 - r_1^2 + d^2}{2 \cdot d}$$

$$h = \sqrt{r_0^2 - a^2}$$

$$P3.x = P2.x \pm h \cdot (y_1 - y_0) / d$$

$$P3.y = P2.y \mp h \cdot (x_1 - x_0) / d$$

4.3.4 PONTO TANGENTE DE UM CÍRCULO

Para determinar um ponto T tangente a uma circunferência que esta contida em uma reta apenas conhecendo a posição relativa da circunferência e outro ponto que também esta contida na reta, precisamos entender como funciona a sua construção geométrica. Para isso, baseando-se em técnicas de construção de figuras geométricas em prancheta, temos que:

- Traçar uma reta unindo o ponto “P” ao centro da circunferência “O”;
- Traçar a mediatriz ao segmento OP, obtendo-se o ponto “M”;
- Traça-se um arco de circunferência centrada em M, passando por O e P.
- Na intersecção deste arco com a circunferência de centro “O” determina-se o ponto “T” que é tangente a circunferência;

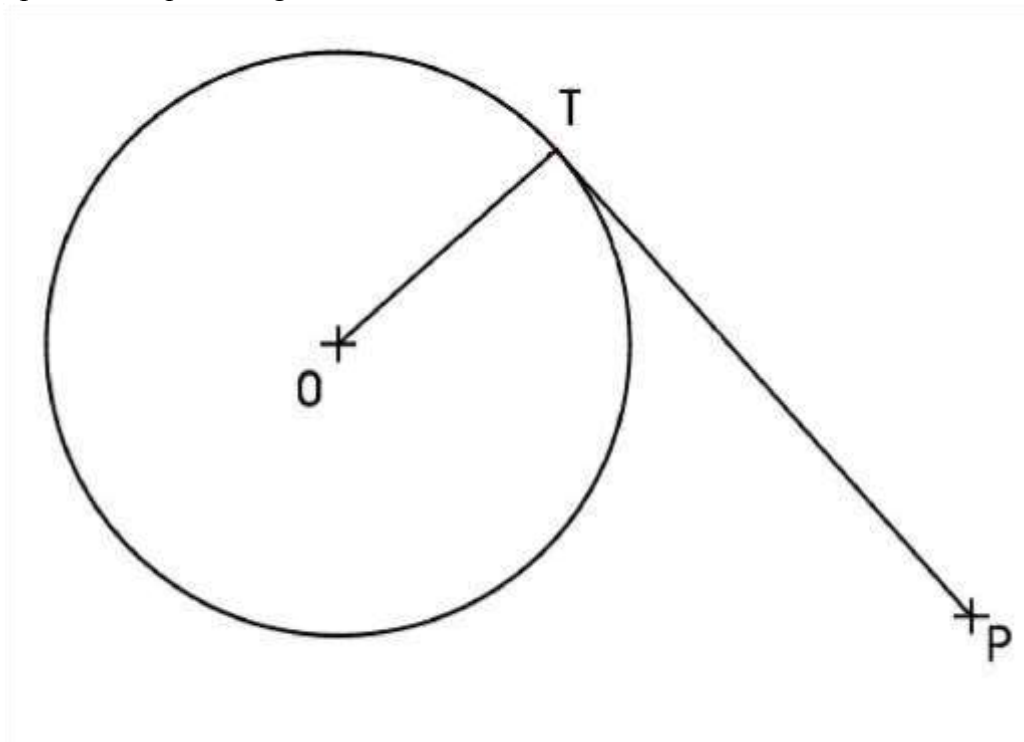


Figura 18 – Ponto tangente num círculo
Fonte: Autor

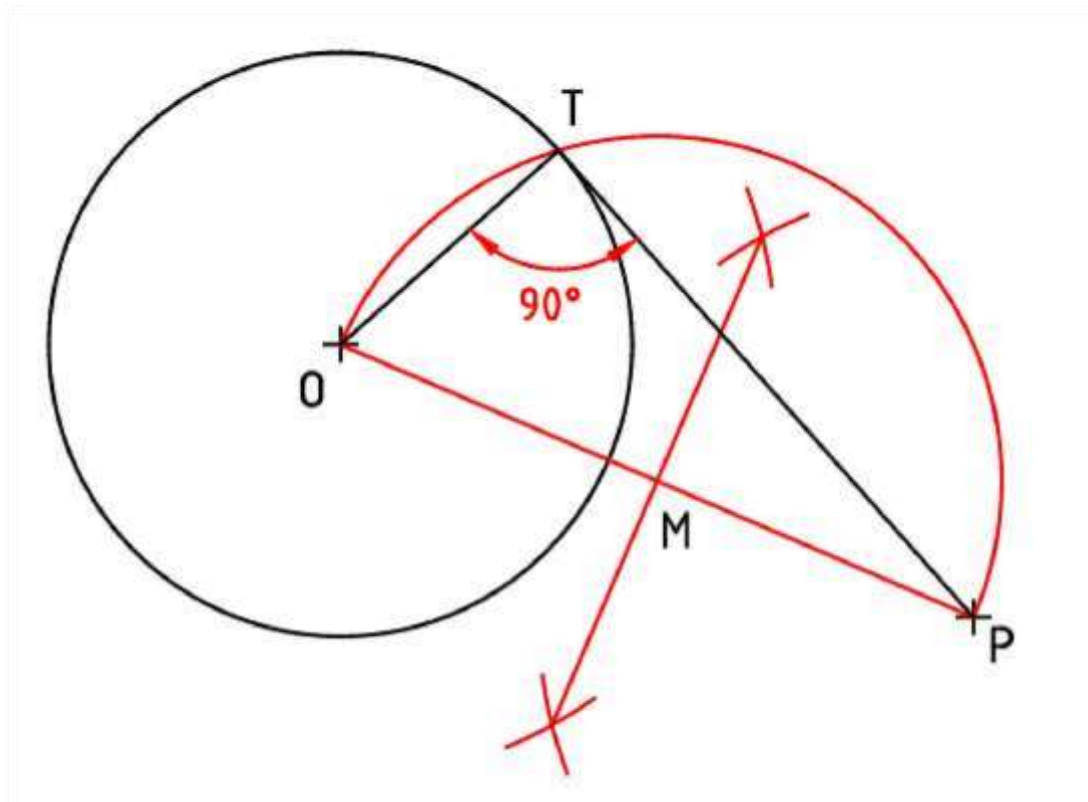


Figura 19 – Método de construção geométrica para ponto tangente num círculos
 Fonte: Autor

Logo, podemos notar que o ponto T é dado pela intersecção de duas circunferências. Sendo a primeira conhecida e a segunda circunferência determinada pelas coordenadas de O e P.

Algoritmo de Intersecção entre dois círculos:

1. Entrar com os valores de entrada: Ponto $C_1(x, y)$, $radius_1$ e $P(x, y)$;
2. Determinar a posição relativa de um círculo auxiliar:

$$C_{aux}(x, y) = \frac{C_1(x, y) + P(x, y)}{2}$$

3. Determinar o raio do círculo auxiliar:

$$radius_{aux} = \frac{|C_{aux}(x, y) - P(x, y)|}{2}$$

4. Chamar a função de intersecção entre dois círculos com os parâmetros de entrada as duas circunferências com suas posições relativas e raios: Ponto

$$C_1(x, y), C_{aux}(x, y), radius_1 \text{ e } radius_{aux}$$

4.4 DETERMINAÇÃO DO MELHOR PONTO PARA EFETUAR O CHUTE AO GOL

O principal objetivo do futebol de robôs é converter o máximo de gols possível, pois o time que efetuar o maior número de gols é a equipe vencedora. Para isso é necessário possuir um bom algoritmo para tal tarefa. É indubitável que o espaço mais aberto do gol é o melhor ponto para chutar. Com isso será apresentado o algoritmo para determinar o melhor ponto para realizar o chute.

O algoritmo *BestPointToShoot* basicamente cria uma reta entre a bola e o ponto tangente a todos os robôs e identificam quais retas interceptam a área do gol. Depois de identificado quais retas interceptam o gol, o algoritmo faz uma varredura da maior distancia entre as retas que não possuem robô.

Veja a figura 20 a seguir. Note que o algoritmo criou 3 retas tangentes aos robôs 1 e 4. Através da varredura do maior intervalo entre as retas obteve o ponto *BestPoint*.



Figura 20 – Escolha do melhor ponto para chutar

Fonte: Autor

Algoritmo *BestPointToShoot*:

1. Guardar em um vetor *step* a coordenada no eixo y dos pontos extremos do gol:

step[0] = Coordenada Y da extremidade superior do gol (+700 mm)
step[1] = Coordenada Y da extremidade inferior do gol (-00 mm)

2. Para todos os robôs em campo determina:

- O ponto que contem uma reta que tangencia o robô e passe pela bola. Para isso utiliza-se o algoritmo *tangentCircle()*;
- Através dos pontos tangentes ao robô guarda a coordenada y do ponto que a reta tangente aos robôs intercepta pela extremidade do campo. Para isso utilizase o algoritmo *IntersectionBetweenLines()*.

3. Ordenam em ordem as coordenadas armazenadas do vetor X;

4. Verifica qual é o maior intervalo entre as coordenadas que não possui nenhum robô, sendo o melhor ponto para realizar o chute no meio deste intervalo.

4.5 PONTO EM QUE A BOLA INTERCEPTA O CAMPO

Para realizar a interceptação da bola é essencial saber qual é a sua trajetória para que o algoritmo tenha eficiência para dominá-la. Logo este algoritmo retorna o ponto em que a bola tende a cruzar os limites do campo.

Para tal tarefa, o algoritmo determina um próximo ponto a partir da velocidade vetorial. Veja a seguir:

angle = $\tan(\text{ângulo da velocidade vetorial da bola})$
nextPoint.x = *posBall.x* + $\cos(\text{angle})$
nextPoint.y = *posBall.y* + $\text{sen}(\text{angle})$

Com esses dois pontos, a posição da bola e o *nextPoint*, podemos definir uma reta e descobrir qual ponto que esta reta intercepta com as bordas do campo utilizando o algoritmo *IntersectionBetweenLines*.

Entretanto, temos 4 bordas do campo, sendo elas as bordas superiores, inferiores, lateral esquerda e lateral direita. Logo deve testar o algoritmo em *IntersectionBetweenLines* todos os lados. Veja a figura 22 que ilustra a trajetória da bola a partir do ponto de intersecção com as bordas do campo. Sendo a trajetória a linha de cor preta e a laranja o rastro de sua trajetória.

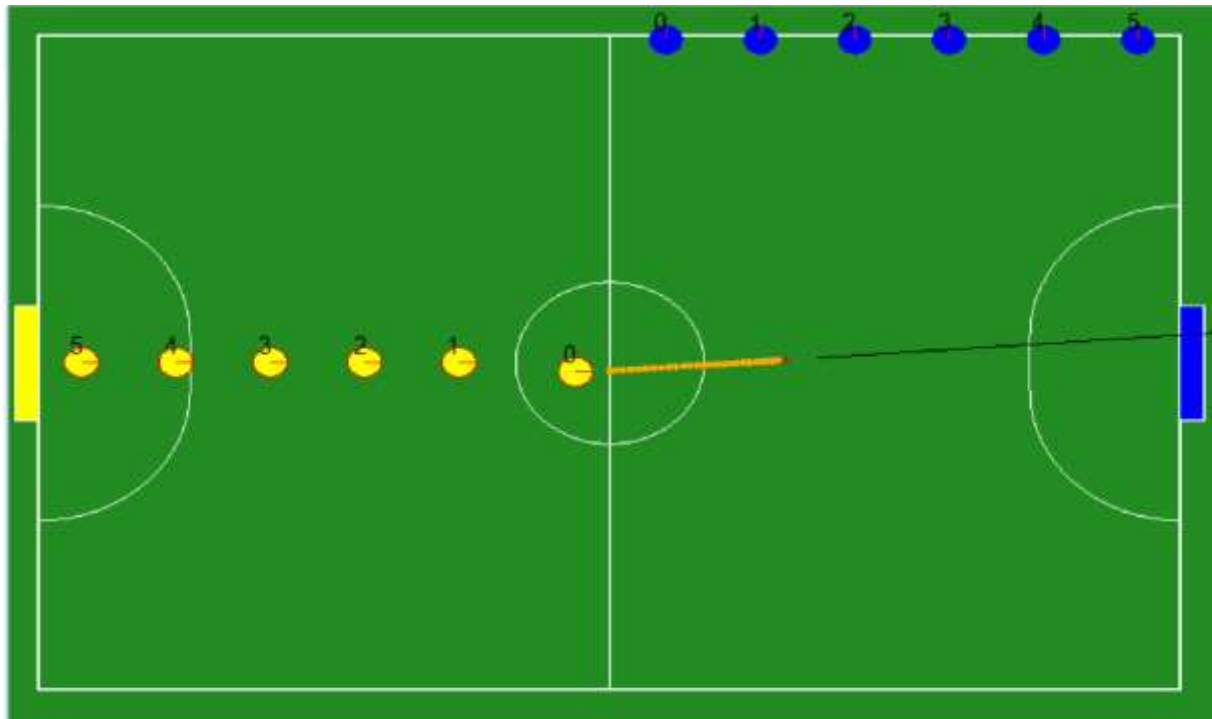


Figura 22 – Trajetória da bola
Fonte: Autor

Algoritmo *BallInterceptBounds*:

1. Entrar com a posição da bola;
2. Determinar o ângulo da velocidade vetorial da bola;

$$angle = \tan\left(\frac{op}{ad}\right)$$
3. Determinar um ponto em que a bola tende a ir através do ângulo da velocidade vetorial;

$$\begin{aligned} nextPoint.x &= posBall.x + \cos(angle) \\ nextPoint.y &= posBall.y + \sin(angle) \end{aligned}$$
4. Determinar a intersecção entre a reta formada pela *posBall* e o *nextPoint* com as linhas laterais do campo.

IntersectionBetweenLines(posBola, nextPoint, Linhas laterais)

4.6 INTERCEPTAR A BOLA

Para que o passe se concretize é necessário que o robô receptor consiga interceptar a bola corretamente, para que possa realizar a próxima ação, como passar a bola para outro robô ou chutar para o gol. Este algoritmo respeita os efeitos dinâmicos do jogo, como a velocidade da bola, capaz de prever qual é o melhor ponto para interceptá-la.

Para o caso em estudo, interceptar a bola significa que o robô deve se mover para certo ponto na direção da bola, e realizar o chute ao gol ou para outro robô quando a bola estiver na posição prevista. Esta tarefa já foi solucionada por outros pesquisadores, com diversos métodos, por exemplo, a equipe *Brainstormers Tribots* da categoria *Middle Size* utiliza o aprendizado de máquina para aperfeiçoar a interceptação da bola.

O algoritmo apresentado neste trabalho, baseia-se no método desenvolvido pela equipe *CMUnited*[24]. Na qual, conhecendo a trajetória da bola, posiciona-se em certo ponto esperando que a bola passe próximo ao robô para efetuar o chute ou domínio.

Inicialmente é determinada a trajetória da bola utilizando o algoritmo *BallInterceptBounds*. Através desta reta o algoritmo faz uma varredura para determinar em qual ponto o robô consegue se aproximar com o tempo suficiente para chutar ou dominar a bola antes que ela ultrapasse. Para isso a trajetória da bola é dividida em pequenos intervalos e feita uma verificação para qual destes pontos o robô deve interceptar a bola. Veja a figura que exemplifica esta tarefa.

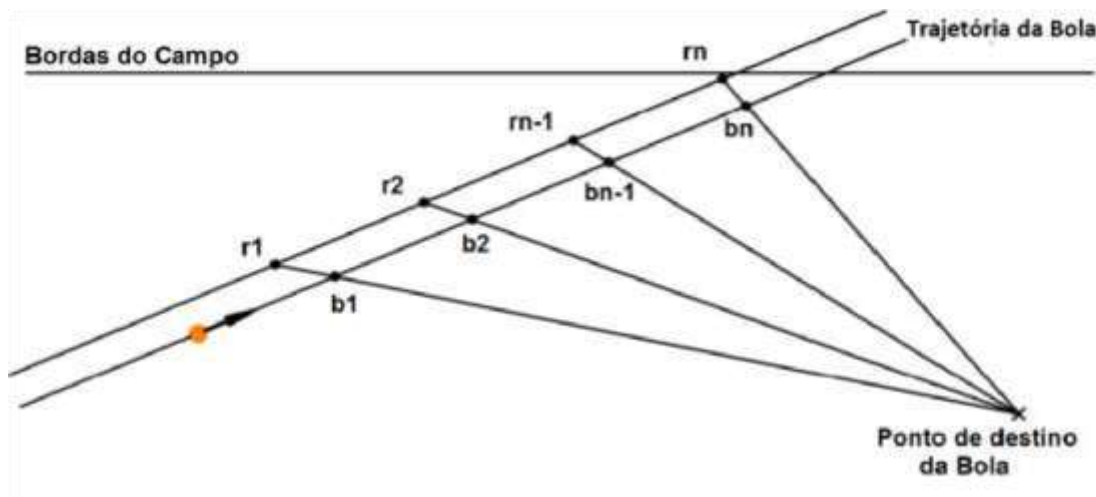


Figura 23 – Interceptação da bola

Fonte: Autor

Para que o robô consiga interceptar, ele deve escolher qual o ponto em que ele é capaz de chegar antes da bola. Com a trajetória dividida com uma distância fixa é determinado uma reta paralela a ela com uma distância de um raio de robô. Depois disso é calculado o tempo em que o robô leva para chegar ao ponto r e o tempo em a bola leva para chegar ao ponto b . Logo quando o tempo do robô for menor que o tempo da bola será o local ideal para se posicionar. Em seguida, quando o robô chegar à posição desejada ele intercepta a bola quando ela estiver entre o robô e o ponto de destino da bola.

4.7 CONSTRUÇÃO DO ALGORITMO DE PASSE

O algoritmo para o passe simples, consiste em grupo de funções, que serão explicadas mais adiante. Inicialmente temos a função principal que executa o passe. Ela está situada na *skill PassToRobot()*.

Esta *skill PassToRobot()* é dividida em três camadas principais, sendo elas:

CalcFitness(): Calcula a probabilidade de sucesso de esta *skill* ser executada corretamente;

Plan(): Realiza o planejamento desta *skill*;

Execute(): Executa os comandos que a *skill* planejou.

Nesta etapa do projeto, foi realizada a construção do algoritmo do *execute()*, assumindo o destino do passe o robô mais próximo. Veja na figura 24 o fluxograma desta função:

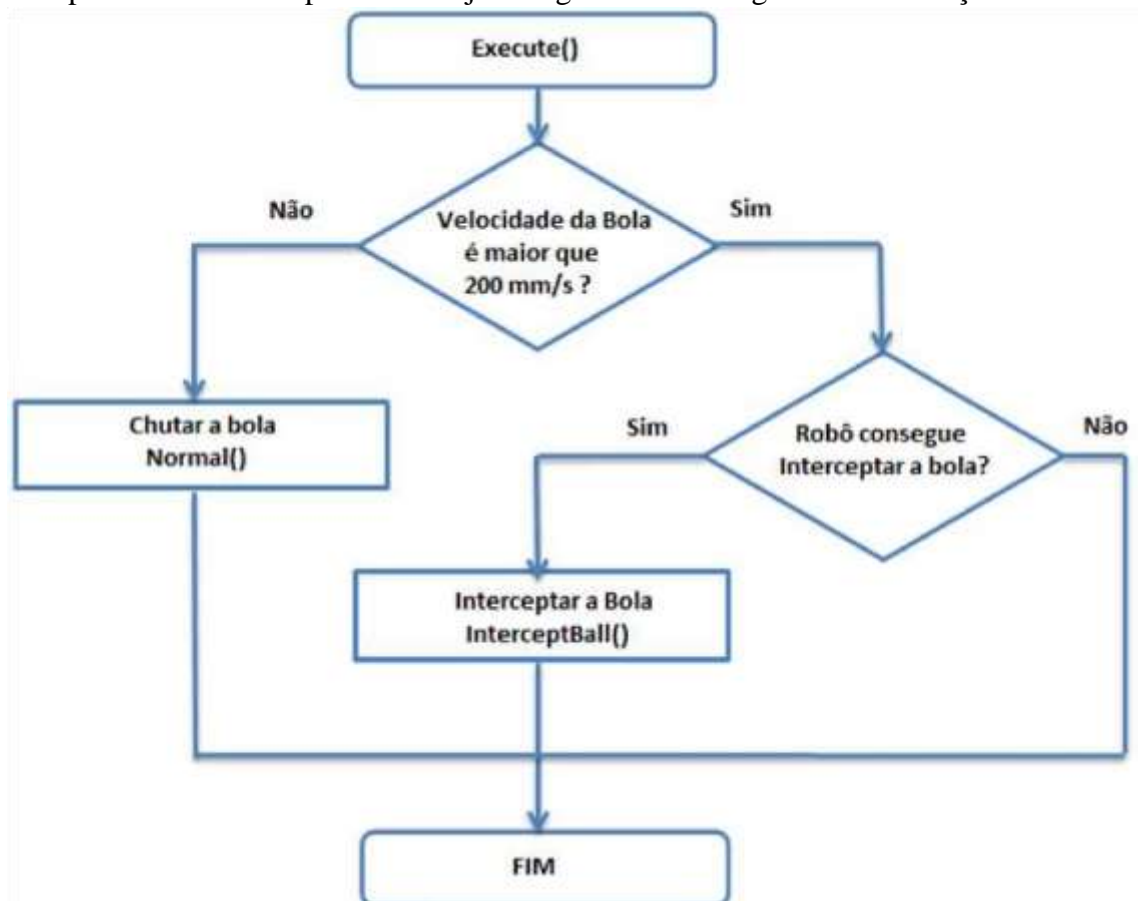


Figura 24 – Fluxograma da função *Execute()* do *PassToRobot()*

Fonte: Autor

Primeiramente, a função *execute()* verifica se a velocidade da bola esta com velocidade superior a 0,2 m/s. Quando esta condição for verdadeira , é verificado se o robô consegue interceptar a bola ou não, e interceptará de acordo com a resposta desta condição. No entanto se a velocidade da bola for inferior a 0,2 m/s o robô irá até a bola realizar o chute. Para verificar se o robô consegue interceptar a bola, primeiramente determina o ponto P2 através da trajetória da bola e seu ponto de intersecção com as bordas do campo (Veja na Figura 16). Sabendo os pontos P2, posição da bola (P1) e a posição do robô que irá interceptar a bola, utiliza o algoritmo de PAUL BOURKES [17] para determinar o ponto T, que é a posição mais próxima que o robô consegue interceptar.

Em seguida, verifica o tempo que a bola leva para chegar ao ponto T e o tempo que o robô leva até chegar ao mesmo ponto. Caso o robô possuir um tempo menor que o tempo da bola, conclui-se que o robô consegue interceptar a bola.

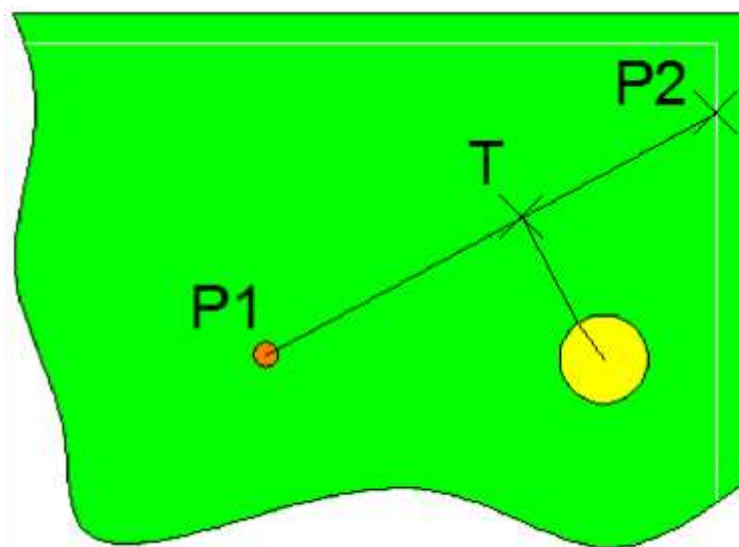


Figura 25 – Intercepção da bola
Fonte: Autor

Robô consegue interceptar a bola?

1. As variáveis de entrada são:
P1 : Posição da Bola
P2 : Ponto situado nas bordas do campo, na direção que a bola esta se deslocando.
robotPos : Posição do robô.
2. Calcular o ponto p2 através da função *LinhaDoCampoEncontraBola()*.
3. Determinar a distancia da posição do robô e o ponto T, através da função *GetPointLineIntercept()*, que é baseado no algoritmo de PAUL BOURKES [17] e suas variáveis de entrada são os pontos: P1, P2, robotPos;

4. Determinar o tempo que o robô leva para chegar ao ponto T. Assumindo a velocidade máxima do robô (*Max.Veloc.Robo*).

$$tempo1 = \frac{|t-robotPos|}{Max.Veloc.Robo}$$

5. Determinar o tempo que a bola leva para chegar ao ponto T. Sabendo que *Vel.Bola* é a velocidade atual da bola.

$$tempo2 = \frac{|t-P1|}{Vel.Bola}$$

6. Calcular a variação (*delta*) do tempo que a bola e o robô levam para chegar ao ponto T.

$$delta = tempo1 - tempo2$$

7. Se $delta < 0$ o robô consegue interceptar a bola. Caso contrário o robô não consegue interceptar a bola.

Caso o robô consiga interceptar a bola e a mesma estiver a uma distância superior a 160 mm do ponto T, ele irá se posicionar a uma distância equivalente de 1 robô (180 mm) do ponto T, com a frente do robô em direção ao ponto de destino, para interceptar a bola. Caso contrário, ele irá chutar a bola até o ponto planejado. Veja o algoritmo a seguir:

Interceptação da Bola

1. As variáveis de entrada são:

P1 : Posição da Bola

robotPos : Posição do robô

DestBall : Ponto onde o robô pretende chutar a bola (Este ponto é determinado pela função *plan()*)

T : Ponto situado na trajetória da bola, sendo ele o ponto mais próximo até o robô.

2. Determinar o ângulo entre os pontos *DestBall* e *T*

$$angle = \frac{(T \rightarrow y) - (robotPos \rightarrow y)}{(T \rightarrow x) - (robotPos \rightarrow x)}$$

3. Determinar o módulo da distancia entre os pontos *P1* e *T*:

$$dist = |P1 - T|$$

4. Se $\text{dist} > 160 \text{ mm}$:

O

P robô deve ir até uma distância de 11,15 cm da bola, com o ângulo correspondente a variável *angle*. Esta distância é equivalente à soma dos raios da bola e do robô.

$$\text{destino do robo} \rightarrow x = (T \rightarrow x) - \cos(\text{angle}) \cdot 11.15$$

$$\text{destino do robo} \rightarrow y = (T \rightarrow y) - \sin(\text{angle}) \cdot 11.15$$

5. Se $\text{dist} < 160 \text{ cm}$:

O robô deve ir até a posição da bola, com o ângulo correspondente a variável *angle*.

$$\text{destino do robo} = P1$$

5. EXPERIMENTOS

Durante a construção do algoritmo para efetuar o passe horizontal, foi realizado testes no simulador PhysX_Sim [12], para comprovar o funcionamento correto dos algoritmos apresentado. No entanto, para obter a comprovação real, é necessário realizar os testes nos próprios robôs.

Neste relatório final está sendo apresentado o passe para um único robô estático, com o intuito de verificar a precisão da trajetória da bola. Veja a figura 26, que a bola passa cerca de 2 cm do robô. Esta diferença é relativamente aceitável, pois a ação de passar a bola, também depende do receptor, pois ele irá se ajustar para receptor a bola.



Figura 26 – Passe Simples para um robô estático
Fonte: Autor

Através deste foi obtido um total de 10 amostras, onde foram separados em três grupos: Passe convertido com sucesso, onde a bola atinge o robô; passe admissível, onde a bola possui uma distância até o robô inferior a 15 cm, e passe não convertido.

Situação do Passe	Quantidade
Passe Convertido	1
Passe Admissível	4
Passe não Convertido	5

Tabela 2 – Resultado obtido no experimento

Fonte: Autor

Através deste levantamento, vimos que o algoritmo requer uma calibração para efetuar o chute com mais precisão, pois a proporção de passe não convertido é maior que o passe convertido mais o passe admissível. Com isso, ressalta a importância de possuir um bom algoritmo para interceptar a bola.

Outro experimento realizado foi à execução do algoritmo para interceptar e realizar o chute ao gol. Veja nas figuras 26 e 27, ilustra o funcionamento deste algoritmo em duas direções distintas da bola. Para realizar esse experimento foi dado um impulso na bola para que ela fosse atrás e conseguisse interceptar e chutar ao gol.



Figura 27 – Intercepção da bola

Fonte: Autor



Figura 28 – Intercepção da bola
Fonte: Autor

6. CONCLUSÃO

A partir da revisão no funcionamento dos softwares do ROBOFEI [1], sendo eles: Futbots Strategy, SSL-LogPlayer 1.1, PhysX_Sim[12], foi possível realizar a inserção de novas funções e modificações de funções existentes.

Inicialmente foram realizadas análises da velocidade da bola, permitindo normalizar as intensidades do chute durante o jogo, de acordo com as regras da competição para máxima velocidade que a bola deve atingir. Foi feita a seleção do nível de intensidade de chute levando em consideração a possível intervenção do oponente e a distância em que o robô pretende lançar a bola.

As funções matemáticas e trigonométricas auxiliaram para a construção dos algoritmos de interceptação e chute da bola. Entretanto são algoritmos modulares, que foram criados não somente para as funções de interceptação, mas sim para qualquer outra atividade.

A interceptação da bola com precisão foi outro tópico importante para a conclusão deste trabalho, pois não basta apenas efetuar o chute para um robô sabendo que ele não é capaz de receber a bola. Portanto, a partir dos algoritmos apresentados foi possível criar uma função que consiga interceptar a bola.

7. REFERÊNCIAS BIBLIOGRÁFICAS

- [1] - Gurzoni, José Angelo; MARTINS, Murilo Fernandes; Tonidandel, Flavio; Bianchi, Reinaldo A. C. . *On the construction of a RoboCup small size league team. Journal of the Brazilian Computer Society* (Impresso), v. 17, p. 69-82, 2011.
- [2] - “Small Size Robot League”. Disponível em: <<http://small-size.informatik.unibremen.de/>>. Acesso em: 14 Set. 12.
- [3] - “RoboCup Official Site” . Disponível em: <<http://www.robocup.org>>. Acesso em: 14 Set. 12.
- [4] – “Competição Brasileira de Robótica”. Disponível em: <<http://www.cbr2011.org/>>. Acesso em: 14 Set. 12.
- [5] – “Federation of International Robosoccer Association”. Disponível em: <<http://www.fira.net/?mid=mirosot> >. Acesso em: 14 Set. 12.
- [6] – “6º SBAI (Simpósio Brasileiro de Automação Inteligente)”. Disponível em: <<http://www.visbai.feb.unesp.br/>>. Acesso em: 14 Set. 12.
- [7] - “The Futepoli Team Homepage”. Disponível em: <<http://www.lti.pcs.usp.br/robotics/futepoli>>. Acesso em: 14 Set. 12.
- [8] - Guaraná COSTA, A. H.R.; PEGORARO, R. **Construindo Robôs Autônomos para Partidas de Futebol: O time Guaraná**. Controle e Automação SBA, v.11,n.2, p.141149., 2000.
- [9] – HALLIDAY, DAVID; RESNICK, ROBERT; KRANE, KENNETH S. – *Física 3*
–
4ª edição - Livros Técnicos e Científicos Editora S/A (LTC) – Rio de Janeiro - 1996
- [10] – SOARES, R. A. *Conversão eletromecânica de energia*. SBC: Universitária Leopoldianum, 2008.
- [11] – VELOSO, M. et al. *SSL-Vision: The Shared Vision System for the RoboCup Small Size League*. SBC: Universitária Leopoldianum, 2008.
- [12] – “The manual for installing, configuring, and using SSL-Vision”. Disponível em: <<http://code.google.com/p/ssl-vision/wiki/Manual>>. Acesso em: 14 Set. 12.

- [13] – WELCH, G. BISHOP, G. “**An introduction to the Kalman Filter**”. Disponível em: <<http://www.cs.unc.edu/~welch/kalman/kalmanIntro.html>>. Acesso em: 14 Set. 12. [14] – VELOSO, M. et al. **CMDragons: Dynamic Passing and Strategy on a Champion Robot Soccer Team**. Disponível em: <<http://www.cs.cmu.edu/~robosoccer/small/>>. Acesso em: 14 Set. 12.
- [15] – VELOSO, M. et al.. **Cooperative 3-robot passing and shooting in the RoboCup Small Size League**. In Proceedings of the RoboCup Symposium 2006, Bremen, Germany, June 2006.
- [16] – “**Classe File**”. Disponível em: <<http://msdn.microsoft.com/ptbr/library/system.io.file.aspx>>. Acesso em: 14 Set. 12
- [17] – BOURKE, Paul. “Geometry, Surfaces, Curves, Polyhedra”. Disponível em: <<http://paulbourke.net/geometry/>>. Acesso em: 14 Set. 12
- [18] – SANTOS, André O. “SIMULADOR DE FÍSICA APLICADO AO FUTEBOL DE ROBÔS”. Disponível em: <<http://www.fei.edu.br>>. Acesso em: 14 Set.12
- [19] – KYRYLOV, Vadim. “**Balancing Gains, Risks, Costs, and Real-Time Constraints in the Ball Passing Algorithm for the Robotic Soccer**”. ROBOCUP SOCCER WORLD CUP. Lecture Notes in Computer Science, 2007, Volume 4434/2007, 304-313, DOI: 10.1007/978-3-540-74024-7_27
- [20] – STONE, McAllester. “**An Architecture for Action Selection in Robotic Soccer**”. AGENTS '01 Proceedings of the fifth international conference on Autonomous agentes. Pages 316 – 323.
- [21] – “Time Skuba”. Disponível em: <<http://iml.cpe.ku.ac.th/skuba/>>. Acesso em: 14 Set. 12
- [22] – KIM, TAEONE, SEO, YONGDUEK, HONG, KI-SANG. “**Physics-based 3D Position Analysis of a Soccer Ball from Monocular Image Sequences**”. Sixth International Conference on Computer Vision (1998). Pages 721–726.
- [23] – JAMILSON, MARCONE. “**Ajuste de Curvas pelo Método dos Quadrados Mínimos**”. Departamento de Computação, Instituto de Ciências Exatas e Biológicas. Homepage: <http://www.decom.ufop.br/prof/marcone>.
- [24] – VELOSO, M. et al.. **Prediction, Behaviors, and Collaboration of Robotic Soccer Agents**. In Proceedings of the International Conference on Multi-Agent Systems (ICMAS'98).

8. ANEXOS

8.1 ARTIGOS PUBLICADOS

1. Santos, E. G. e Tonidandel, F. SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS. In: Resumos do 14º. Simpósio ICT – FATEC –SP – Outubro de 2012

SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS

Erivelton Gualter dos Santos¹, Flavio Tonidandel
Centro Universitário da FEI
erivelton.gualter@gmail.com, flaviot@fei.edu.br

Resumo: Este trabalho tem o objetivo de aprimorar a estratégia de jogo da equipe ROBOFEI [1], através da inserção de um sistema de passe. Para isso, está sendo realizada análise de alguns fatores importantes, como o controle de chute, recepção da bola e uma seleção para determinar qual o melhor robô para efetuar e receber o passe.

1. Introdução

O projeto de Futebol de Robôs no Centro Universitário da FEI, conhecido como ROBOFEI [1], é utilizado como plataforma de pesquisa na área de robótica e inteligência artificial.

As tarefas a serem executadas pelos robôs são similares aos jogos de futebol, cujo objetivo é realizar o gol. Com isso é necessário o auxílio de um sistema de visão computacional, onde é realizado o filtro das posições do robô e da bola dentro de campo.

O robô possui um sistema de chute e um mecanismo de manipulação da bola, onde é possível efetuar dois tipos de chute, o primeiro cuja trajetória é uma parábola e o outro, a bola percorre uma trajetória linear.

2. Metodologia

Para realização do projeto, primeiramente foi realizado a extração de dados no momento que o robô efetua o chute. Nesta tarefa, foram obtidos as coordenadas da bola no campo e o tempo neste determinado instante.

Atualmente, o robô possui 15 níveis de intensidade de chute. Com a análise dos dados, foi possível determinar a velocidade instantânea em cada nível, onde posteriormente foram inseridas num gráfico (Figura 1), sendo o eixo das ordenadas a velocidade da bola capturada ao longo do tempo, que é o eixo da abscissa.

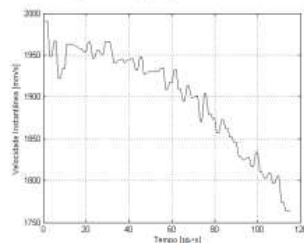


Figura 1 – Gráfico Velocidade x Tempo.

Conhecer a velocidade da bola durante o jogo é importante, pois é necessário saber a força que deve ser aplicada para o robô receber a bola, sem intervenção de um robô oponente.

Para determinar a velocidade necessária do passe, para que o robô oponente não intercepte a bola, será utilizado o algoritmo de PAUL BOURKE [2]. Esta solução determina a interseção (ponto I) da reta formada entre a bola e o robô receptor, com a linha entre o robô oponente a esta reta (Figura 2).

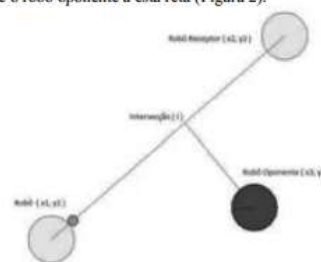


Figura 2 – Diagrama da Intervenção do robô oponente

Para que não ocorra a interrupção do passe, a bola deve passar pelo ponto I antes do robô oponente. Através da equação 1 temos o tempo (t1) que o robô oponente leva para chegar até este ponto. Onde Δs é a distância entre o robô oponente e o ponto I, e Vm é a velocidade máxima do robô oponente.

$$t1 = \frac{\Delta s}{V_m} \quad (1)$$

Logo, o tempo (t2) que a bola percorre da sua origem até o ponto I deve ser maior que o tempo t1 (t2 > t1). Com essa avaliação, podemos determinar a velocidade da bola (Vb), onde ΔS é a distância entre a bola e o ponto I. Veja a equação 2.

$$V_b = \frac{\Delta S}{t_1} \quad (2)$$

4. Conclusões

Com a gravação de dados iniciais, resultaram na análise das velocidades que a bola pode percorrer o campo, respeitando a interceptação de robôs oponente. Porém ainda deve ser elaborada, a escolha de qual robô efetuar o passe, através de análises probabilísticas e realizar o recebimento da bola corretamente, para que resulte em maiores chances de finalizações ao gol.

5. Referências

- [1] - "RoboFEI". Disponível em: <http://www.fei.edu.br/robo>. Acesso em: 13 Ago. 2012
- [2] - "Minimum Distance between a Point and a Line". Disponível em: <http://paulbourke.net/geometry/pointline/>. Acesso em: 14/Ago/2012

Agradecimentos

¹ Aluno de IC do Centro Universitário da FEI

2. Santos, E. G. e Tonidandel, F. SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS. In: Simpósio de Pesquisa do Grande ABC - SGABC – São Bernardo do Campo - SP – Novembro de 2012

SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS

SANTOS, E. G. and TONIDANDEL, Flavio
Centro Universitário da FEI
estrelas.gustavo@gmail.com, flavio@fei.edu.br
Extensão Científica do Seso, Flávio Tonidandel

Resumo: Este trabalho tem o objetivo de apresentar a estratégia de jogo da equipe ROBOPÊ [1], através da descrição de um sistema de passe. Para isso, está sendo realizada análise de alguns fatores importantes, como o controle de chute, recepção da bola e uma seleção para determinar qual o melhor robô para efetuar o passe e receber o passe.

Palavras-chave: Futebol de robô, Velocidade, Intercepção.

1. Introdução

O projeto de Futebol de Robô no Centro Universitário da FEI, conhecido como ROBOPÊ [1], é utilizado como paradigma de pesquisa na área de robótica e inteligência artificial.

Os robôs utilizados neste projeto são robôs semiautômatos e obedecem às regras da categoria Small Size [2], com cerca de 140 mm de altura física, com uma altura que não deve ultrapassar a 150 mm e um diâmetro de 110 mm.

As partidas são jogadas em campos de robô, são similares aos jogos de futebol, cujo objetivo é realizar o gol. Com isso é necessário o auxílio de um sistema de visão computacional, chamado SSL-Vision, onde é realizado o fluxo das posições do robô e da bola dentro do campo.

O robô possui um sistema de chute e um mecanismo de manipulação da bola, onde é possível efetuar dois tipos de chute, o primeiro cuja trajetória é parabólica e o outro, a bola percorre uma trajetória linear.

2. Material e Métodos

O dispositivo de chute horizontal (veja a Fig.1) é composto por um cilindro de 30 mm de diâmetro, feito de Nylon, acoplado com fita de corte adesivo AWO-12. Seu fecho contém 14 mm de diâmetro feito de aço SAE1020.

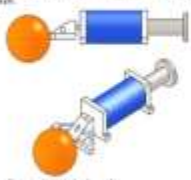


Fig. 1 - Sistema de chute horizontal

Ele é um dispositivo eletromecânico com estrutura simples. Os eixos, basicamente, é uma bolina alimentada por uma corrente elétrica, com uma única unidade elétrica própria e sem cabos, não existe o efeito da mesma instantânea. Ela é denominada por silêncio.

Até o sistema de visão computacional, SSL-Vision, o robô recebe controle através da equipe ROBOPÊ [1] para geração de dados, no possível realizar a recepção de algumas variáveis no momento que o robô atinge o chute. Foram obtidas a posição da bola e o tempo de simulação da recepção. Com esse conhecimento é possível determinar os efeitos físicos da bola dentro o jogo, como a velocidade que a bola atinge em cada instantâneo de chute.

O SSL-Vision fornece as posições da bola e dos robôs assim como suas orientações. Porém, durante a leitura e transmissão de dados, pode haver ruído na detecção das posições e orientação dos objetos. Para evitar possíveis erros, foi utilizado um Filtro de Kalman para fazer tracking da bola e dos robôs. Depois de obtidas as posições dentro da bola, foi determinada a velocidade que a bola atinge em cada instantâneo, para isso foi calculado as composições variáveis da velocidade X e Y , e depois a média da velocidade. Veja as equações a seguir:

$$Vel_x = \frac{(Pos_{bola X} - x) - (Pos_{bola I} - x)}{Tempo F - Tempo I}$$

$$Vel_y = \frac{(Pos_{bola Y} - y) - (Pos_{bola I} - y)}{Tempo F - Tempo I}$$

Velocidade = $\sqrt{Vel_x^2 + Vel_y^2}$

Conhecendo a velocidade da bola dentro o jogo é importante, pois é necessário saber a força que deve ser aplicada para o robô receber a bola, sem interferência de um robô oponente.

Para determinar a velocidade necessária do passe, para que o robô oponente não atrapalhe a bola, será utilizado o algoritmo de FAULT, BOUNKE [2]. Essa solução determina a intercepção (ponto 1) da reta

formada entre a bola e o robô oponente, com a linha entre o robô oponente a esta reta (Fig. 2).

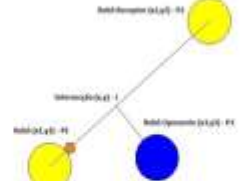


Fig. 2 - Diagrama de intercepção de robô oponente

O algoritmo de FAULT, BOUNKE [2] utiliza o segmento de reta entre os dois pontos $P_1(x_1, y_1)$ e $P_2(x_2, y_2)$:

$$P = P_1 + u(P_2 - P_1)$$

O robô oponente $P(x_2, y_2)$ está mais próximo do segmento de reta, portanto ao ponto 1, onde a reta formada entre o ponto P_1 e o robô oponente P_2 é perpendicular. Sabendo que as retas entre os pontos P_1-P_2 e P_1-P são ortogonais, logo o produto escalar entre elas é zero:

$$(P_2 - P_1) \cdot (P - P_1) = 0$$

Substituindo a equação de reta P , temos:

$$(P_2 - P_1) \cdot (P_1 + u(P_2 - P_1) - P_1) = 0$$

Resolvendo a equação acima e isolando u a resposta em:

$$u = \frac{(x_2 - x_1)(x_2 - x_1) + (y_2 - y_1)(y_2 - y_1)}{\|P_2 - P_1\|^2}$$

Logo, para determinar o ponto 1, deve-se substituir a expressão de u na equação da reta P (equação 1), resultando nas seguintes equações paramétricas, onde a coordenada (x, y) corresponde ao ponto 1:

$$x = x_1 + u(x_2 - x_1)$$

$$y = y_1 + u(y_2 - y_1)$$

Para que não ocorra a intercepção do passe, a bola deve passar pelo ponto 1 (x, y) antes do robô oponente. Através da equação 7 temos o tempo (t) que o robô oponente leva para chegar até este ponto. Onde d_1 é a distância entre o robô oponente e o ponto 1, e V_{max} é a velocidade máxima do robô oponente.

$$t_1 = \frac{d_1}{V_{max}}$$

Logo, o tempo (t_2) que a bola percorre de uma origem até o ponto 1 deve ser menor que o tempo t_1 (Equação 8). Com essa análise, podemos determinar a velocidade da bola (V_b), onde V_b é a distância entre a bola e o ponto 1. Veja a equação 8:

$$V_b \geq \frac{d_1}{t_2}$$

3. Resultados

Com a obtenção dos dados da posição da bola durante o chute, foi possível determinar os dados, onde foi identificada a velocidade que a bola atinge em cada instantâneo de chute. Veja a tabela 1 os dados obtidos da velocidade em cada instantâneo de chute.

Nível de intensidade de Chute	Velocidade Obtida (m/s)
1	0,1
2	0,2
3	0,3
4	0,4
5	0,5
6	0,6
7	0,7
8	0,8
9	0,9

Tabela 1 - Resultado da análise de velocidade

Com esses dados regularizados, é possível realizar o chute com a velocidade necessária através da análise do tempo que o oponente leva para chegar a bola.

Na análise de robô oponente suas proximidades da linha de passe, não é necessário utilizar o algoritmo apresentado para determinar a velocidade de chute. Para isso, foi realizada força aplicada na bola em função da distância supramencionada e plotada no gráfico a seguir:




Gráfico 1 - Intensidade de Chute e Decremento

3. Conclusões

Com a geração de dados locais, realizou-se na análise das velocidades que a bola pode percorrer o campo, restando a intercepção de robô oponente. Porém ainda deve ser elaborada a análise de qual robô efetuar o passe, através de análises probabilísticas e realizar o recebimento da bola corretamente, para que não ocorra a perda de finalização ao gol.


4. Referências

[1] - "RoboPê". Disponível em: <http://www.fei.edu.br/robo>. Acesso em: 12 Ago. 2012.

[2] - "Algorithm Distance between a Point and a Line". Disponível em: <http://www.fei.edu.br/robo/algorithm>. Acesso em: 06 Out. 2012.


[3] - "Small Size Robot League". Disponível em: <http://smallsizeleague.com/league/>. Acesso em: 14 Set. 12.

3. Santos, E. G. e Tonidandel, F. SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS. In: Simpósio de Iniciação Científica - SICFEI – Centro Universitário da FEI – São Bernardo do Campo - SP – Outubro de 2012



SISTEMA DE PASSE PARA O FUTEBOL DE ROBÔS

Erivelton Gualter dos Santos, Flavio Tonidandel
erivelton.gualter@gmail.com, flavio.t@fei.edu.br
Departamento da Ciência da Computação



Objetivo

Este trabalho tem o objetivo de aprimorar a estratégia de jogo da equipe ROBOFEI, através da inserção de um sistema de passe. Para isso, está sendo realizada análise de alguns fatores importantes: como o controle de chute, recepção da bola e um sistema de seleção para determinar qual o melhor robô para efetuar e receber o passe.

Introdução Teórica

- ❖ **Sistema de chute:** É composto por um solenoide de 30 mm de diâmetro, feito de Nylon e enrolado com fio de cobre esmaltado.

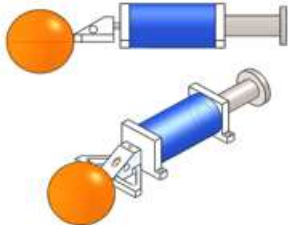


Figura 1 - Sistema de chute horizontal

- ❖ **Funcionamento do chute:** No instante em que ocorre a excitação de corrente elétrica no solenoide é criado um campo magnético, resultando no deslocamento do seu êmbolo.
- ❖ **Intensidade de Chute:** Foi dividido em 15 diferentes níveis de intensidade do chute.

Metodologia

- ❖ **Gravação de dados:** O objetivo deste experimento é obter as posições do robô e da bola através da gravação de dados de uma simulação real.
- ❖ **Análise dos dados:** Depois de realizado a gravação de dados, foi realizada a análise gráfica desses resultados, obtendo a velocidade que a bola atinge em cada intensidade do chute.
- ❖ **Determinar a velocidade da bola:** Com as velocidades do chute normalizada, agora basta identificar qual a velocidade a bola deve atingir.

Resultados

- ❖ Depois de obtidos as posições da bola e o tempo de captura da mesma, foi possível determinar a velocidade que a bola atinge.

Agradecimentos:

- Ao Centro Universitário da FEI pelo patrocínio do projeto e concessão de bolsa de iniciação científica ao aluno Erivelton Gualter dos Santos. E a equipe ROBOFEI pelo auxílio nas tarefas realizadas.

Resultado da Análise:

Nível de intensidade de Chute	Velocidade Obtida [m/s]
2	0,3
3	2
4	4
5	5
6	6
7	7,8
8	8,1
9	8,5

Figura 1 - Sistema de chute horizontal

- ❖ **Tempo de Intercepção da bola:** O tempo em que a bola deve passar pelo ponto I, deve ser maior que o tempo que o robô oponente leva para alcançar o mesmo ponto.

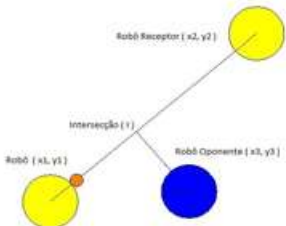


Figura 1 - Sistema de chute horizontal

- ❖ Tempo em que o robô oponente leva para chegar no ponto I:
$$T_{oponente} = \frac{|\text{robo oponente} - \text{ponto I}|}{\text{Velocidade Máxima}}$$
- ❖ Velocidade que a bola deve passar por esse ponto:
$$\text{Velocidade da Bola} = \frac{|\text{pos. bola} - \text{ponto I}|}{T_{oponente}}$$

Conclusão

- ❖ Foi possível identificar a velocidade que a bola atinge, em cada instante do chute.
- ❖ Constatou-se a velocidade que a bola deve atingir a partir da posição do robô oponente.

8.2 NAMESPACE COM FUNÇÕES MATEMÁTICAS E TRIGONOMÉTRICAS

```
/*  
    Calculate the intersection point of two lines.  
    The first line is defined from s1 and s2.  
    The other line is defined from t1 and t2.  
    Return:  
        false If the lines are parallel, coincident or not intersecting  
        true If the lines are intersecting. And there is point of intersection:  
        "intersection"  
*/  
bool Support::IntersectionBetweenTwoLine(Vector2D s1, Vector2D s2, Vector2D t1,  
Vector2D t2, Vector2D &intersection)  
{  
    float denom = (t2.y - t1.y)*(s2.x - s1.x) - (t2.x - t1.x)*(s2.y - s1.y);  
    float nume_a = (t2.x - t1.x)*(s1.y - t1.y) - (t2.y - t1.y)*(s1.x - t1.x);  
  
    float nume_b = (s2.x - s1.x)*(s1.y - t1.y) - (s2.y - s1.y)*(s2.x - s1.x);  
  
    // If denom equal 0, the lines are parallel or coincident  
    if(denom == 0.0f)  
    {  
        return false;  
    }  
  
    float ua = nume_a / denom;  
    float ub = nume_b / denom;  
  
    // Return 'true' if the lines are intersecting  
    // Get the intersection point.  
    intersection.x = s1.x + ua*(s2.x - s1.x);  
    intersection.y = s1.y + ua*(s2.y - s1.y);  
}
```

```
        return true;
    }

    /*
    Calculate the intersection of a line and a sphere
    The line segment is defined from p1 to p2
    The sphere is of radius 'radius' and centered at center
    There are potentially two points of intersection given by
        intersection1 = p1 + mu1 (p2 - p1)
        intersection2 = p1 + mu2 (p2 - p1)
    Return FALSE if the line doesn't intersect the sphere.
    */
    bool Support::IntersectionLineCircle(Vector2D p1, Vector2D p2, float radius, Vector2D
center, Vector2D &intersection1, Vector2D &intersection2)
    {
        float      a,b,c;
        float      bb4ac;
        float      mu1, mu2;
        Vector2D    dp;
        dp.x = p2.x - p1.x;
        dp.y = p2.y - p1.y;
        a          = dp.x * dp.x + dp.y * dp.y ;
        b          = 2 * (dp.x * (p1.x - center.x) + dp.y * (p1.y - center.y));
        c          = center.x * center.x + center.y * center.y + p1.x * p1.x + p1.y * p1.y
- 2 * (center.x * p1.x + center.y * p1.y) - radius * radius;
        bb4ac = b * b - 4 * a * c;

        if (System::Math::Abs(a) < 0.0001f || bb4ac < 0)
        {
            mu1 = 0;
            mu2 = 0;
            return false;
        }
    }
```



```
mu1 = (-b + (float)System::Math::Sqrt(bb4ac)) / (2 * a);
intersection1.x = p1.x + mu1*(p2.x - p1.x);
intersection1.y = p1.y + mu1*(p2.y - p1.y);

mu2 = (-b - (float)System::Math::Sqrt(bb4ac)) / (2 * a);
intersection2.x = p1.x + mu2*(p2.x - p1.x);
intersection2.y = p1.y + mu2*(p2.y - p1.y);

return true;
}

/*
Determina o ponto de uma reta tangente ao círculo com um ponto de referência
Entradas: centro do círculo, raio do círculo, ponto de referência
```

```
*/  
void Support::tangentCircle(Vector2D point, Vector2D centerPoint, float radius,  
Vector2D &pointTangent1, Vector2D &pointTangent2)  
{  
    Vector2D centerLine;    centerLine =  
(point + centerPoint)/2;    float r = (point -  
centerPoint).length()/2;  
    Support::getIntersectionBetweenCircles(centerLine, r, centerPoint, radius,  
pointTangent1, pointTangent2);  
}  
  
/*  
    Calculates the intersection points of two overlapping circles.  
    Returns true if the circles intersect.  
    Returns false if the circles do not intersect.  
*/  
bool Support::getIntersectionBetweenCircles(Vector2D centerPoint1, float  
radius1, Vector2D centerPoint2, float radius2, Vector2D &pointIntersect1, Vector2D  
&pointIntersect2)  
{  
    //Distance between centers    float dist =  
(centerPoint2 - centerPoint1).length();  
  
    /*Circles share centers. This results in division by zero,  
infinite solutions or one circle being contained within the other. */  
    if(dist == 0.0)    return false;  
    //Circles do not touch each other  
    else if(dist > (radius1 + radius2))  
        return false;  
    //One circle is contained within the other  
    else if(dist < (radius1 - radius2))  
        return false;  
  
    float a = ((radius1*radius1) - (radius2*radius2) + (dist*dist)) / (2.0f * dist);  
    //Solve for h by substituting a into a^2 + h^2 = radius1^2  
    float h = sqrt((radius1*radius1) - (a*a));  
  
    //Find point p2 by adding the a offset in relation to line d to point p0  
    float px2 = centerPoint1.x + ((centerPoint2.x-centerPoint1.x) * a/dist);  
    float py2 = centerPoint1.y + ((centerPoint2.y-centerPoint1.y) * a/dist);
```



```
//Tangent circles have only one intersection
if(dist == (radius1 + radius2))
{
    pointIntersect1.x = pointIntersect2.y = px2;
    pointIntersect1.y = pointIntersect2.y = py2;
    return true;
}

//Get the perpendicular slope by multiplying by the negative reciprocal
//Then multiply by the h offset in relation to d to get the actual offsets
float mx = -((centerPoint2.y-centerPoint1.y) * h/dist);    float my =
((centerPoint2.x-centerPoint1.x) * h/dist);

//Add the offsets to point p2 to obtain the intersection points
pointIntersect1.x = px2 + mx;    pointIntersect1.y = py2 + my;
pointIntersect2.x = px2 - mx;    pointIntersect2.y = py2 - my;

return true;
}

Vector2D Support::BallInterceptBounds(FutEnvironment *envptr)
{
    float        angle;
    Vector2D      nextPointBall, interceptBounds;

    angle        = envptr->currentBall.velocity.angle();
    nextPointBall.x = envptr->currentBall.pos.x + cos(angle);
    nextPointBall.y = envptr->currentBall.pos.y + sin(angle);

    std::vector<Vector2D> corner;
    corner.push_back(Vector2D(envptr->getFieldGeometry()->field.right, envptr-
>getFieldGeometry()->field.top));    corner.push_back(Vector2D(envptr-
>getFieldGeometry()->field.left, envptr-
>getFieldGeometry()->field.top));    corner.push_back(Vector2D(envptr-
>getFieldGeometry()->field.left, envptr-
>getFieldGeometry()->field.bottom));
    corner.push_back(Vector2D(envptr->getFieldGeometry()->field.right,
envptr->getFieldGeometry()->field.bottom)); for( int i=0; i<3; i++ )
```

```
{
    Vector2D intersect;
    bool intercept = IntersectionBetweenTwoLine(envptr->currentBall.pos,
nextPointBall, corner[i], corner[i+1], intersect);
    if(intercept)
    {
        if (i == 0 && angle > 0 || i == 1 && abs(angle) > PI/2 || i == 2
&& angle < 0 || i == 3 && abs(angle) < PI/2)
        {
            if(interceptBounds.x >= envptr->getFieldGeometry()-
>field.left && interceptBounds.x <= envptr->getFieldGeometry()->field.right)
            {
                interceptBounds = intersect;
            }
        }
    }
    corner.clear();

    return interceptBounds;
}

void Support::Intercept(FutEnvironment *envptr, Vector2D shootTarget, Vector2D
robotPos, float velocity, Vector2D &intercept)
{
    // constantes utilizadas neste trecho do algoritmo
    const float STEP_LENGTH = 43.0f;
    const float ROBOT_RADIUS = 90.0f;

    // Trajetoria da bola
    Vector2D trajectoryball = BallInterceptBounds(envptr);

    bool canIntercept = false;

    // Numero de steps
    int nSteps = (int)(trajectoryball -
envptr->currentBall.pos).length()/STEP_LENGTH;

    std::vector<Vector2D> step;
```

```
        step.push_back(envptr->currentBall.pos);  
        Vector2D stepBall, stepRobot; for( int i=0; i<nSteps &&  
canIntercept == false; i++ )  
        {  
            float m = envptr->currentBall.velocity.angle();  
            stepBall.x = step[i].x + cos(m)*STEP_LENGTH;    stepBall.y =  
            step[i].y + sin(m)*STEP_LENGTH;    step.push_back(stepBall);  
  
            float angle;  
            //angle = (stepBall - shootTarget).angle();  
            angle = (shootTarget - stepBall).angle();  
            stepRobot.x = stepBall.x + cos(angle)*ROBOT_RADIUS;  
            stepRobot.y = stepBall.y + sin(angle)*ROBOT_RADIUS;  
            float timeBall;
```

```
        timeBall = (stepBall - envptr->currentBall.pos).length() / envptr->currentBall.velocity.length();

        float timeRobot, distRobotPoint, desacel_init_dist, vo;
        //teste
        velocity = 5000.0f * velocity / 100;
        desacel_init_dist = 0.75f * velocity + 0.01f; //0.01 serve para nao dar
divisao por zero
        distRobotPoint = (robotPos - stepRobot).length();
        if (distRobotPoint > desacel_init_dist){
            vo = velocity;
        }else{
            vo = velocity * distRobotPoint/(desacel_init_dist);
        }
        timeRobot = (stepRobot - robotPos).length()/vo;

        if(timeBall > timeRobot)
        {
            canIntercept = true;
        }
        if(canIntercept)
        {
            float distRobotTrajectoryBall, distRobotStepBall;
            distRobotTrajectoryBall =
Support::GetPointLineIntercept(&robotPos, &trajectoryball, &envptr->currentBall.pos,
intercept);
            distRobotStepBall = (step[i] - robotPos).length();
            if(distRobotTrajectoryBall < ROBOT_RADIUS*4 )//&&
distRobotStepBall < ROBOT_RADIUS)
                intercept = intercept;
            else
                intercept = stepRobot;
        }
        else
            intercept= robotPos;
    }
    step.clear();
}
```

8.3 BESTPOINTTOSHOOT

```
BestPointToShoot()
{
    Vector2D pointGoalSuperior, pointGoalInferior, bestPoint;
    pointGoalSuperior.y = this->fieldgeometry->goal.top; pointGoalInferior.y
    = this->fieldgeometry->goal.bottom;

    std::vector<float> pointBoundsY;
    pointBoundsY.push_back(pointGoalSuperior.y);
    pointBoundsY.push_back(pointGoalInferior.y);

    if(envptr->getAttackSide() == -1)
    { pointGoalInferior.x = this->fieldgeometry->field.left;
      pointGoalSuperior.x = this->fieldgeometry->
        >field.left;
      bestPoint.x          = this->fieldgeometry->field.left;
    }
    else
    {
        pointGoalInferior.x          = this->fieldgeometry->field.right;
        pointGoalSuperior.x = this->fieldgeometry->field.right; bestPoint.x = this->
        >fieldgeometry->field.right;
    }

    for(int i=0; i<PLAYERS_PER_SIDE; i++)
    {
        Vector2D tan1, tan2;
        Support::tangentCircle(envptr->currentBall.pos, envptr->opponent[i].pos,
        90.0f, tan1, tan2);
        Support::IntersectionBetweenTwoLine(envptr->currentBall.pos, tan1,
        pointGoalInferior, pointGoalSuperior, tan1);
        Support::IntersectionBetweenTwoLine(envptr->currentBall.pos, tan2,
        pointGoalInferior, pointGoalSuperior, tan2);

        if( envptr->getAttackSide() == -1 && (envptr->currentBall.pos.x > envptr->
        >opponent[i].pos.x) ||
```

```
envptr->getAttackSide() == 1 && (envptr->currentBall.pos.x < envptr-
>opponent[i].pos.x))
{
    if(abs(tan1.y) < this->fieldgeometry->goal.height/2)
    {
        pointBoundsY.push_back(tan1.y);
    }
    if(abs(tan2.y) < this->fieldgeometry->goal.height/2)
    {
        pointBoundsY.push_back(tan2.y);
    }
}
sort(pointBoundsY.begin(), pointBoundsY.end());
int nPoints = pointBoundsY.size();      float
diff, diffMaior=0.0f;
for(int i=0; i<(nPoints-1); i++)
{
    diff = pointBoundsY[i] - pointBoundsY[i+1];
    bool hasRobot = false;
    for(int ii=0; ii<PLAYERS_PER_SIDE && hasRobot == false; ii++)
    {
        hasRobot = Support::DistancePointLine(&envptr->opponent[ii].pos,
&envptr->currentBall.pos, &Vector2D(bestPoint.x, (pointBoundsY[i] +
pointBoundsY[i+1])/2), 90.0f);
    }
    if(abs(diff) > diffMaior && hasRobot == false)
    {
        diffMaior = abs(diff);
        bestPoint.y = (pointBoundsY[i] + pointBoundsY[i+1])/2;
    }
}
pointBoundsY.clear();
```

```
}  
    return bestPoint;  
}
```

8.4 EXECUTE: PASSTOROBOT

```
int PassToRobotSkill::Execute(Robot *robot, FutEnvironment *env) {  
    bool front = false;  
  
    if(env->currentBall.velocity.x < 0 && env->  
currentBall.velocity.length() > 200) { robot->  
AddDebugMessage("Ball Moving");  
        front = this->FrontIntercept(robot, env);  
    }  
  
    if(front == false) {  
        this->Normal(robot, env);  
    }  
  
    robot->planning.velocity = MAX_VELOCITY;  
  
    return 0;  
}  
  
bool PassToRobotSkill::FrontIntercept(Robot *robot, FutEnvironment  
*env) {  
    robot->AddDebugMessage("FrontIntercept"); //  
    If the velocity exceeds 100  
        if(env->currentBall.velocity.length() > 100) {  
            Vector3D closestPoint; // closestPoint: Is the  
point more closest of trajectory of the ball  
  
            // If the robot can intercept the ball. It is determined the point  
            if(this->CalcTimeBallClosestPoint(robot, closestPoint,  
env) == true) {  
                float shootTargetAngle = (robot->  
planning.shootTarget-closestPoint).AnglePI();
```

```
robot->planning.targetAngle    =    shootTargetAngle;          robot-
>planning.plannerAngle = 1000.;    robot->planning.includeBall = false;

// If the length between the robot and the ball is greater than
"ROBOT_RADIUS*1.8"
                                if((env->currentBall.pos - closestPoint).length() >
ROBOT_RADIUS*1.8) {
                                robot->planning.destination.x = closestPoint.x
- cos(shootTargetAngle)*(ROBOT_RADIUS+BALL_RADIUS);
                                robot->planning.destination.y = closestPoint.y
- sin(shootTargetAngle)*(ROBOT_RADIUS+BALL_RADIUS);
                                robot->kick    = this->kickLength;    robot->customKick  = this-
>customKick;    robot->roller  = true;
                                robot->cmd_rollerspd = 80;
                                }
```



```
        // If the length between the robot and the ball is
less than "ROBOT_RADIUS*1.8"
        else {
            robot->planning.destination      = env-
>currentBall.pos;
            robot->kick                      = this->kickLength;
            robot->customKick                = this->customKick;
        }
        robot->planning.velocity            = MAX_VELOCITY;
    }
    // If the robot can't intercept the ball
    else {
        return false;
    }
}
robot->roller = false;
float rotation = robot->rotation;

if(Support::IsFacingOpponentGoal(robot->pos, rotation, env)) {
    robot->kick          = this->kickLength;
    robot->customKick = this->customKick;
}
return true;
}

void PassToRobotSkill::Normal(Robot *robot, FutEnvironment *env) {

    float robotRotation = robot->rotation;
    float targetAngle = (robot->planning.shootTarget-env-
>currentBall.pos).angle();
    float angleToDest = targetAngle;
    float distToBall = (robot->pos - env-
>currentBall.pos).length();
    float angleRobot2Target = abs(robotRotation - targetAngle);
    bool sball_in_front      = true;

    if ( angleRobot2Target < maxAngleError ) { //
Check if the robot angle to kick is correct
        robot->kick          = this->kickLength;
        robot->customKick = this->customKick;
    }
    else{
        robot->kick = 0;
    }

    // If the distance between robot and the ball is less than
"ROBOT_RADIUS*3" set roller
    if (distToBall < ROBOT_RADIUS*3){
        robot->roller = true;
        robot->cmd_rollerspd = 80;
    }
    else {
        robot->roller = false;
    }
}
```



```

        if(robot->kickSensor == true) { // If sensor indicates that the
ball is present
            angleToDest = 1000; // We dont need of
angle to get the ball. Because we already have it
        }

        // If robot doesn't with the ball in sensor
        if(angleToDest != 1000) {
            Vector3D sball_rel = env->currentBall.pos - robot-
>pos; // Vector of difference between position of ball and robot
            Vector3D srobot_ball = sball_rel.rotate(-robot-
>rotation); // New position of ball with source robot
            sball_in_front = (srobot_ball.x > -20) &&
(Math::Abs(srobot_ball.y) < 30); // true if the ball is in front of the
robot

            // If ball is in front of the robot
            if(sball_in_front) {
                float angleBall2Robot = (env->currentBall.pos -
robot->pos).angle(); // Angle between the ball and robot
                float delta = abs(angleBall2Robot
- angleToDest); // Difference between
                if(delta < PI/6) {
                    angleToDest = 1000;
                }
            }
        }

        if(angleToDest != 1000) {
            Vector3D sball_rel= robot->planning.shootTarget - robot-
>pos;
            Vector3D srobot_ball= sball_rel.rotate(-robot->rotation);
            bool sball_in_front = (srobot_ball.x > -10) &&
(Math::Abs(srobot_ball.y) < 60); // a bola esta na frente do robo?
            Vector3D sssball_rel= env->currentBall.pos - robot->pos;
            Vector3D sssrobot_ball = sssball_rel.rotate(-robot-
>rotation);
            bool sssball_in_front = (sssrobot_ball.x >
ROBOT_RADIUS+BALL_RADIUS) && (Math::Abs(sssrobot_ball.y) < 120);
            if(sssball_in_front) {
                targetAngle = (env->currentBall.pos-robot-
>pos).angle();
            }
        }

        if(robot->kickSensor == true) {
            float dist2Ball = (robot->pos - env-
>currentBall.pos).length();
            if(dist2Ball < ROBOT_RADIUS-BALL_RADIUS) {
                destination = robot->pos;
            }
        }
        else{
            if(distToBall > 4*ROBOT_RADIUS){

```



```

        destination =
FunctionMath::IntersectionCircleOutLine(robot->planning.shootTarget, env-
>currentBall.pos, 3*ROBOT_RADIUS);
        robot->planning.velocity = MAX_VELOCITY;
    }
    else{
        robot->roller = true;
        robot->cmd_rollerspd = 80;
        destination = Support::rotateRobot(robot,
3*BALL_RADIUS, env->currentBall.pos, robot->planning.shootTarget);
        if ( angleRobot2Target < maxAngleError ){
            destination = env->currentBall.pos;
            robot->AddDebugMessage("Angle is Correct");
        }
    }
    robot->planning.destination = destination;
    robot->planning.plannerAngle = angleToDest;
    robot->planning.targetAngle = targetAngle;
    this->useBallIntercept = true;
}

    /// This function return "false" or "true", if the robot can
    intercept the ball
    bool PassToRobotSkill::CalcTimeBallClosestPoint(Robot *robot,
Vector3D &closestPoint, FutEnvironment *env) {
        Vector3D p1,p2,point; //
Points main of algorithms
        float ballCoef = Support::BallCoef(env); // "coeficiente
angula" of the ball

        // p1 = Current position of the ball
        p1 = env->currentBall.pos;
        //p2 = Extent of the field lines
        p2 = Support::LinhaDoCampoEncontraBola(ballCoef, env-
>currentBall.pos,env);

        // Calculates the distance and the closest point between the
robot and the line ( p1 and p2 )
        float dist =
Support::GetPointLineIntercept(&robot->pos,&p1,&p2,closestPoint);
        float time2ClosestPoint = (closestPoint-robot-
>pos).length()/MAX_ROBOT_VEL;
        float time2BallClosestPoint = (closestPoint - env-
>currentBall.pos).length()/env->currentBall.velocity.length();
        float deltaTime = time2ClosestPoint -
time2BallClosestPoint;

        if(deltaTime < 0) {
            return true;
        }
        return false;
    }

```