

**MINISTÉRIO DA DEFESA
EXÉRCITO BRASILEIRO
DEPARTAMENTO DE CIÊNCIA E TECNOLOGIA
INSTITUTO MILITAR DE ENGENHARIA
(Real Academia de Artilharia, Fortificação e Desenho/1792)
CURSO DE GRADUAÇÃO EM ENGENHARIA DE COMPUTAÇÃO**

**Jan Segre
Victor Bramigk**

**Uma Ferramenta de Representação
Comportamental Baseada em Otimização
para Futebol de Robôs**

**Rio de Janeiro
Maio de 2015**

Instituto Militar de Engenharia

Uma Ferramenta de Representação Comportamental Baseada em Otimização para Futebol de Robôs

Projeto Final de Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientador: Paulo Fernando Ferreira Rosa

Orientador: Bruno Eduardo Madeira

**Rio de Janeiro
Maio de 2015**

INSTITUTO MILITAR DE ENGENHARIA
Praça General Tibúrcio, 80 - Praia Vermelha
Rio de Janeiro-RJ CEP 22290-270

Este exemplar é de propriedade do Instituto Militar de Engenharia, que poderá incluí-lo em base de dados, armazenar em computador, microfilmar ou adotar qualquer forma de arquivamento.

É permitida a menção, reprodução parcial ou integral e a transmissão entre bibliotecas deste trabalho, sem modificação de seu texto, em qualquer meio que esteja ou venha a ser fixado, para pesquisa acadêmica, comentários e citações, desde que sem finalidade comercial e que seja feita a referência bibliográfica completa.

Os conceitos expressos neste trabalho são de responsabilidade do(s) autor(es) e do(s) orientador(es).

629.892 Segre, Jan

S455h Uma ferramenta de representação comportamental baseado em otimização para futebol de robôs / Jan Segre, Victor Bramigk; orientados por Paulo Fernando Ferreira Rosa - Rio de Janeiro: Instituto Militar de Engenharia, 2015.

64p. : il.

Projeto de Fim de Curso (PFC) - Instituto Militar de Engenharia - Rio de Janeiro, 2015.

1. Curso de engenharia da computação - Projeto Final de Curso. 2. Robótica. 3. Otimização. I. Bramigk, Victor. II. Rosa, Paulo Fernando Ferreira. III. Bruno Eduardo Madeira. IV. Título. V. Instituto Militar de Engenharia.

Instituto Militar de Engenharia

**Jan Segre
Victor Bramigk**

Uma Ferramenta de Representação Comportamental Baseada em Otimização para Futebol de Robôs

Projeto Final de Curso de Graduação em Engenharia de Computação do Instituto Militar de Engenharia.

Orientadores: Paulo Fernando Ferreira Rosa e Bruno Eduardo Madeira

Aprovado em 25 de maio de 2015 pela seguinte Banca Examinadora:

Paulo Fernando Ferreira Rosa - Ph.D., do IME
Orientador

Bruno Eduardo Madeira - M.Sc., do IME
Orientador

Ricardo Choren Noya - D.Sc., do IME

Julio Cesar Duarte - D.Sc., do IME

**Rio de Janeiro
Maio de 2015**

Resumo

Este trabalho apresenta uma ferramenta de representação comportamental baseada em otimização para futebol de robôs para uma equipe da *Small Size League* (SSL) da RoboCup, RoboIME (representando IME nesta competição). Equipes da SSL são geralmente controlados por heurísticas puras. Isso restringe o movimento dos robôs para um conjunto fixo de comportamentos, limitando as possíveis jogadas.

O objetivo deste trabalho é desenvolver uma ferramenta de representação comportamental baseada em otimização para futebol de robôs.

Um modelo discreto e sequencial no domínio das ações foi criado, juntamente com uma função utilidade. Isso possibilitou que uma busca seja feita para encontrar jogadas com boa avaliação de acordo com essa função. Foi implementada uma arquitetura de controle baseada nessa abstração, onde é feita uma busca pela combinação do método de descida do gradiente aplicado a um conjunto de jogadas aleatórios, ações sugeridas (inseridos através da GUI) e o planejamento anterior. A ferramenta é capaz de controlar um conjunto de robôs, dando origem a comportamentos que vão desde bloquear chutes diretos, avançar para a quadra adversária e se posicionar para atacar através de passes.

Palavras-chaves: inteligência artificial, discretização, otimização, robótica, robocup.

Abstract

This paper presents a behavioral representation tool based on optimization for robot soccer for a RoboCup's Small Size League (SSL) team, RoboIME (representing IME in this competition). SSL teams are usually controlled by pure heuristics. This restricts the movement of the robots to a fixed set of behaviors, limiting the possible plays.

The objective of this work is to develop a behavioral representation tool based on optimization for robot soccer.

A discreet and sequential model in the field of actions was created, along with a utility function. This enabled a search to be made to find plays with good evaluation according to that function. A control architecture based on this abstraction was implemented, where a search is made by combining a gradient descent method applied to a set of random moves, suggested actions (inserted through the GUI) and the previous planning. The tool is able to control a set of robot, yielding behaviors which range from blocking direct kicks, advancing to the opponent's court and positioning for attacking through passes.

Key-words: artificial intelligence, discretization, optimization, robotics, robocup.

Sumário

1	Introdução	11
1.1	Motivação	12
1.2	Objetivo	13
1.3	Justificativa	13
1.4	Metodologia	14
1.5	Estrutura	14
2	Modelagem	15
2.1	Arquitetura do jogo	15
2.2	Definições	16
2.3	Discretização do jogo	22
2.3.1	Representação do jogo	22
2.3.1.1	Posse de bola	22
2.3.1.2	Ações Consideradas	23
2.3.2	Execução do planejamento	24
2.4	Função Objetivo	24
2.4.1	Custo da Abertura do Gol	25
2.4.2	Correção da Abertura do Gol Devido a Movimentação da Bola	26
2.4.3	Distância Total das Ações $Mover(r)$	26
2.4.4	Distância Máxima das Ações $Mover(r)$	26
2.4.5	Mudança do Planejamento da Tabela de Decisão	27
2.4.6	Custo do Ataque	27
2.4.7	Custo da Defesa	27
2.4.8	Custo das Aberturas do Gol Vistas por $r \in T_c$	28
2.4.9	Custo das Aberturas do Gol Vistas por $r \in T_{ad}$	28
2.4.10	Número de Receptores	28
2.4.11	Penalização por Proximidade do Gol do Adversário	28
2.4.12	Abertura Mínima para Chute a Gol	28
2.4.13	Número da Ramificação	29
2.5	Estratégia de Busca	29
2.5.1	Distribuição da Ação $Mover(r)$	30
2.5.2	Tabela de Decisão	30
2.5.3	Posições Chave	31
3	Arquitetura do Software	34
3.1	Comunicação com Componentes Externos	35
3.2	Threads do Sistema	36
3.3	Interface Gráfica	37

4	Resultados	39
4.1	Interface Gráfica	39
4.2	Influência dos Parâmetros no Comportamento do Time	45
4.2.1	Correção da Abertura do Gol Devido a Movimentação da Bola	46
4.2.2	Custo das Aberturas vistas por $r \in T_c$	48
4.2.3	Custo das Aberturas vistas por $r \in T_{ad}$	50
4.2.4	Abertura mínima para chute à gol	51
4.3	Goleiro	51
5	Conclusão	52
Referências		53
Apêndices		54
APÊNDICE A API Pública		55

Listas de ilustrações

Figura 1.1 – Imagem da SSL <i>RoboCup</i> 2013 em Eindhoven, na Holanda	11
Figura 2.1 – Arquitetura básica da SSL	16
Figura 2.2 – Padrões oficiais da SSL [1]	17
Figura 2.3 – Parâmetros de estado mutáveis do robô	17
Figura 2.4 – Disposição das câmeras no campo	18
Figura 2.5 – Atuadores típicos de um robô da SSL	19
Figura 2.6 – Abertura do gol considerando-se uma variação de 15cm na posição da bola (esquerda) e sem variação (direita)	26
Figura 2.7 – Ilustração da distribuição dos estados considerados no planejamento .	29
Figura 2.8 – Sugestão de uma barreira (em rosa)	32
Figura 2.9 – Sugestão de posições laterais (em rosa)	32
Figura 3.1 – Diagrama de comunicação entre os componentes.	35
Figura 3.2 – Diagrama de relação entre <i>threads</i> e dados.	36
Figura 3.3 – Diagrama de relação entre componentes gráficos	38
Figura 4.1 – Aparência geral e representação do campo	39
Figura 4.2 – Zoom e arraste do campo	40
Figura 4.3 – Parâmetros configuráveis	41
Figura 4.4 – Controles	41
Figura 4.5 – Representação do tempo para chegar na bola	42
Figura 4.6 – Representação de ação de passe	43
Figura 4.7 – Representação de ação de chute	43
Figura 4.8 – Representação das ações de movimentação	44
Figura 4.9 – Planejamento com os parâmetros iniciais no ataque	45
Figura 4.10–Planejamento com os parâmetros iniciais na defesa	46
Figura 4.11–Planejamento com os parâmetros iniciais e sem ajuste de movimentação da bola em ambiente de ataque (acima) e defesa (abaixo)	47
Figura 4.12–Planejamento com os parâmetros iniciais e o ajuste de movimentação da bola ajustado para 0.24 em ambiente de ataque (acima) e defesa (abaixo)	48
Figura 4.13–Planejamento com os parâmetros iniciais e com o peso do custo das aberturas do gol vistas pelos robôs do time igual a 1000. No ataque (acima) e na defesa (abaixo)	49
Figura 4.14–Planejamento com os parâmetros iniciais e com o custo das aberturas do goal vistos pelos robôs do time nulo. No ataque (acima) e na defesa (abaixo)	50

Figura 4.15–Planejamento com os parâmetros iniciais e com a abertura mínima para chute alterado para 5. No ataque (acima) e na defesa (abaixo) 51

Lista de abreviaturas e siglas

API	<i>Application Programming Interface</i>
IA	Inteligência Artificial
SSL	<i>Small Size League</i>
Robocup	<i>Robotics World Cup</i>
RoboIME	Equipe de alunos do Laboratório de Robótica do IME

1 Introdução

A robótica é um ramo da tecnologia que lida com a concepção, construção, operação e aplicação de máquinas capazes de realizar uma série de ações de maneira autônoma. Atualmente é um tópico em rápida ascensão. Pesquisar, projetar e fabricar novos robôs serve vários propósitos práticos tais como domésticos, comerciais e militares. Um dos problemas atuais da robótica é o planejamento em ambientes multi-agentes dinâmicos e competitivos. Um exemplo de um problema dessa classe é um jogo de futebol de robôs, onde um grupo de robôs é controlado por uma IA independente. A Figura 1.1 mostra uma imagem da Robocup 2013, competição internacional de robótica, onde a equipe RoboIME (de alunos do Laboratório de Robótica do IME) participou.



Figura 1.1 – Imagem da SSL *RoboCup* 2013 em Eindhoven, na Holanda

Devido a alta complexidade desses ambientes, não é viável o planejamento considerando diretamente as leis físicas. Como consequência, limita-se as ações possíveis do robô no modelo utilizado no planejamento para que se possa simular mais situações em tempo hábil, uma vez que o ambiente está continuamente sujeito a modificações. Entretanto, para que as simulações sejam válidas, o robô real deve estar em sintonia com seu modelo. Com efeito, o robô real deve executar os comandos conforme o robô simulado, caso o mesmo ambiente simulado seja encontrado na prática.

A ideia de robôs jogando futebol foi mencionada pela primeira vez pelo professor Alan Mackworth (University of British Columbia, Canadá) em um artigo intitulado "On Seeing

Robots", apresentado no Vision Interface 92 e posteriormente publicado em um livro chamado Computer Vision: System, Theory and Applications [2]. Independentemente, um grupo de pesquisadores japoneses organizou um Workshop no Ground Challenge in Artificial Intelligence, em Outubro de 1992, Tóquio, discutindo e propondo problemas que representavam grandes desafios. Esse Workshop os levou a sérias discussões sobre usar um jogo de futebol para promover ciência e tecnologia. Estudos foram feitos para analisar a viabilidade dessa ideia. Os resultados desses estudos mostram que a ideia era viável, desejável e englobava diversas aplicações práticas. Em 1993, um grupo de pesquisadores, incluindo Minoru Asada, Yasuo Kuniyoshi e Hiroaki Kitano, lançaram uma competição de robótica chamada de Robot J-League (fazendo uma analogia à J-League, nome da Liga Japonesa de Futebol Profissional). Em um mês, vários pesquisadores já se pronunciavam dizendo que a iniciativa deveria ser estendida ao âmbito internacional. Surgia então, a Robot World Cup Initiative (RoboCup).

RoboCup é uma competição destinada a desenvolver os estudos na área de robótica e Inteligência Artificial (IA) por meio de uma competição amigável. Além disso, ela tem como objetivo, até 2050, desenvolver uma equipe de robôs humanoides totalmente autônomos capazes de derrotar a equipe campeã mundial de futebol humano. A competição possui várias modalidades. Neste trabalho, será analisada a Small Size Robot League (SSL), também conhecida como F180. De acordo com as regras da SSL de 2015, as equipes devem ser compostas por 6 robôs, sendo um deles o goleiro, que deve ser designado antes do início do jogo. Durante o jogo, nenhuma interferência humana é permitida com o sistema de controle dos robôs. É fornecido aos times um sistema de visão global e esses controlam seus robôs através de máquinas próprias. O sistema de controle dos robôs geralmente é externo e recebe os dados de um conjunto de duas câmeras localizadas acima do campo. Esse sistema de controle processa os dados, determina qual comando deve ser executado por cada robô e envia este comando através de ondas de rádio aos robôs.

1.1 Motivação

O futebol de robôs, problema padrão de investigação internacional, reúne grande parte dos desafios presentes em problemas do mundo real a serem resolvidos em tempo real. As soluções encontradas para o futebol de robôs podem ser estendidas, possibilitando o uso da robótica em locais de difícil acesso para humanos, ambientes insalubres e situações de risco de vida iminente. Há diversas novas áreas de aplicação da robótica, tais como exploração espacial e submarina, navegação em ambientes inóspitos e perigosos, serviço de assistência médica e cirúrgica, além do setor de entretenimento. Essas áreas podem ser beneficiadas com o desenvolvimento de sistemas multi robôs. Nestes domínios de aplicação, sistemas de multi robôs deparam-se sempre com tarefas muito difíceis de serem

efetuadas por um único robô. Um time de robôs pode prover redundância e contribuir cooperativamente para resolver o problema em questão. Com efeito, eles podem resolver o problema de maneira mais confiável, mais rápida e mais econômica, quando comparado com o desempenho que único robô teria.

Devido a alta complexidade de sistemas multi-agentes dinâmicos, torna-se necessário um modelo simplificado para que sejam executadas o maior número de simulações possível. Caso seja possível uma discretização, ter-se-á um número finito de casos para serem avaliados. Com isso, pode-se desenvolver um sistema multi-agente baseado em utilidade. Assim, o computador passa a escolher parte da estratégia com base na função utilidade escolhida.

Isso é mais desejável que um modelo heurístico de IA, onde as soluções são criadas com base nos ambientes identificados pelos modeladores. Isso, pois a modelagem puramente heurística limita o número de jogadas que se pode executar e limita a capacidade que o computador tem de testar um grande número de possibilidades. O resultado é que a qualidade das jogadas se limita a capacidade de quem cria as heurísticas.

1.2 Objetivo

O objetivo deste trabalho é desenvolver uma ferramenta de representação comportamental baseado em otimização para futebol de robôs. A meta intermediária é criar um modelo discreto sequencial para o problema do futebol de robôs. A partir desta discretização, foi desenvolvida uma arquitetura de controle que seleciona jogadas o mais próximas da jogada ótima possível, de acordo com uma função de avaliação e dentro do tempo disponível para o planejamento.

1.3 Justificativa

Uma arquitetura de controle que simule os diversos ambientes dinamicamente de maneira sequencial de um ambiente multi-agente permite que várias jogadas sejam criadas dinamicamente, diferentemente de uma arquitetura estática baseada somente em heurística. Essa abordagem heurística tem origem na maneira como estratégias são planejadas nos times de futebol humano.

Com tal mecanismo é possível melhorar a IA em uso pela RoboIME para tomar decisões que levem a resultados melhores e, como consequência, ganhar mais partidas. Nenhuma equipe atualmente esta seguindo esta abordagem, mas os autores acreditam que essa é uma linha de pesquisa promissora, já que se utiliza da capacidade que o computador tem de simular várias possibilidades em um curto intervalo de tempo.

1.4 Metodologia

Para atingir os objetivos propostos foi seguida a seguinte metodologia. O problema em questão foi modelado. A partir dessa modelagem, foi criada uma função de avaliação para se avaliar as jogadas consideradas. Com base nesse estudo, foi construída uma ferramenta para escolher jogadas com base nessa função. Ao longo dos testes, foi necessário ajustar a função citada, bem como o método de busca utilizado.

1.5 Estrutura

No Capítulo 2, um modelo do futebol de robôs é proposto. Também é apresentada a função de avaliação.

No Capítulo 3, a ferramenta proposta é desenvolvida com base no modelo proposto anteriormente.

No Capítulo 4, são apresentados os resultados obtidos neste trabalho.

Finalmente, são apresentados os principais resultados atingidos neste trabalho.

2 Modelagem

Neste capítulo é criado um modelo sequencial do problema do futebol de robôs. Para isso, a arquitetura do futebol de robôs considerada neste trabalho é detalhada. São apresentadas definições de termos utilizados nas seções seguintes, juntamente com um estudo das influências da arquitetura do jogo descrita anteriormente.

Um modelo abstrato discreto do futebol de robôs é proposto para simplificar o problema e permitir que sejam feitas várias simulações durante o planejamento das jogadas. Também será apresentada a tabela de decisão, necessária para permitir que o melhor planejamento seja armazenado, bem como incorporar a mudança de planejamento na função de avaliação. A função de avaliação utilizada para avaliar as jogadas.

Devido a complexidade do problema, também foi desenvolvida uma maneira de incorporar sugestões ao planejamento. Essas sugestões são chamadas de *posições chave*, cuja descrição encontra-se no final deste capítulo.

2.1 Arquitetura do jogo

A arquitetura a ser considerada é baseada em [3]. A *RoboCup Small Size League* (SSL) envolve vários sistemas. Logo, com o objetivo de facilitar a compreensão do problema, a arquitetura do jogo a ser considerada é apresentada na Figura 2.1. Essa arquitetura é composta pelos seguintes sistemas:

- Câmeras Visão: conjunto de câmeras *firewire* que captura as imagens do campo e as envia para a SSL-Vision;
- Comunicação: módulo responsável por receber os parâmetros dos motores, drible, chute baixo e chute alto e enviar o comando via ondas de rádio para os robôs;
- Mundo Real: campo de futebol real, onde os times 1 e 2 interagem através de seus respectivos robôs
- Referee-Box: *software* padronizado pela Robocup para que as regras da competição sejam cumpridas sem que haja intervenção humana excessiva durante uma partida;
- *Software Time 1/2*: *software* do time 1/2;

- SSL-Vision: *software* padronizado pela Robocup que permite a integração com um conjunto de câmeras *firewire* que capturam imagens do campo e as processa, extraíndo informações de posicionamento dos robôs e da bola contidas nessas imagens;
- Time 1/2: time de robôs que executa os comandos recebidos pelo sistema de transmissão do time 1/2;
- Transmissão 1/2: sistema de transmissão do time 1/2;

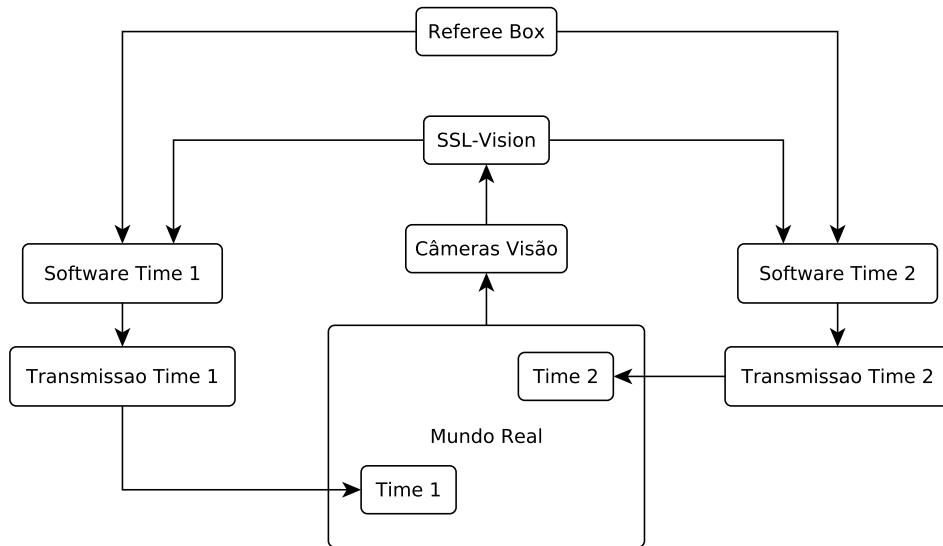


Figura 2.1 – Arquitetura básica da SSL

Conforme apresentado na Figura 2.5, cada robô presente no campo possui um padrão de cores. A cor central indica a que time o robô pertence, podendo ser azul ou amarela. A disposição das outras cores é utilizada para identificar o número do robô, i.e., cores diferentes são identificadas com números diferentes. A Figura 2.2 apresenta os padrões oficiais da competição.

A posição dos círculos que compõem o padrão são utilizados para calcular a orientação e posição do robô no campo, conforme ilustrado na Figura 2.3. Isso tudo é feito pelo *software* da SSL-Vision, que se utiliza das imagens capturadas pelas duas câmeras posicionadas acima do campo. Esses dados são fornecidos em média a cada 16,6 ms. A Figura 2.4 mostra a disposição das câmeras no campo.

2.2 Definições

Definição 2.2.1 (Corpo Rígido). *Um corpo rígido cr é definido por dois subconjuntos disjuntos de parâmetros cr = {̄cr, ̄̄cr} em que:*

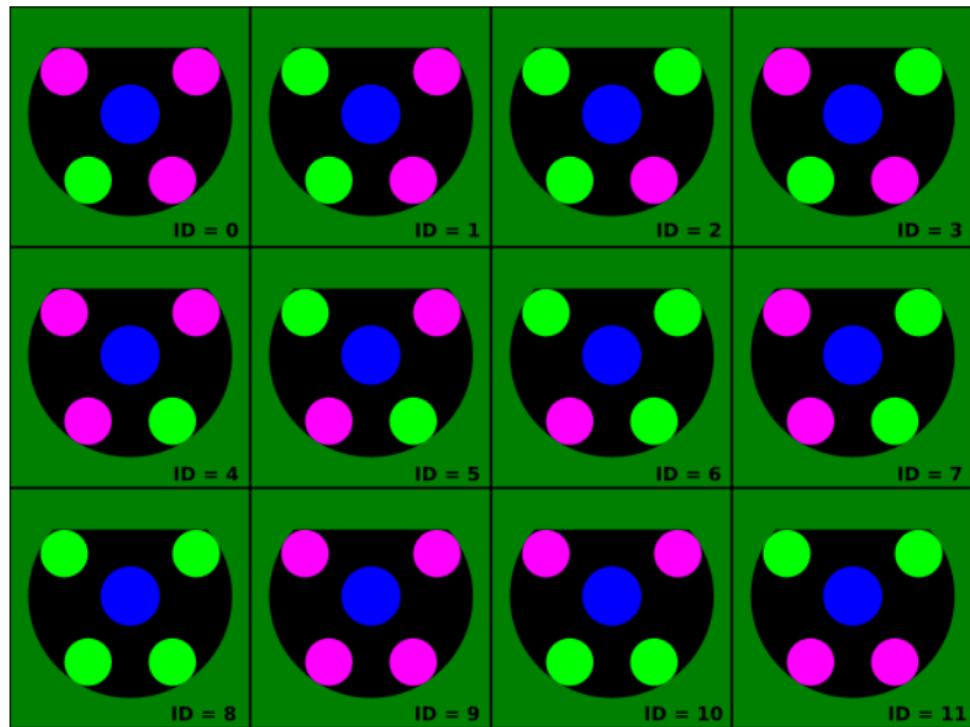


Figura 2.2 – Padrões oficiais da SSL [1]

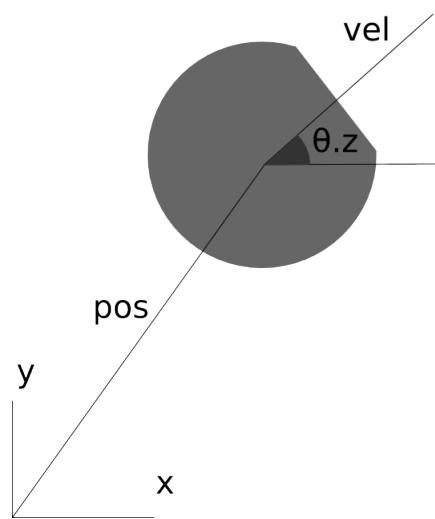


Figura 2.3 – Parâmetros de estado mutáveis do robô



Figura 2.4 – Disposição das câmeras no campo

- $\hat{c}r = \langle pos, \theta, vel, \omega \rangle$, que são os parâmetros de estado mutáveis, respectivamente: posição (\mathbb{R}^3), orientação (\mathbb{R}^3), velocidade linear (\mathbb{R}^3), velocidade angular (\mathbb{R}^3)
- $\bar{c}r$: parâmetros imutáveis do corpo que descrevem sua natureza fixa e que permanecem constantes ao longo do curso de planejamento.

Exemplos de parâmetros considerados nesta modelagem imutáveis são: coeficiente de atrito estático e dinâmico, descrição 3D do corpo (por exemplo, por meio de um conjunto de primitivas 3D), centro de massa no referencial do corpo, coeficiente de restituição, coeficiente de amortecimento linear e angular.

Definição 2.2.2 (Bola). *Bola é um corpo rígido b , no qual somente as componentes $\langle x, y \rangle$ do parâmetro $\hat{b}.pos$ são observáveis.*

De acordo com a arquitetura do jogo descrita na Seção 2.1, tem-se que, a partir de uma sequência de quadros, é possível obter um valor estimado para o parâmetro $\hat{b}.vel$ a partir do intervalo entre os dados recebidos da *SSL-Vision* e da equação $vel \cong \frac{\Delta pos}{\Delta t}$. Entretanto, uma vez que a componente z de $\hat{b}.pos$ não é observável, $\hat{b}.vel.z$ não pode ser estimada a partir do intervalo entre os dados recebidos da *SSL-Vision*. Semelhantemente, uma vez que o parâmetro $\hat{b}.\theta$ também não pode ser observado, não se pode estimar o valor de $\hat{b}.\omega$ com exatidão.

Definição 2.2.3 (Robô). *Robô, representado por r , é um conjunto de sistemas compostos de corpos rígidos, hardware e firmware. Neste trabalho será considerado que o robô tem os seguintes sistemas:*

- *Drible:* imprime um torque a bola;
- *Chute baixo:* imprime uma força à bola b e, possivelmente, um torque, com o objetivo de alterar as componentes $\langle x, y \rangle$ do parâmetro $\hat{b}.vel$;
- *Chute alto:* imprime uma força à bola b e, possivelmente, um torque, com o objetivo de alterar as componentes $\langle x, y, z \rangle$ do parâmetro $\hat{b}.vel$, com $\hat{b}.vel_z \neq 0$;
- *Receptor:* recebe comandos enviados pelo sistema de transmissão de seu respectivo time;
- *Sistema de movimentação:* imprime uma força e um torque ao centro de massa global do r.

A Figura 2.3 ilustra os parâmetros de estado mutáveis do robô. Por meio dos sistemas listados acima, cada robô r pode executar um conjunto de ações A_{rob} . A Figura 2.5 apresenta os atuadores típicos de um robô da SSL.

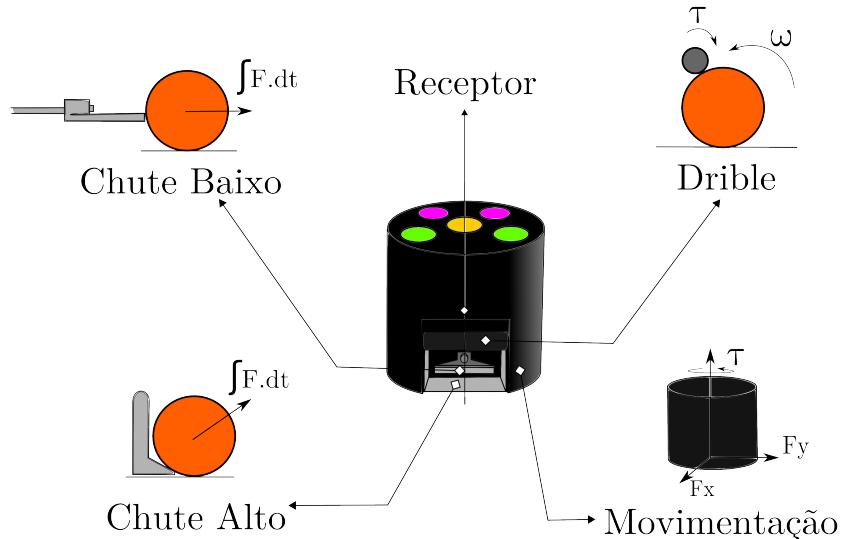


Figura 2.5 – Atuadores típicos de um robô da SSL

Apesar de o modelo descrito acima abranger a maioria dos robôs utilizados atualmente por equipes da SSL, é importante ressaltar que o robô pode ter um conjunto de sensores que poderiam coletar informações adicionais às transmitidas pela *SSL-Vision* juntamente com um sistema de transmissão para enviá-las ao software do seu respectivo time. Isso é

interessante, pois, conforme observado na Definição 2.2.2, o parâmetro $\hat{b}.\theta$ não é observável. Como o sistema de drible impõe um torque à bola, por meio de um sensor, é possível estimar o valor de $\hat{b}.\omega$. Sem esse sensor, não é possível prever com exatidão a trajetória da bola somente com informações de simulação ou da visão.

Definição 2.2.4 (Time). *Sejam os seguintes parâmetros:*

Rob_c o conjunto dos robôs controlados;

Rob_{ad} o conjunto dos robôs adversários, isto é, não controlados;

X o espaço de estado de todos os corpos rígidos envolvidos na partida considerada;

$x_{init} \in X$ o estado inicial;

$X_{goal} \subset X$ o conjunto de estados objetivo;

x_{ob}^i os estados observados pelo módulo SSL-Vision no instante i ;

$X_{ob}^i = \{x_{ob}^0 = x_{init}, \dots, x_{ob}^i\};$

$Sk \subset A_{rob}$ um conjunto de skills;

$prob : X \rightarrow [0, 1]$ uma distribuição de probabilidade, cujo argumento é $x \in X$;

$tk = G(V \in Sk, E \in prob)$ o conjunto de todas as táticas possíveis formadas a partir de grafos orientados, em que os vértices são skills $sk \in Sk$ e as arestas são prob associadas a possibilidade de ocorrerem as transições entre uma skill e outra;

$A_c = A_{rob1} \cup \dots \cup A_{robn_c}$ o conjunto das ações possíveis de Rob_c ;

$A_{ad} = A_{rob1} \cup \dots \cup A_{robn_{ad}}$ o conjunto das ações possíveis de Rob_{ad} ;

$A = A_c \cup A_{ad}$ o conjunto das ações possíveis de $Rob_c \cup Rob_{ad}$;

$e : \langle x, a \rangle \rightarrow x'$ a função de transição de estado que pode aplicar uma ação $a \in A$ em um estado particular $x \in X$ e computar o estado seguinte $x' \in X$;

$f_U : X \rightarrow \mathbb{R}^+ \cup \{0\}$ uma função utilidade tal que $f_U(x)$ mede a utilidade do estado $x \in X$ um entre estados do mundo dado os estados;

$m_{reac ad} : \langle A_{ad}, X_{ob}^i \rangle \rightarrow a'_{ad}$ o modelo de reação dos robôs adversários dado X_{ob}^i ;

$AB = \{V \subset X, E \subset A\}$ uma árvore de busca;

$e_b : \langle X_{ob}^i, e, f_U, m_{reac ad}, AB \rangle \rightarrow AB'$ uma estratégia de busca.

Então, um time T é definido por:

$$T : \langle Rob_c, A, X_{ob}^i, e, e_b, m_{reac\ ad} \rangle \longrightarrow a_c^{i+1}$$

Assim, utilizando-se de e , T pode simular várias sequência de ações a_c dado X_{ob}^i a partir de f_U e e_b . Pode-se, a partir de $m_{reac\ ad}$, considerar as ações do time adversário baseado em estados anteriores.

Definição 2.2.5 (Partida). *Dado dois times T_1 e T_2 . Uma partida p é definida por:*

$$p = \{T_1, T_2, \Delta t, \delta t, \langle Ref^0, X_{ob}^0, A_1^0, A_2^0 \rangle, \dots, \langle Ref^N, X_{ob}^N, A_1^N, A_2^N \rangle\}$$

uma sequência , em que:

Δt é o tempo de duração da partida;

δt é o tempo médio entre cada frame enviado pela SSL-Vision ao longo de Δt ;

$N \approx \frac{\Delta t}{\delta t}$ é número total de frames enviados pela SSL-Vision ao longo de Δt ;

Ref^i são os comandos enviados pelo módulo Referee-Box no instante i ;

X_{ob}^i são os dados enviados pelo módulo SSL-Vision no instante i ;

A_1^i são as ações executadas por T_1 no instante i ;

A_2^i são as ações executadas por T_2 no instante i .

Conforme citado em na Seção 2.1, δt normalmente é 16,6 ms. Um exemplo de A^i pode ser visto na Figura 4.9.

Definição 2.2.6 (Logs). *Dada uma partida p . O log de p é definidor por:*

$$\log(p) = \{p.\langle Ref^0, X_{ob}^0 \rangle, \dots, p.\langle Ref^N, X_{ob}^N \rangle\}$$

A principal diferença entre uma partida p e \log é o desconhecimento das ações que levaram aos movimentos observados nas partidas. Isso é um metadado importante quando se quer utilizar os dados de um jogo para aproximar o comportamento de algum time (i.e., $m_{reac\ ad}$). Este é um problema que está fora do escopo deste trabalho. Um exemplo de uma abordagem para esse problema pode ser encontrado em [4].

2.3 Discretização do jogo

Uma das dificuldades de se discretizar um sistema é a necessidade de criar uma abstração válida para o jogo, de modo que o que ocorra na simulação aconteça na prática caso a mesma situação simulada seja observada no mundo físico.

Essa abstração pode ser separada em duas etapas:

- Representação do jogo
- Execução do planejamento

Essas etapas são descritas a seguir.

2.3.1 Representação do jogo

2.3.1.1 Posse de bola

Para simplificar o modelo, foi introduzido o conceito de posse de bola. O robô que possui a bola é aquele que consegue interceptá-la no menor tempo possível. Como introduzir um modelo que considere a dinâmica do robô iria introduzir um custo computacional muito alto, foi utilizado um modelo simplificado. As simplificações são as seguintes (sómente para o cálculo do tempo da interceptação):

- A bola se move a velocidade constante
- Os robôs conseguem se mover em uma velocidade máxima $\hat{r}.vel_{max}$ em qualquer direção a qualquer momento

Dadas essas simplificações, o tempo para atingir o ponto de encontro pode ser calculado da seguinte maneira:

$$\hat{b}.vel \cdot t_{encontro} + \hat{b}.pos = \hat{r}.vel_{max} \cdot t_{encontro} + \hat{r}.pos \quad (2.1)$$

Trabalhando a equação acima, tem-se a seguinte equação do segundo grau:

$$0 = (\hat{r}.vel_{max}^2 - \hat{b}.vel^2) \cdot t^2 - \| \hat{r}.pos - \hat{b}.pos \|^2 - 2 \cdot \hat{b}.vel \cdot (\hat{b}.pos - \hat{r}.pos) \cdot t_{encontro} \quad (2.2)$$

Logo, tem-se:

$$t_{encontro} = \frac{-\hat{b}.vel \cdot (\hat{r}.pos - \hat{b}.pos) \pm \sqrt{\frac{\Delta}{4}}}{(\hat{r}.vel_{max}^2 - \hat{b}.vel^2)} \quad (2.3)$$

Onde $\Delta = 4 \cdot [(\hat{b}.vel \cdot (\hat{b}.pos - \hat{r}.pos))^2 + (\hat{r}.vel_{max}^2 - \hat{b}.vel^2) \cdot \| \hat{r}.pos - \hat{b}.pos \|^2]$.

2.3.1.2 Ações Consideradas

Cada um dos robôs em campo será modelado com as seguintes ações possíveis:

- Time no ataque (i.e., com a bola):

$$A_{atq} = \{Mover(r_i), Chutar(r_{com\ bola}), Passar(r_j, r_{com\ bola}) : r_i, r_j, r_{com\ a\ bola} \in Rob_{atq}\}$$

- Time na defesa (i.e., sem a bola): $A_{def} = \{Mover(r_i) : r_i \in Rob_{def}\}$

Note que o robô com a bola pode se mover. Entretanto isso é indesejável, já que a velocidade do passe é maior que a velocidade de movimentação do robô. Também existe uma restrição quanto à distância máxima que se pode conduzir a bola que, se não for respeitada, resulta em penalidade para o time do robô infrator.

O nível de complexidade das ações possíveis influí diretamente no número de ações que poderão ser consideradas a tempo de serem úteis para o jogo real. Por exemplo, caso não fosse considerada a ação de passe, esta ação ainda sim poderia acontecer na prática pela composição de outras ações. Entretanto, seriam necessários mais níveis de planejamento, uma vez que ela seria a composição de chutes e movimentações. Isso tem a contrapartida de reduzir o número de estados que podem ser simulados, uma vez que o tabuleiro é dinâmico e o jogo real ser simultâneo, e não sequencial. Isso fica mais evidente se fossem utilizadas somente as *skills* para o planejamento. A principal desvantagem disso é que o planejador teria que considerar aspectos como colisões e a orientação dos robôs no planejamento final. Além de ser ineficiente, coisas como posicionamento global dos robôs no campo não teriam estados suficientes na árvore do jogo para serem úteis.

Outra questão que se deve ter em mente ao se modelar as ações básicas dos robôs é definir ações muito complexas. Passando para a linguagem da arquitetura STP (*Skill, Tactic Play*), se fossem usadas *plays* no lugar de *tactics*, o espaço de jogadas seria muito limitado e poucas jogadas seriam geradas pelo computador. Mais detalhes da arquitetura STP podem ser encontrados em [5].

Como a complexidade cresce exponencialmente, onde o número de ações básicas é a base, isso não é um problema que pode ser tratado simplesmente com o aumento da

velocidade de processamento. Deve-se ajustar o nível de abstração de acordo com os resultados obtidos nos teste práticos.

2.3.2 Execução do planejamento

Esta etapa do modelo é responsável por converter o resultado do planejamento em comandos mais concretos. Conforme evidenciado na seção anterior, é nesta parte que o planejamento de trajetória deve ser levado em consideração. Esta é a parte que leva em consideração o modelo dinâmico do robô. Como isso é um problema complexo, com o objetivo de focar o escopo da pesquisa no planejamento de alto nível, será utilizada a arquitetura de controle do pyroboime.

2.4 Função Objetivo

A função objetivo f_U é definida pela seguinte expressão:

$$f_U = \sum c_i \quad (2.4)$$

Onde $\{c_i\}$ é o conjunto formado por custos. Os pesos de cada custo serão denotados por $p_{descrição}$. São eles:

1. Custo da Abertura do Gol
2. Correção da Abertura do Gol Devido a Movimentação da Bola
3. Distância Total das Ações $Mover(r)$
4. Distância Máxima das Ações $Mover(r)$
5. Mudança do Planejamento da Move Table
6. Custo do Ataque
7. Custo das Aberturas vistas por $r \in T_c$
8. Custo das Aberturas vistas por $r \in T_{ad}$
9. Custo da Defesa
10. Número de Receptores
11. Penalização por Proximidade do Gol do Adversário

12. Penalização por Proximidade

Além dos parâmetros listados acima, outros parâmetros afetam o comportamento do time:

1. Abertura do gol mínima para chute à gol
2. Número de ramificações

Esses parâmetros foram os primeiros utilizados no programa. Ao longo dos testes realizados, foram criados novos parâmetros. Entretanto, os parâmetros listados são suficientes para ilustrar a mudança no comportamento do time.

2.4.1 Custo da Abertura do Gol

O objetivo deste parâmetro é valorizar as aberturas maiores de acordo com a soma total das aberturas com relação a cada $r \in T$ e com aberturas maiores. O cálculo deste custo é feito da seguinte maneira:

$$c_{abertura\ gol} = taxa_{abertura\ total/max\ gol} \cdot \sum_{r_i \in T} abertura\ gol(bola, r) + (1 - taxa_{abertura\ total/max\ gol}) \cdot max\{abertura\ gol(bola, r) : r \in T\} \quad (2.5)$$

As aberturas são calculadas de acordo com a posição de um determinado robô r (obstáculo) e a posição atual da bola. Isso gera um conjunto de segmentos de reta $\{abertura\ gol(bola, r) : r \in T\}$.

Este custo afeta diretamente os seguintes custos:

- Custo do Ataque
- Custo da Defesa
- Custo das aberturas do gol vistas por $r \in T_c$
- Custo das aberturas do gol vistas por $r \in T_{ad}$

2.4.2 Correção da Abertura do Gol Devido a Movimentação da Bola

O objetivo deste parâmetro é incorporar a movimentação da bola pelo atacante no cálculo da abertura do gol. Isso é importante, pois defensores mais próximos são mais facilmente driblados que defensores mais distantes. Isso pode ser visualizado na figura 2.6, onde foi considerada uma variação de 15cm e de 0cm. Essa variação é medida na linha normal à reta formada pela bola e pelo robô em questão.

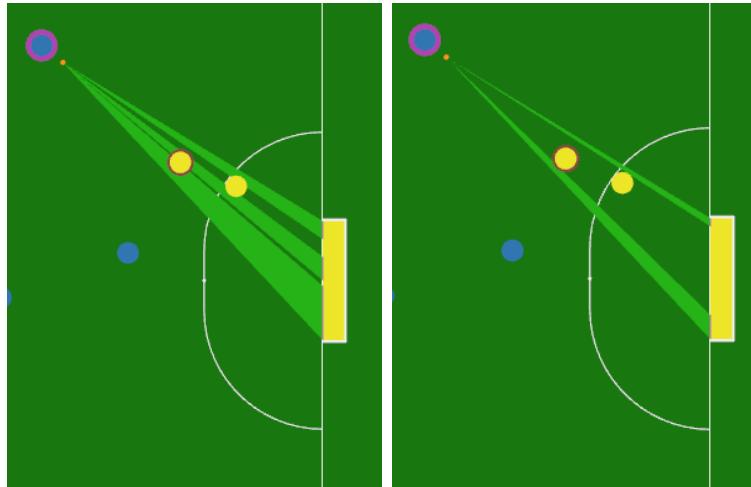


Figura 2.6 – Abertura do gol considerando-se uma variação de 15cm na posição da bola (esquerda) e sem variação (direita)

2.4.3 Distância Total das Ações $Mover(r)$

O objetivo deste parâmetro é valorar o custo da mudança de estado na função objetivo. Ele é formado pela soma das distâncias que cada ação $Mover(r)$ irá percorrer partindo das respectivas posições do estado atual.

$$c_{dist\ total\ mover} = p_{dist\ total\ mover} \cdot \sum_{r_i \in T_c} \|pos_{r_i} - Mover(r_i)\| \quad (2.6)$$

2.4.4 Distância Máxima das Ações $Mover(r)$

O objetivo deste parâmetro é incorporar o custo da mudança de estado na função objetivo. Ele é formado pela ação $Mover(r)$ do planejamento atual com maior deslocamento.

$$c_{dist\ max\ mover} = p_{dist\ max\ mover} \cdot \max\{r_i \in T_c : \|pos_{r_i} - Mover(r_i)\|\} \quad (2.7)$$

2.4.5 Mudança do Planejamento da Tabela de Decisão

O objetivo deste parâmetro é evitar mudanças grandes no planejamento da ação $Mover(r)$. Uma das razões para isso é evitar um modelo dinâmico exato neste nível de planejamento, já que isso aumentaria muito o custo computacional desta etapa do planejamento e, consequentemente, reduziria o número de simulações possíveis. Por essas razões mudanças no planejamento das ações $Mover(r)$ inicialmente foram penalizadas de acordo com a distância euclidiana entre o $Mover_p(r)$ planejado anteriormente e o $Mover_m(r)$ modificado, conforme a equação:

$$c_{mudança\ mover} = p_{mudança\ mover} \cdot \max \left\{ \sum_{r_i \in T_c} \|Mover(r_i) - Mover_p(r_i)\| \right\} \quad (2.8)$$

Isso permite que sejam selecionados $Mover_m(r)$ mais próximos do $Mover_p(r)$, refinando assim o planejamento anterior. Entretanto, um parâmetro mais significativo é o valor absoluto do ângulo entre essas ações e a posição de r , $\|ang(r, Mover_m(r), Mover_p(r))\|$. Dessa maneira, tem-se que este custo é computado da seguinte maneira:

$$c_{mudança\ mover} = p_{mudança\ mover} \cdot \max \left\{ \sum_{r_i \in T_c} \|ang(r, Mover_m(r), Mover_p(r))\| \right\} \quad (2.9)$$

2.4.6 Custo do Ataque

Este custo valoriza situações nas quais o time em questão possui o domínio da bola (descrito na Subseção 2.3.1).

$$c_{ataque} = p_{ataque} \cdot abertura\ gol_{ad} \quad (2.10)$$

Onde $abertura\ gol_{ad}$ é o valor da abertura do gol adversário visto pelo robô que tem domínio da bola.

2.4.7 Custo da Defesa

Quando não se tem o domínio da bola a abertura do gol do time em questão visto pelo robô do time adversário que tem a bola é utilizado como penalização adicional.

$$c_{defesa} = p_{defesa} \cdot \sum_{r_i \in T_{ad}} abertura\ gol_c(r_i) \quad (2.11)$$

2.4.8 Custo das Aberturas do Gol Vistas por $r \in T_c$

Este custo tem o objetivo de valorizar configurações nas quais os robôs que não tem a bola possuam visada para o gol do time adversário. Isso é desejável, já que permite que um gol seja feito caso o robô que esta com a posse de bola execute um passe para um dos robôs em questão.

$$c_{abertura\ gol\ ad} = p_{abertura\ gol_{ad}} \cdot \sum_{r_i \in T_c} abertura\ gol_{ad}(r_i) \quad (2.12)$$

2.4.9 Custo das Aberturas do Gol Vistas por $r \in T_{ad}$

Este custo é o análogo do custo anterior, mas para o time adversário. Ele visa penalizar situações nas quais os robôs adversários tenham grande visada (i.e., abertura) para o gol.

$$c_{bloqueio\ gol} = -p_{bloqueio\ gol} \cdot \sum_{r_i \in T_{ad}} abertura\ gol_{ad}(r_i) \quad (2.13)$$

2.4.10 Número de Receptores

Este custo visa valorizar situações nas quais existam vários robôs que possam receber passe.

$$c_{receptores} = p_{receptores}.num\{r_{receptor_i}\} \quad (2.14)$$

2.4.11 Penalização por Proximidade do Gol do Adversário

Esta penalização visa evitar que os robôs que estão no ataque se concentrem dentro da área do time adversário. Caso este parâmetro não seja adicionado, é esse o comportamento resultante do custo das aberturas vistas pelos robôs do time em questão. A partir de uma determinada distância, é adicionada uma parcela de penalização na função objetivo.

$$c_{penalização\ prox} = -p_{penalização\ prox} \sum num\{r_{perto}\} \quad (2.15)$$

2.4.12 Abertura Mínima para Chute a Gol

Este parâmetro é o ângulo total da abertura do gol para permitir o chute pelo atacante. Ele é relevante, pois afeta o comportamento do atacante e, como consequência, o comportamento do time durante a posse de bola. Quando a abertura do gol é menor que este valor, o atacante tem ação de chutar em direção ao goal.

2.4.13 Número da Ramificação

Este parâmetro é o número de possibilidades que serão simulados em cada iteração do algorítimo de avaliação. Ele influencia o tempo de resposta do planejamento.

2.5 Estratégia de Busca

A estratégia de busca utilizada para gerar os novos estados a partir do estado atual x é apresentada na Figura 2.7.

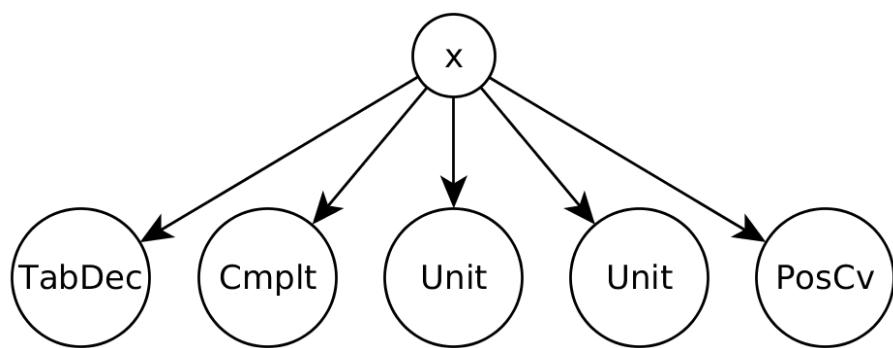


Figura 2.7 – Ilustração da distribuição dos estados considerados no planejamento

Os tipos de ações que são geradas são:

- *Tabela de decisão*: Ação escolhida no último planejamento;
- *Único*: Ação gerada a partir da tabela de decisão onde a ação de um único robô foi modificada;
- *Total*: Ação gerada a partir da tabela de decisão onde a ação do time completo foi modificada;
- *Posição Chave*: Ação gerada utilizando posições chave (vide Seção 2.5.3).

A ação selecionada é aquela que gera o maior valor de f_U . O método da decisão do gradiente foi implementado para melhorar o planejamento. Calculando a série de Taylor para f_U em torno de x_k , tem-se:

$$f_U(x) = f_U(x_k) + \nabla f_U(x_k)^T(x - x_k) + O(\|x - x_k\|^2) \quad (2.16)$$

Calculando no ponto $x_k + \gamma d$, com $\|d\| = 1$, tem-se:

$$f_U(x_k + \gamma d) = f_U(x_k) + \gamma \nabla f_U(x_k)^T d + O(\gamma^2) \quad (2.17)$$

Para γ suficientemente pequeno, tem-se que:

$$\begin{aligned} f_U(x_k + \gamma d) - f_U(x_k) &\approx \gamma \nabla f_U(x_k)^T d \\ f_U(x_k + \gamma d) - f_U(x_k) &> 0 \Rightarrow \gamma \nabla f_U(x_k)^T d > 0 \end{aligned} \quad (2.18)$$

Logo, tomando $d = \frac{\nabla f_U(x_k)}{\|\nabla f_U(x_k)\|}$ e $\gamma > 0$, pode-se aumentar o valor de f_U [6].

Como a taxa das soluções é fundamental para se controlar os robôs, pode-se aplicar esse método de duas maneiras:

- somente na melhor ação selecionada
- em todas as ações consideradas

A segunda opção reduz a taxa á zero pacotes por segundo, o que prejudica o controle e invalida o planejamento, já que o estado do jogo se modifica continuamente devido a ação do time adversário.

2.5.1 Distribuição da Ação $Mover(r)$

Para gerar as ações do tipo $Mover(r)$, são utilizadas três distribuições uniformes circulares. Essas distribuições tem centro na posição $r.pos$. São utilizadas para concentrar localmente as movimentações de cada robô.

2.5.2 Tabela de Decisão

A tabela de decisão contém a memória das últimas decisões $a_{anterior} \in A_c$ de todos os robôs de T_c . Isso é incorporado na próxima rodada do algorítimo, conforme ilustrado na Figura 2.7.

Devido a movimentação dos robôs em Rob_{ad} , somente as ações $Mover(r_i)$ e $Chutar(r_{com\ bola})$ são guardadas integralmente. No caso da ação $Passar(r_j, r_{com\ bola})$, devido a restrição de o robô que recebe interceptar a bola antes dos robôs em Rob_{ad} , o cálculo do receptor r_j é feito a cada rodada. O receptor anterior só entra no custo da mudança (vide Seção 2.4.5).

Caso a posse de pola passe para o time adversário, o robô que possuía a bola anteriormente fica com a sua última ação do tipo $Mover(r_i)$.

2.5.3 Posições Chave

Devido ao grande número de possibilidades para as posições dos robôs, somente a geração de posições aleatórias não gera bons resultados para as ações do tipo $Mover(r_i)$. Portanto, para direcionar a busca, foi desenvolvida uma maneira de se sugerir posições para as ações em questão com o auxílio da interface gráfica. A Figura 2.8 mostra a sugestão de uma barreira. Já na Figura 2.9 são sugeridas posições laterais.



Figura 2.8 – Sugestão de uma barreira (em rosa)



Figura 2.9 – Sugestão de posições laterais (em rosa)

Com o objetivo de verificar a utilidade das posições chave, foi criada uma maneira de identificar se uma posição chave foi utilizada.

As posições chave são utilizadas das seguinte maneira:

- Caso um número de posições inferior ao número de robôs sejam sugeridas, são criadas ações aleatórias para completar a sugestão;
- As posições sugeridas são atribuídas para o robô mais próximo da posição sugerida. Isso é feito na ordem em que as posições foram sugeridas.

3 Arquitetura do *Software*

O *software* desenvolvido é uma ferramenta, e ao longo deste capítulo ele é referido simplesmente por "a ferramenta". Este capítulo descreve como a ferramenta interage com os outros *softwares* em uso na competição. Os seguintes são de relevância para o entendimento da arquitetura escolhida:

- ssl-vision: desenvolvido pela comunidade e de uso oficial na competição para processamento das imagens da câmera nas partidas e distribuição pela rede dos dados processados (estado do jogo);
- grSim: desenvolvido pela comunidade e amplamente usado pelas equipes para simular o ambiente das partidas, o protocolo usado para enviar o estado pela rede é idêntico ao do ssl-vision;
- pyroboime: também chamado de core desenvolvido pela RoboIME, atualmente provê uma camada de abstração sobre a comunicação com o ambiente de jogo (real ou simulado) incluindo redução de ruído, planejamento de trajetória e controle.

A ferramenta está implementada em C++11 (revisão recente de C++ suportada pela maioria dos compiladores modernos).

3.1 Comunicação com Componentes Externos

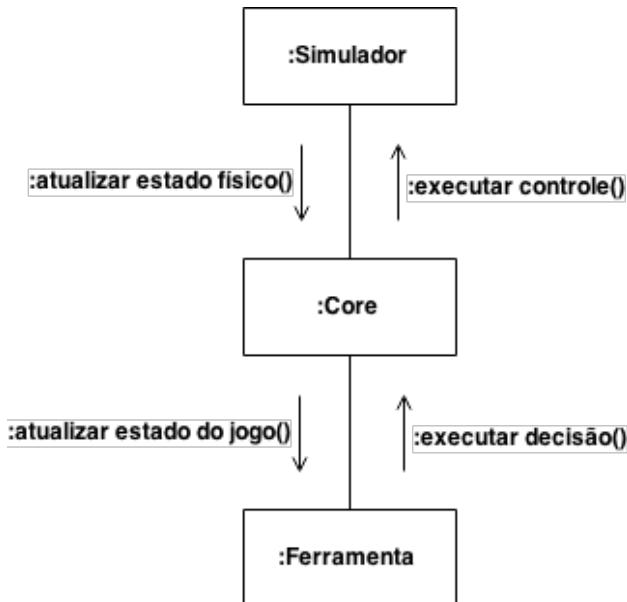


Figura 3.1 – Diagrama de comunicação entre os componentes.

A Figura 3.1 representa como os componentes se comunicam. A ferramenta se comunica apenas com o pyroboime para aproveitar todas as funcionalidades necessárias que fogem ao escopo desse trabalho. Por isso o foco do desenvolvimento está concentrado em atender diretamente os objetivos.

As mensagens trocadas entre a ferramenta e o core (*pyroboime*) são codificadas com o *Protobuf* [7], uma biblioteca bem estabelecida que também é usada nos protocolos oficiais da competição. Desse modo o *overhead* de comunicação é baixo, não é necessário introduzir uma dependência ou codificação manual no core e é possível que versões futuras sejam retro-compatíveis.

São dois tipos de mensagens:

- atualização do estado: sentido do core para a ferramenta, codificadas com a mensagem *UpdateMessage*, descreve todas as informações necessárias para criar um estado novo;
- comando de ações: sentido ferramenta para o core, codificadas com a mensagem *CommandMessage*, descreve a ação que cada agente (robô) deve realizar.

As mensagens são transmitidas em um socket ZMQ (*ZeroMQ* [8], uma biblioteca de transmissão de dados na rede) visando a extensibilidade da ferramenta, pois com tal bi-

blioteca é possível distribuir mensagens entre vários nós de forma confiável (característica desejável para um sistema distribuído) além de prover confiabilidade e auto-reconexão para o uso atual.

O modo de transmissão é o *request-reply*, em que a ferramente age como servidor respondendo as requisições do core. Esse modelo é composto por atualizações cuja a resposta deve ser o comando a ser executado naquele estado requisitado. Na prática a ferramenta irá transmitir o comando mais recente, que se encontra em seu *buffer*, descrito brevemente no Seção 3.2.

3.2 *Threads* do Sistema

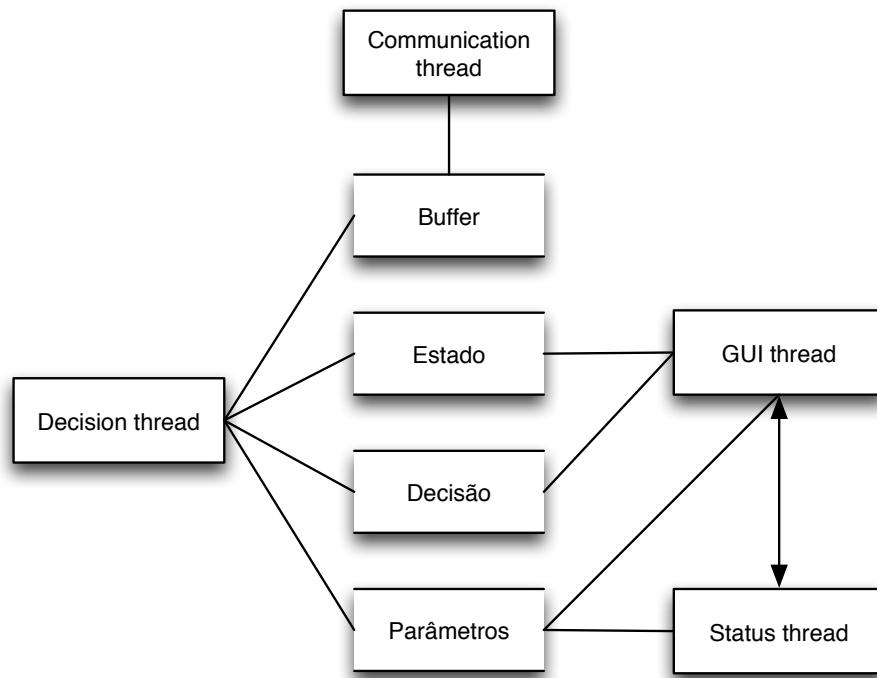


Figura 3.2 – Diagrama de relação entre *threads* e dados.

A ferramenta está implementada em quatro *threads* fixas de acordo com a Figura 3.2. A separação foi motivada por:

- interface gráfica responsiva ao usuário;
- taxa de tomada de decisão poder ser mais lenta que a taxa de atualização do estado, portanto a comunicação tem sua própria *thread* que escreve e lê de um buffer compartilhado pela *thread* de tomada de decisão;

- certas informações devem ser coletadas periodicamente para atualizar alguns parâmetros e exibidas para o usuário.

3.3 Interface Gráfica

É importante destacar o fato que a interface gráfica tem um papel muito importante na ferramenta. Ela possibilita que um usuário não programador modifique, em tempo de execução, um conjunto de configurações que modifica, com uma grande amplitude, o comportamento do time controlado. A interface visa prover uma representação de simples compreensão do estado do jogo e de como as várias configurações podem afetar o comportamento. Maiores detalhes sobre a interface em si são discutidos na Seção 4.1. Esta seção aborda a arquitetura implementada que provê a interface.

A Figura 3.3 mostra como os componentes interagem. Em mais detalhes, esses componentes são:

- GLFW: biblioteca que abstrai as APIs nativas de manipulação de janelas e de eventos de mouse e teclado. Além de fornecer contextos OpenGL. Seu uso é importante pois permite que a ferramenta seja compilada sem alteração para diferentes sistemas operacionais.¹ (biblioteca disponível em [9])
- OpenGL: API de renderização 2D e 3D. Seu uso se restringe a renderizações no plano (2D). É usada para desenhar todos os objetos que representam estados, decisões e alguns artefatos auxiliares, além de ser usado para prover o *backend* de renderização do ImGui. (API disponível em [10])
- ImGui: Biblioteca leve de *widgets* usada para criar janelas internas, botões, barras de configuração deslizantes, caixas de seleção, entre outros *widgets*. Como não possui dependências é necessário prover um *backend* de renderização. Outra vantagem é ser orientada a "modo imediato", na prática isso significa que, por exemplo, desenhar um botão é uma única chamada de função, sem a necessidade de estruturação e gerenciamento manual dos *widgets*. (biblioteca disponível em [11])

¹ Fato que ocorreu durante todo o desenvolvimento. A ferramenta foi desenvolvida principalmente no OS X, e em parte no Ubuntu (distribuição de Linux). Além disso, um serviço de integração (Travis CI) foi usado para validar a compilação no Ubuntu. Embora desejável, não houve tentativa de compilação no Windows.

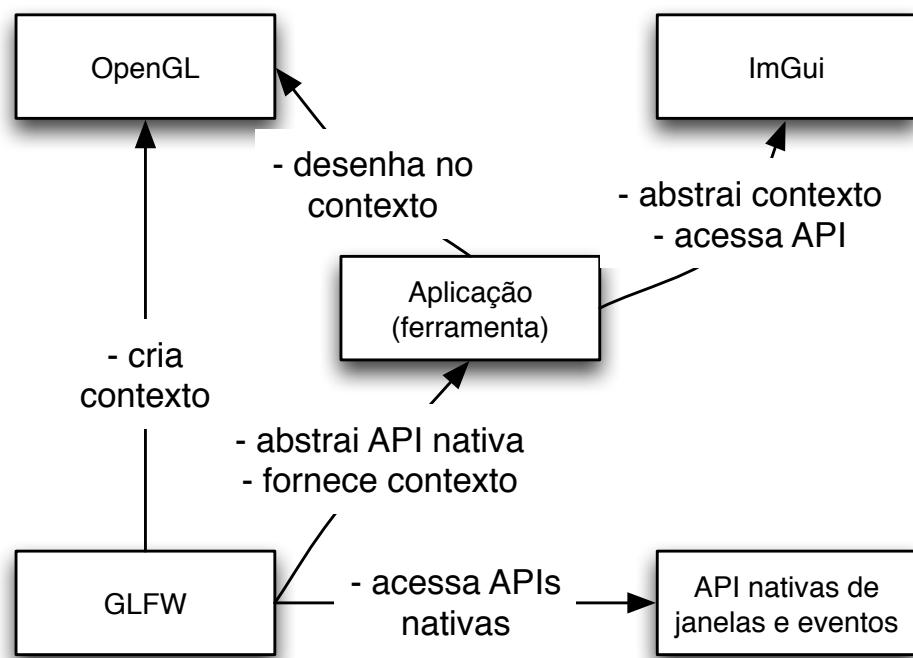


Figura 3.3 – Diagrama de relação entre componentes gráficos

4 Resultados

4.1 Interface Gráfica



Figura 4.1 – Aparência geral e representação do campo

A ferramenta representa o estado atual do jogo desenhando o campo, os robôs e a bola, como pode ser visto na Figura 4.1. O campo não faz parte do estado mas é necessário para ter uma referência visual das posições.

Para visualizar situações com mais detalhes a ferramenta permite zoom e arraste do campo (Figura 4.2).

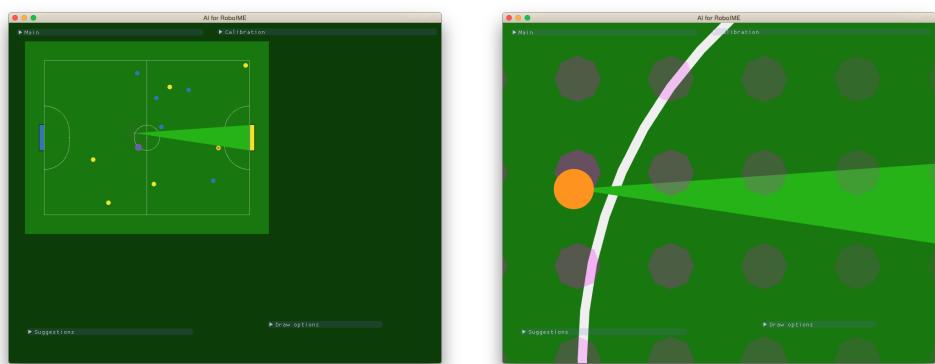


Figura 4.2 – Zoom e arraste do campo

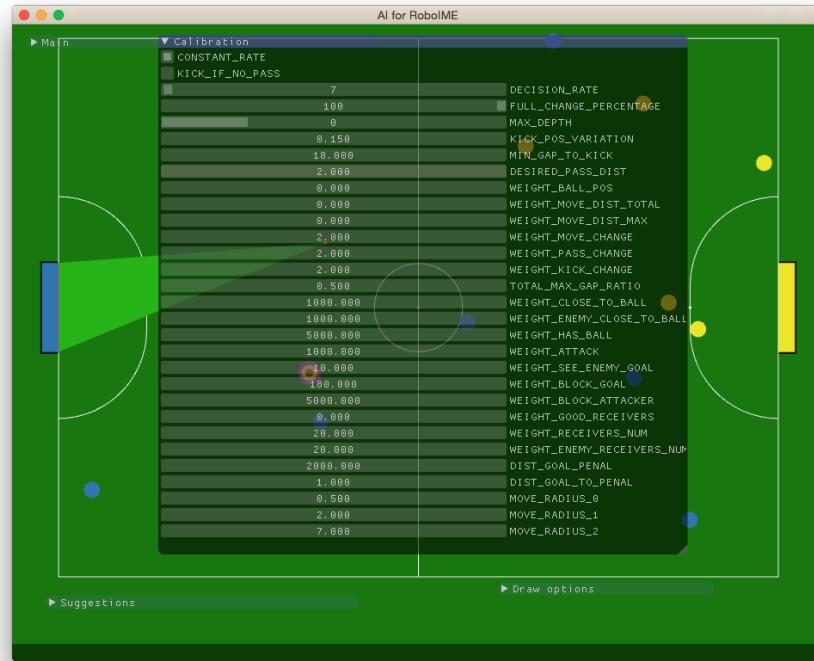


Figura 4.3 – Parâmetros configuráveis

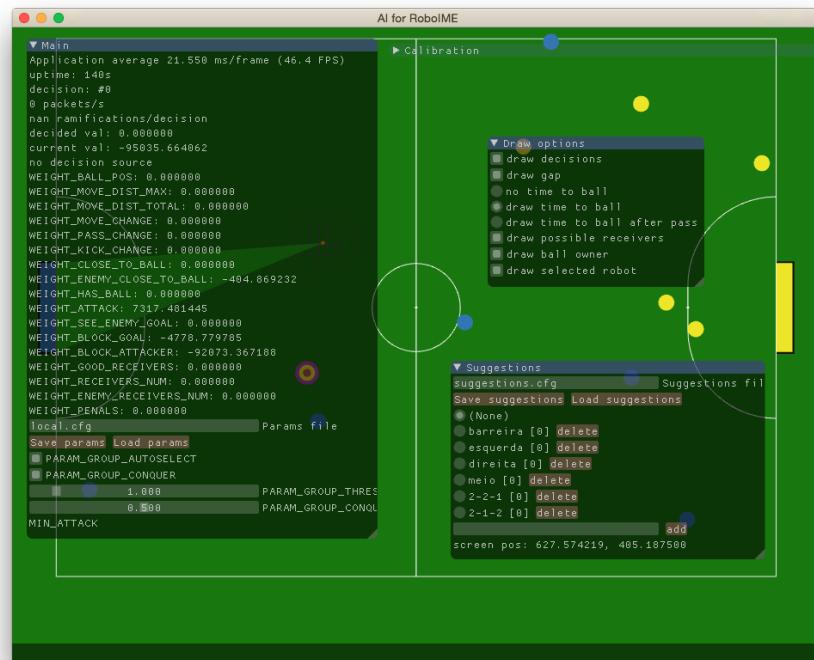


Figura 4.4 – Controles

Existem quatro abas disponíveis: *Main*, *Calibration*, *Suggestions*, *Draw options*. As abas são retráteis e translúcidas para que as mudanças de configurações possam ser observadas facilmente. Na Figura 4.1, por exemplo, todas as estão minimizadas, na Figura 4.3 é mostrada a aba *Calibration*, as outras três podem ser vistas na Figura 4.4.



Figura 4.5 – Representação do tempo para chegar na bola

A interface também possui indicadores para o tempo de chegar na bola, o robô que chegará primeiro na bola e os robôs que podem receber um passe.

O indicador do robô que chegará primeiro, às vezes chamado de "dono da bola", é representado com um círculo rosa em torno do robô.

O indicador para o tempo de chegar na bola está representado na Figura 4.5 como um campo de pontos translúcidos ao redor da bola, quanto maior o diâmetro do ponto menor o tempo para se chegar na bola a partir dali. Na imagem à direita, a bola está parada e o robô amarelo é o dono, já na imagem da esquerda a bola está em movimento, aproximadamente na direção do robô azul, que nesse caso é o dono da bola.

As decisões tomadas possuem representação gráfica, desenhando individualmente cada ação. As ações de passe podem ser vistas na Figura 4.6 uma linha tracejada da bola para o receptor do passe, nessa mesma figura está exemplificado a representação dos robôs que podem receber passe.

Similarmente a Figura 4.7 apresenta a representação da ação de chute. E por fim, o último tipo de ação e mais comum, a de movimentação, pode ser vista na Figura 4.8.

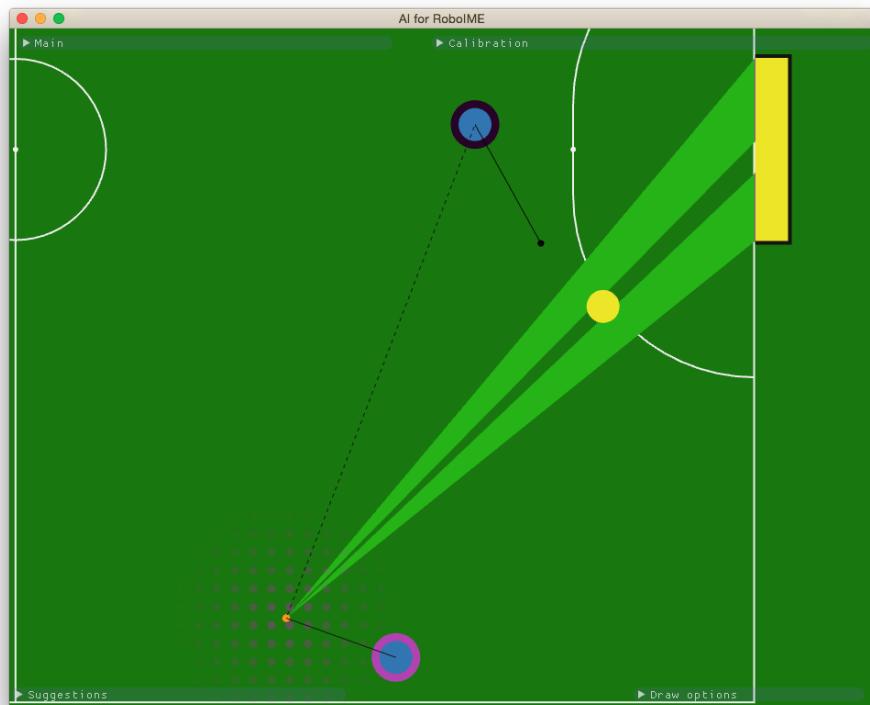


Figura 4.6 – Representação de ação de passe

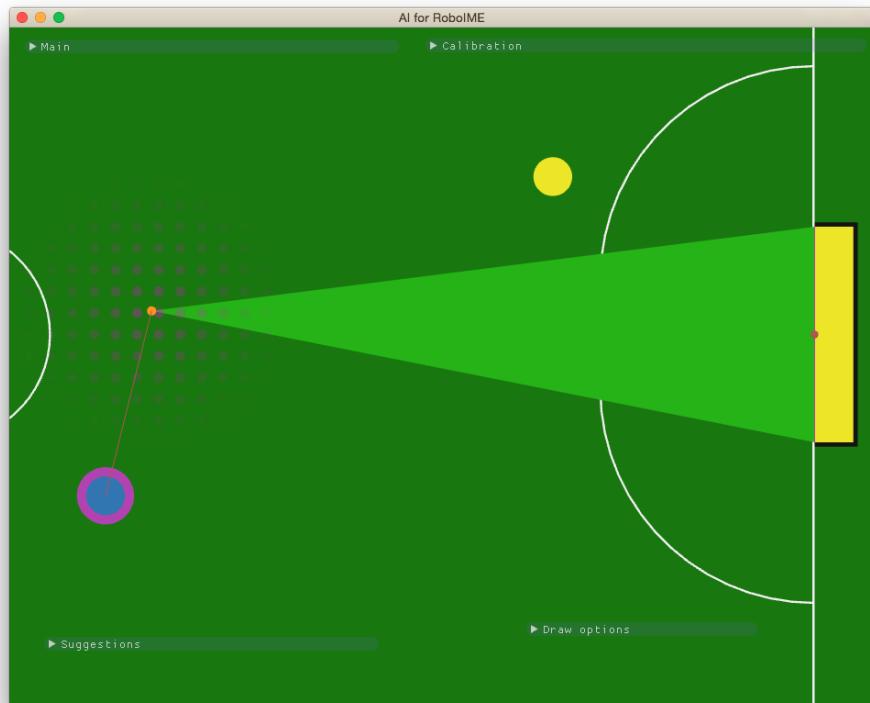


Figura 4.7 – Representação de ação de chute



Figura 4.8 – Representação das ações de movimentação

4.2 Influência dos Parâmetros no Comportamento do Time

Nesta secção são apresentados os diferentes comportamentos que podem ser obtidos através da modificação dos parâmetros apresentados no Capítulo 2. Cada parâmetro é modificado e os resultados na mudança do planejamento são evidenciados.

Os valores iniciais dos parâmetros modificados nos experimentos realizados são:

1. Distância de movimentação da bola: 15cm
2. Abertura mínima para chute à gol: 18.0 graus
3. $p_{abertura\ gol_{ad}} = 10$
4. $p_{abertura\ gol_c} = 180$

As Figuras 4.9 e 4.10 apresentam o planejamento em um ambiente de ataque e defesa, respectivamente, com os parâmetros iniciais apresentados anteriormente.



Figura 4.9 – Planejamento com os parâmetros iniciais no ataque

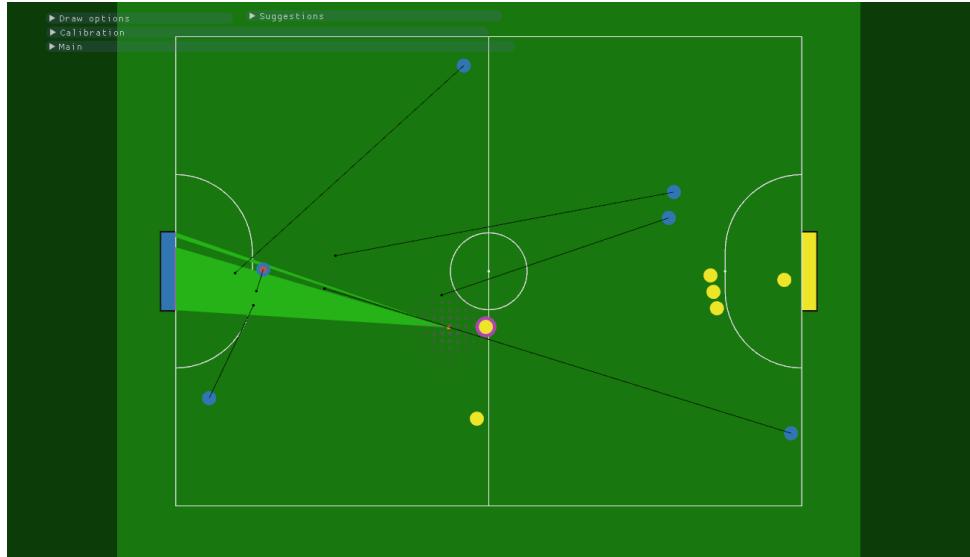


Figura 4.10 – Planejamento com os parâmetros iniciais na defesa

4.2.1 Correção da Abertura do Gol Devido a Movimentação da Bola

Somente o ajuste de movimentação da bola foi anulado. Os resultados no planejamento são apresentados na Figura 4.11. Conforme pode ser observado, a sombra dos robôs é maior, reduzindo assim abertura do gol. Logo, poucos robôs são necessários para bloquear o gol. Isso gerou mais ações direcionadas para posições de ataque.

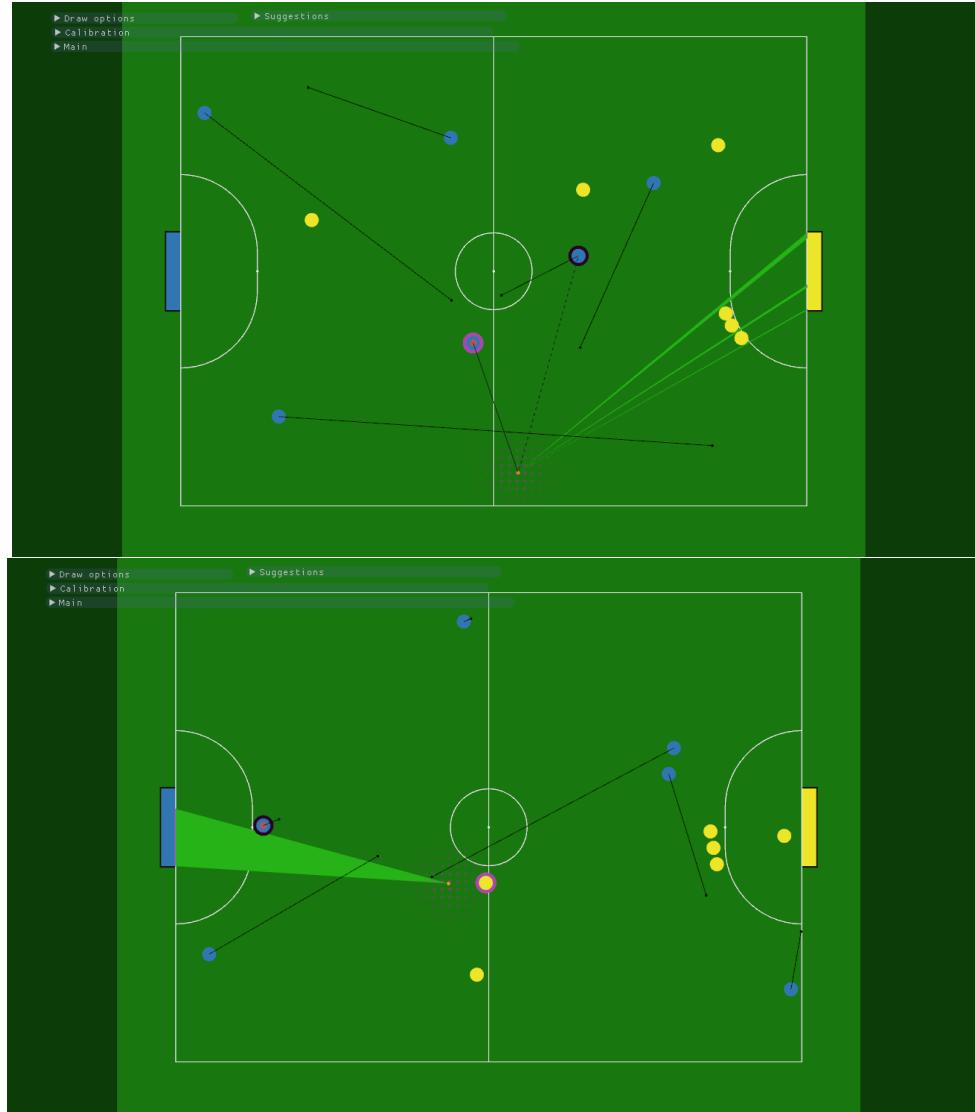


Figura 4.11 – Planejamento com os parâmetros iniciais e sem ajuste de movimentação da bola em ambiente de ataque (acima) e defesa (abaixo)

Depois esse ajuste foi alterado para 0.24. Os resultados em ambiente de ataque e defesa são apresentados na Figura 4.12. A sombra de cada robô é menor, resultando em mais robôs próximos ao gol para reduzir a abertura do gol vista pelos robôs adversários.

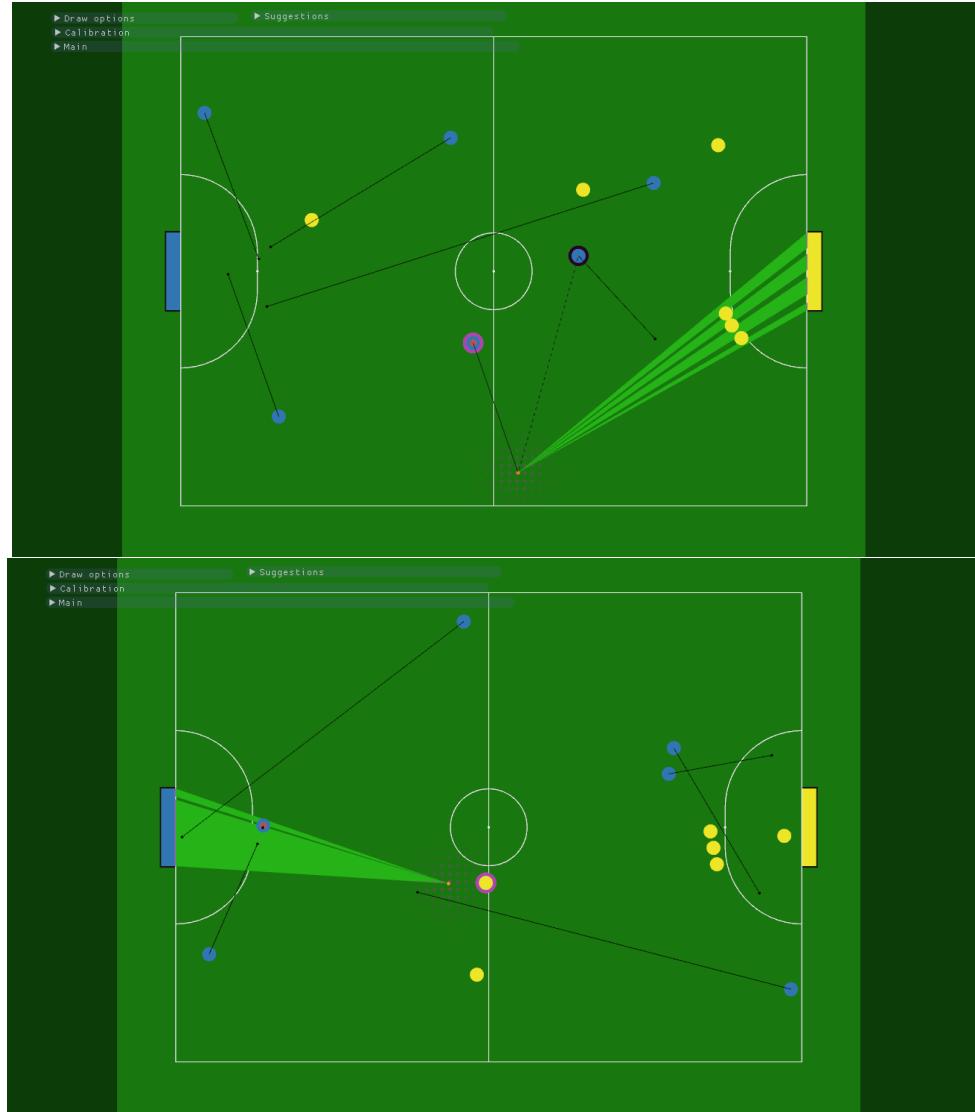


Figura 4.12 – Planejamento com os parâmetros iniciais e o ajuste de movimentação da bola ajustado para 0.24 em ambiente de ataque (acima) e defesa (abaixo)

4.2.2 Custo das Aberturas vistas por $r \in T_c$

Somente o peso do custo das aberturas do gol vistas pelos robôs do time foi alterado para 1000. Os resultados no planejamento são apresentados na Figura 4.13. Fica evidente em ambos os ambientes de ataque e defesa que os robôs se posicionaram o mais próximo possível do gol de T_{ad} , se limitando apenas pela penalização por proximidade do gol adversário.

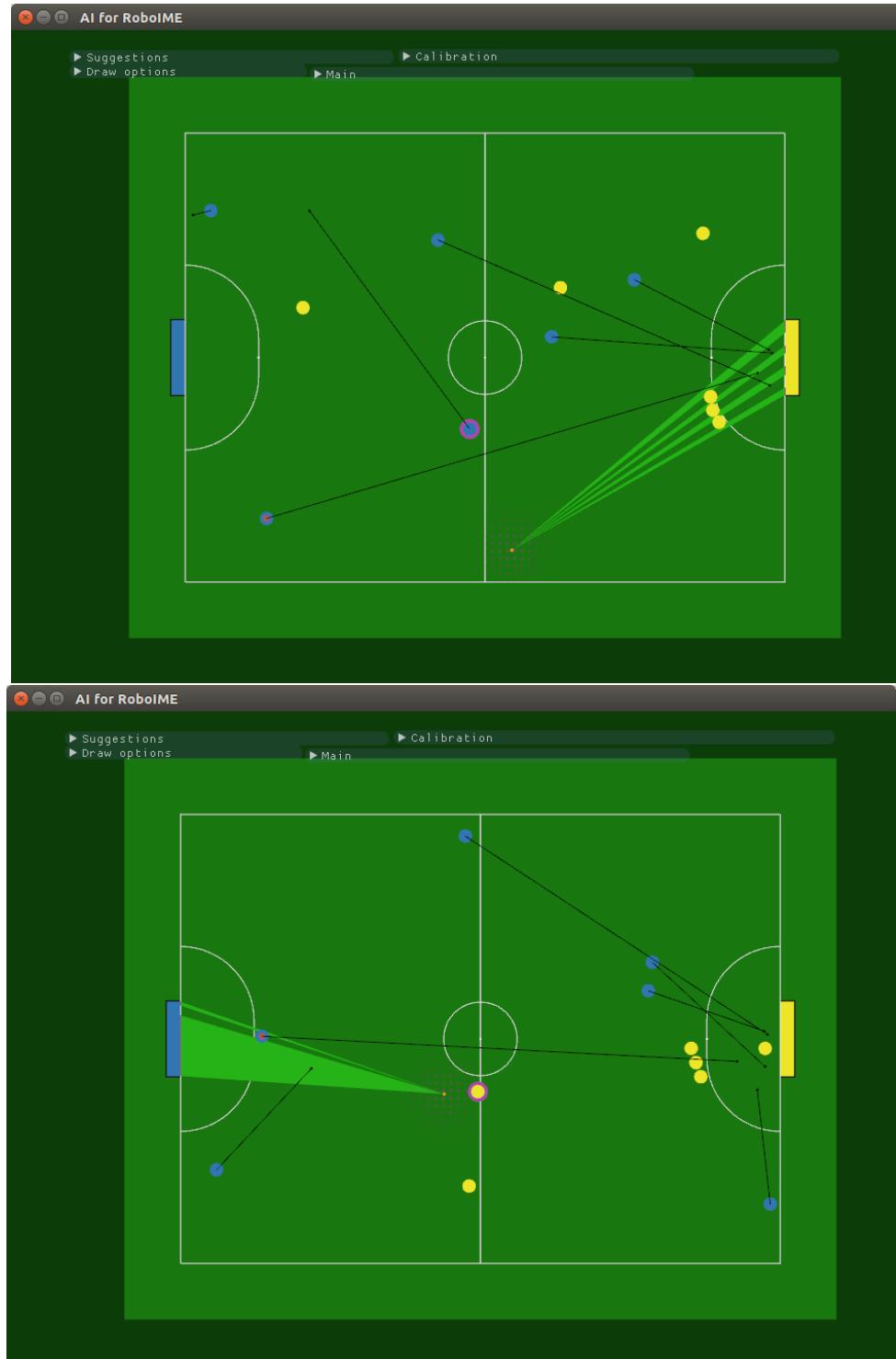


Figura 4.13 – Planejamento com os parâmetros iniciais e com o peso do custo das aberturas do gol vistas pelos robôs do time igual a 1000. No ataque (acima) e na defesa (abaixo)

4.2.3 Custo das Aberturas vistas por $r \in T_{ad}$

Somente o peso do custo das aberturas do goal vistas pelos robôs do time adversário alterado para zero. Os resultados no planejamento são apresentados na Figura 4.14. Houve uma redução nas posições de defesa com e sem posse de bola.

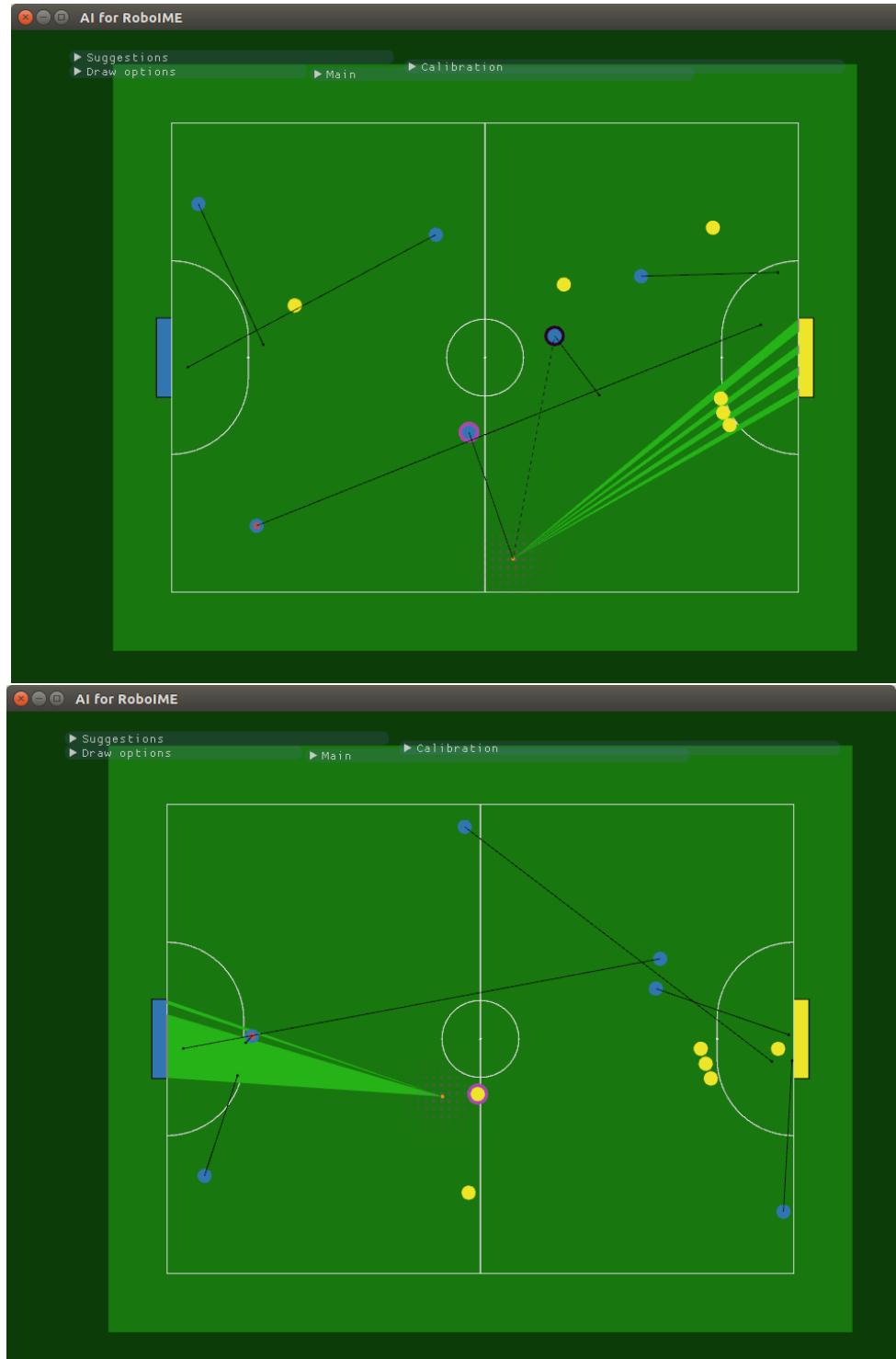


Figura 4.14 – Planejamento com os parâmetros iniciais e com o custo das aberturas do goal vistos pelos robôs do time nulo. No ataque (acima) e na defesa (abaixo)

4.2.4 Abertura mínima para chute à gol

Somente o peso do parâmetro que permite chute à gol foi alterado para 5. Os resultados no planejamento são apresentados na Figura 4.15. Conforme pode ser visto, somente no ambiente de ataque houve uma mudança grande, gerando uma ação de chute no lugar de uma ação de passe.

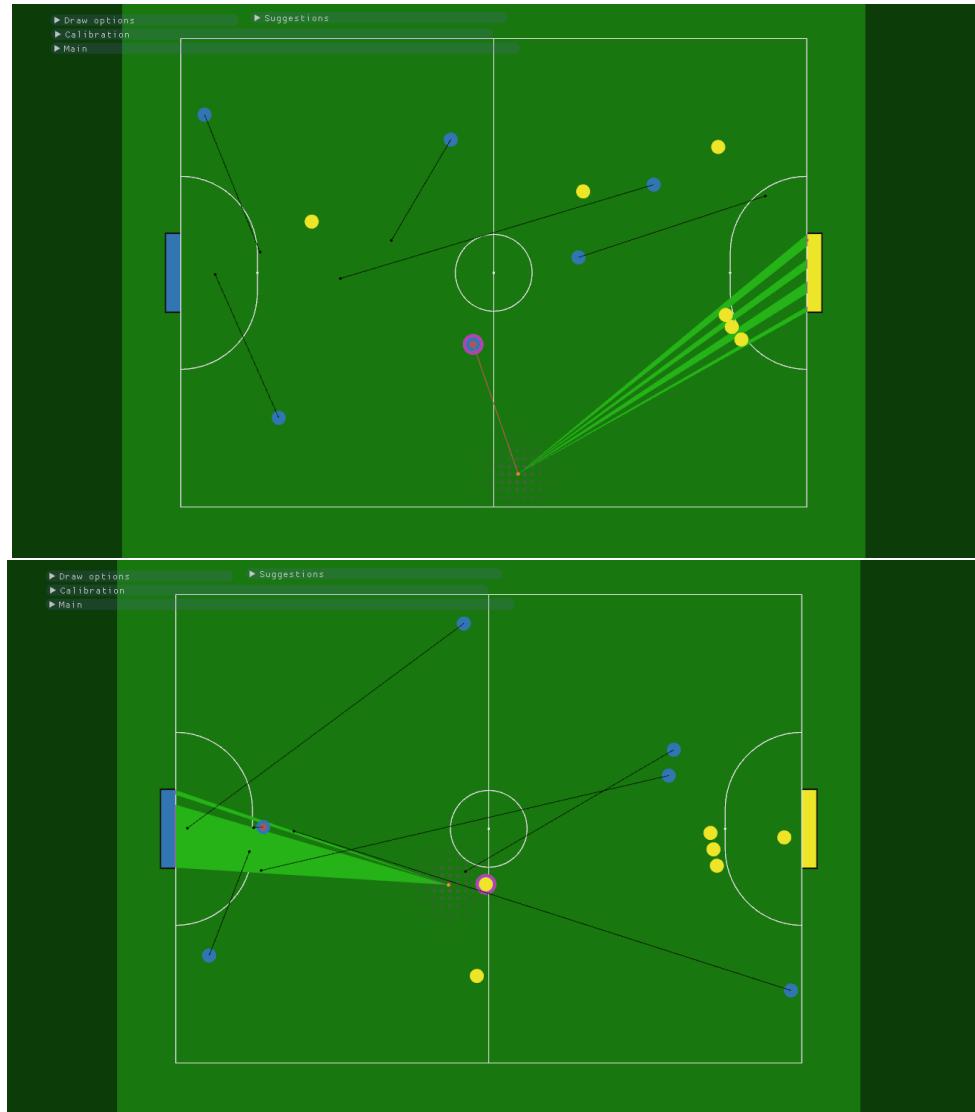


Figura 4.15 – Planejamento com os parâmetros iniciais e com a abertura mínima para chute alterado para 5. No ataque (acima) e na defesa (abaixo)

4.3 Goleiro

Conforme pode ser visto nos experimentos anteriores, somente em um experimento o goleiro não surgiu. A penalização foi desconsiderada para um único robô de T_c . Isso é interessante, pois não foi necessário restringir o movimento deste robô, permitindo que este se mova conforme a valoração de f_U .

5 Conclusão

Este trabalho objetivou desenvolver uma ferramenta de representação comportamental baseada em otimização para futebol de robôs, com meta intermediária de criar um modelo discreto sequencial para o problema do futebol de robôs.

Foi desenvolvido um modelo abstrato do futebol de robôs no Capítulo 2. Esse modelo foi base para o programa apresentado no Capítulo 3. Essa ferramenta utiliza uma combinação de gradiente da função objetivo, buscas aleatórias e posições chaves para selecionar jogadas ótimas dentro do tempo disponível para o planejamento.

A ferramenta criada atingiu os objetivos desejados, gerando uma variedade de comportamentos através da modificação dos parâmetros da função objetivo, conforme mostrado no Capítulo 4. Também foi desenvolvida com sucesso uma interface gráfica que permite modificar esses parâmetros em tempo de execução.

O trabalho atual permite que seja criado um time superior apenas modificando os parâmetros da função objetivo ou adicionando novos custos à função objetivo. Também é possível utilizar esse modelo para aplicar outros métodos de otimização.

Como trabalhos futuros, sugere-se desenvolver um sistema automático para realizar as partidas e avaliar o desempenho do time nessa partida. Isso permitiria que algorítimos de otimização também fossem utilizados para melhorar os parâmetros. Para isso, é necessário um juiz automático para permitir jogos não supervisionados.

Referências

- [1] S. Zickler, T. Laue, O. Birbach, M. Wongphati, and M. Veloso, “SSL-Vision: The Shared Vision System for the RoboCup Small Size League,” *RoboCup 2009: Robot Soccer World Cup XIII*, pp. 425–436, 2009.
- [2] A. Basu, *Computer vision : systems, theory, and applications*. Singapore River Edge, NJ: World Scientific, 1993.
- [3] T. A. N. Amaral and D. F. de Almeida, “Robôs autônomos cooperativos: Small size league.” Monografia de Iniciação à Pesquisa ao Curso de Graduação em Engenharia de Computação, 2011.
- [4] D. L. Vail and M. M. Veloso, “Feature selection for activity recognition in multi-robot domains,” 2008.
- [5] B. Browning, J. Bruce, M. Bowling, and M. Veloso, “Stp: skills, tactics, and plays for multi-robot control in adversarial environments,” *IEEE Journal of Control and Systems Engineering*, 2004.
- [6] A. D. Belegundu and T. R. Chandrupatla, *Optimization Concepts and Applications in Engineering*. Berlin New York: Cambridge University press, 2011.
- [7] “Protocol buffers api reference.” <https://developers.google.com/protocol-buffers/docs/reference/overview>. Acessado em 13/05/2015.
- [8] “ZMQ guide.” <http://zguide.zeromq.org/page:all>. Acessado em 13/05/2015.
- [9] “GLFW documentation.” <http://www.glfw.org/docs/3.0.4/quick.html>. Acessado em 13/05/2015.
- [10] “OpenGL documentation.” <https://www.opengl.org/sdk/docs/man2/>. Acessado em 13/05/2015.
- [11] “ImGui github.” <https://github.com/ocornut/imgui>. Acessado em 13/05/2015.

Apêndices

APÊNDICE A – API PÚBLICA

Uma API consiste nas estruturas e funções (ou também métodos) que estão disponíveis para o programador. A ferramenta está programada de forma que também é possível fazer o uso programático de suas funcionalidades.

Esta seção descreve com detalhes cada estrutura e função presente na API da ferramenta. É um objetivo desta seção documentar o código da ferramenta para possibilitar a evolução dessa em projetos futuros ou integração em outros projetos.

Todas as estruturas e funções estão expostas em *headers C++*, porém esta documentação irá tratar apenas das funcionalidades e organização sem se prender à sintaxe ou modo de uso na linguagem em que está programada.

Enumeração *ActionType*

Representa o tipo da ação.

Alternativas

- *NONE*: nenhuma ação;
- *MOVE*: ação de movimentação;
- *PASS*: ação de passe;
- *KICK*: ação de chute.

Estrutura *Action*

Representa a ação individual a ser executada por um robô, não está amarrada à um robô, essa associação deve ser feita pelo usuário da estrutura, isso permite uma associação implícita baseada na ordem do vetor ou outros tipos de otimizações, por esse motivo algumas funções recebem um Id de robô para saber qual robô usar em certas circunstâncias.

Atributos

- *type*: tipo (*ActionType*), por padrão é do tipo NONE;
- *move_pos*: posição destino da movimentação;

- *kick_pos*: posição alvo do chute;
- *pass_receiver*: Id do robô que deve receber o passe.

Métodos

- *make_move_action*

Entrada: vetor indicando um destino.

Saída: ação do tipo MOVE.

- *make_kick_action*

Entrada: vetor indicando um alvo.

Saída: ação do tipo KICK.

- *make_pass_action*

Entrada: Id de robô, que irá receber o passe.

Saída: ação do tipo PASS.

- *gen_move_action*

Entrada: Id de um robô, estado, tabela de decisão.

Saída: ação do tipo MOVE aleatória, que não colide com a posição de nenhum outro robô, nem com a posição destino desses de acordo com a tabela de decisão.

- *gen_kick_action*

Entrada: Id de um robô, estado, tabela de decisão.

Saída: ação do tipo KICK no ponto de maior abertura do gol do inimigo que continuar aberto após as movimentações da tabela de decisão. É assumido que um chute é possível e isso foi verificado previamente.

- *gen_pass_action*

Entrada: Id de um robô, estado, tabela de decisão.

Saída: ação do tipo PASS aleatória para algum robô (do mesmo time) que possa receber o passe, o critério para receber o passe envolve que o receptor chegue em seu destino antes que o passador consiga chegar na bola e chutá-la até o destino do receptor, além disso após o chute o receptor deve conseguir ser o dono da bola.

- *gen_primary_action*

Entrada: Id de um robô, estado, tabela de decisão, flag que indica se é de chute.

Saída: ação do tipo KICK ou PASS, simplesmente abstrai *gen_pass_action* ou *gen_kick_action* para evitar verbosidade.

- *apply_to_state*

Entrada: ação, Id do robô, referência para um estado.

Efeito: aplica a ação a um estado. É assumido que a ação pode ser aplicada. Ações de chute levam a bola para o gol.

Estrutura *Decision*

Representa uma decisão total de um time, isto é todos os robôs irão possuir ações, mesmo que sejam do tipo NONE. Similarmente à uma ação individual, uma decisão não está atrelada a um time, e por isso é comum alguns métodos precisarem de uma decisão e o time a qual será aplicada.

Atributos

- *action*: array de ações de tamanho igual ao número de robôs que um time possui.

Métodos

- *apply_to_state*

Entrada: decisão, time e referência a um estado.

Efeito: todas as ações da decisão são aplicadas ao estado, assim como em uma ação é assumido que é possível aplicar a decisão.

- *gen_decision*

Entrada: flag que indica se é chute, estado, jogador, tabela de decisão e robô que irá se movimentar (opcional)

Saída: decisão gerada aleatoriamente usando as restrições de geração de ações. Caso um robô para se movimentar seja especificado esse será o único a receber uma movimentação aleatória os outros seguirão com as ações da tabela de decisão.

- *to_proto_command*

Entrada: decisão, time, referência para um mensagem *CommandMessage*, tabela de Ids.

Efeito: converte uma decisão para a estrutura de mensagem gerada pelo *Protobuf*.

Enumeração *DecisionSource*

Representa de qual tipo de ramo uma decisão foi gerada.

Alternativas

- *NO_SOURCE*: não há fonte, usado como padrão para decisões vazias.
- *SUGGESTION*: decisão gerada a partir de uma sugestão (posição chave).
- *TABLE*: decisão gerada a partir de uma tabela de decisão.
- *FULL_RANDOM*: decisão gerada a partir de uma movimentação de todos os robôs.
- *SINGLE_RANDOM*: decisão gerada a partir da movimentação de um único robô.

Estrutura *DecisionTable*

Atributos

- *kick_robot*: Id do robô com ação de chute, é opcional, na ausência indica que não há robô com ação de chute.
- *kick*: ação de chute.
- *pass_robot*: Id do robô com ação de chute, é opcional, na ausência indica que não há robô com ação de chute.
- *pass*: ação de passe.
- *move*: array de ações de movimentação com tamanho igual ao número de robôs que um time possui.

Estrutura *Optimization*

Uma instância representa uma unidade de otimização. A princípio seria necessário apenas uma função otimizadora mas com existe persistência de informações (como tabela de decisão) então foi criada essa estrutura para persistir tais dados de maneira explícita.

Atributos

- *table*: tabela de decisão.
- *robot_to_move*: usado para fazer o rodízio do robô que irá ser movimentado no modo SINGLE_RANDOM.
- *table_initialized*: flag para marcar se a tabela já foi inicializada, usado apenas para algumas otimizações.

Métodos

- *decide*

Entrada: instância de otimização, estado, time, sugestões (opcional)

Saída: decisão valorada, número de ramificação (opcional)

Enumeração *Player*

Indica um time.

Alternativas

- *MIN*: time cujo valor é minimizado, normalmente configurado como time inimigo.
- *MAX*: time cujo valor é maximizado.

Estrutura *Segment*

Representa um segmento de uma dimensão, usado para cálculo de *gaps* (vãos) no gol.

Atributos

- *u*: valor superior (up).
- *d*: valor inferior (down).

Estrutura *State*

Representa o estado do jogo. É importante ressaltar que os Ids usados dentro da ferramenta têm significado diferente do Id usado no jogo. Aqui os Ids são únicos para cada robô, mesmo entre times, isto é, um Id usado pelo time MIN não pode ser usado pelo time MAX. Mais notável é que os Ids variam necessariamente entre 0 e 11 inclusive, sendo a primeira porção correspondente ao time MIN e a segunda ao time MAX. Esse design permite usar Ids como índices de arrays e determinar o time a partir de um Id. Para tanto é necessário uma tabela de Ids para mapear para os Ids usados na partida.

Atributos

- *ball*: vetor posição da bola.
- *ball_v*: vetor velocidade da bola.

- *robots*: array de vetores posição de cada robô do jogo.
- *robots_v*: array de vetores velocidade de cada robô do jogo.

Métodos

- *uniform_rand_state*

Saída: estado uniformemente aleatório, usado apenas para testes.

- *can_kick_directly*

Entrada: estado, time.

Saída: booleano, se o time é capaz de realizar um chute direto.

- *robot_with_ball*

Entrada: estado.

Saída: Id do robô com a bola.

Saídas opcionais: tempo para o time MIN alcançar a bola, tempo para o time MAX alcançar a bola, Id do robô do time MIN mais próximo à bola, Id do robô do time MAX mais próximo à bola.

A posse de bola usa o tempo para chegar à bola, que leva em consideração a velocidade da bola.

- *total_gap_len_from_pos*

Entrada: estado, posição origem, time, robô a ser ignorado (opcional).

Saída: soma de todas as aberturas vistas a partir da origem no gol do time especificado.

- *max_gap_len_from_pos*

Entrada: estado, posição origem, time, robô a ser ignorado (opcional).

Saída: maior entre todas as aberturas vistas a partir da origem no gol do time especificado.

- *time_to_pos*

Entrada: vetor posição da origem, vetor velocidade da origem, vetor posição do destino, vetor velocidade do destino, velocidade máxima do ponto de origem (opcional, por padrão do robô).

Saída: menor tempo para o ponto chegar da origem no destino assumindo aceleração infinita do ponto e velocidade constante do destino.

- *discover_gaps_from_pos*

Entrada: estado, posição de origem, jogador, robô a ser ignorado (opcional).

Saída: array de segmentos e tamanho do array, contém todos os segmentos que representam os vãos (gaps) que podem ser observados no gol do time especificado.

- *evaluate_with_decision*

Entrada: time, estado, decisão, tabela de decisão.

Saída: valor, valores individuais por peso (opcional)

Essa é a função objetivo.

- *discover_possible_receivers*

Entrada: estado, tabela de decisão, jogador, Id do robô que faz o passe.

Saída: lista dos robôs to time dado que podem receber passes.

- *update_grom_proto*

Entrada: referência para um estado, mensagem UpdateMessage, tabela de Ids.

Efeito: atualiza o estado com os dados da mensagem se baseando no mapeamento da tabela de Ids.

Estrutura *SuggestionTable*

Tabela de sugestão, também referida por Posições Chaves. Representa um conjunto de posições chaves que é usado para gerar uma única decisão.

Atributos

- *name*: nome do conjunto, útil para rótulos como 'barreira' ou '2-2-1' ou 'ataque pela direita'.
- *spots_count*: contagem de posições.
- *spots*: array de vetores posição.
- *usage_count*: contagem de uso desta tabela, usado para estatísticas.

Métodos

- *add_spot*

Entrada: tabela de sugestão.

Saída: novo tamanho da tabela, ou -1 em caso de erro.

Efeito: adiciona uma posição na tabela.

- *del_spot*

Entrada: tabela de sugestão, posição a remover.

Saída: novo tamanho da tabela, ou -1 em caso de erro.

- *gen_decision*

Entrada: flag indicando se decisão é de chute, tabela de sugestão, estado, tabela de decisão, time.

Saída: decisão construída a partir da tabela de sugestão, o comportamento há menos posições que robôs é preencher com moves aleatórios e quando há mais posições que robôs é usar as posições mais próximas há cada robô sem que haja repetição.

Estrutura *Suggestions*

Representa o conjunto de todas as tabelas de sugestões.

Atributos

- *tables*: tabelas de sugestão.
- *tables_count*: contagem de tabelas.
- *last_use*: posição da última tabela usada para gerar uma decisão.

Métodos

- *add_suggestion*

Entrada: conjunto de sugestões.

Saída: novo tamanho do conjunto, ou -1 em caso de erro.

Efeito: adiciona uma posição no conjunto com uma tabela vazia.

- *del_suggestion*

Entrada: conjunto de sugestões, posição a remover.

Saída: novo tamanho do conjunto, ou -1 em caso de erro.

- *save_suggestions*

Entrada: conjunto de sugestões, caminho de arquivo.

Efeito: salva no arquivo o conjunto de sugestões em um formato textual.

- *load_suggestions*

Entrada: conjunto de sugestões, caminho de arquivo.

Efeito: carrega do arquivo o conjunto de sugestões.

Estrutura *ValuedDecision*

Essa estrutura existe para anexar um valor à uma decisão, a intenção é que esse seja um valor calculado pela função objetivo. Uma alternativa seria usar tuplas ou retornar o valor em um argumento de saída (ponteiro ou referência) mas a estrutura é simples e possui uma semântica explícita e facilita evoluções, como por exemplo a própria adição do atributo *values*.

Atributos

- *value*: valor associado à decisão.
- *values*: array de valores que explicita quanto parcela da função objetivo contribui com o valor total, a soma dos valores será igual ao atributo *value*.
- *decision*: decisão associada ao valor.

Estrutura *Vector*

Vetor em duas dimensões usado para representar posições e velocidades no programa. Não deve ser confundido com um *array*, que é uma sequência de tamanho genérico (porém fixo) sem semântica associada.

Atributos

- *x*: componente no eixo *x*.
- *y*: componente no eixo *y*.

Métodos

Alguns operadores foram definidos para essa estrutura de modo que suas operações refletem as operações matemáticas normalmente associadas a vetores.

- *norm2*

Entrada: vetor.

Saída: norma do vetor ao quadrado, usado para simplificar algumas operações, é ligeiramente menos custoso que a norma.

- *norm*

Entrada: vetor.

Saída: norma do vetor.

- *unit*

Entrada: vetor.

Saída: vetor com mesma direção porém norma unitária.

- *uniform_rand_vector*

Entrada: valor r_x , valor r_y .

Saída: vetor aleatório distribuído uniformemente no região: $-r_x < x < r_x$, $-r_y < y < r_y$.

- *normal_rand_vector*

Entrada: vetor origem, valor sigma.

Saída: vetor normalmente distribuído na origem e sigma entrados.

- *rand_vector_bounded*

Entrada: vetor origem, valor do raio, valores r_x e r_y .

Saída: vetor uniformemente distribuído em um círculo centrado na origem com o raio entrado que necessariamente está contido na região $-r_x < x < r_x$, $-r_y < y < r_y$, a segunda condição é sempre atendida nem que seja necessário quebrar a primeira.

- *line_segment_cross_circle*

Entrada: vetores p_1 , p_2 , vetor origem e valor do raio.

Saída: verdadeiro se o segmento (p_1, p_2) intercepta o círculo centrado na origem com o raio entrado.

- *dist*

Entrada: vetores v_1 e v_2 .

Saída: distância entre os vetores, sinônimo de $norm(v_1 - v_2)$.