

Very Small Size Soccer (VSSS) RobotBulls - Time e simulador

Alexandre B. Lugli¹, Davi Rosim³, Edmundo Henrique de P. Silva⁴, Egídio R. Neto², João Henrique S. Delfino⁵, João Roberto T. dos Santos⁶, Rodrigo O. Salles⁷, Wanderson E. Saldanha⁸

Abstract— O documento desenvolvido pela equipe de robótica do Instituto Nacional de Telecomunicações, a RobotBulls, na categoria IEEE Very Small Size Soccer, traz informações sobre as alterações feitas, a fim de evoluir e melhorar o desempenho dos robôs que participarão da competição. Neste contém novos métodos que aprimoram efetivamente o processamento das informações e tomadas as decisões. A ideia do trabalho é repartir as informações e percepção adquirida, almejando sempre ajudar as equipes que ingressam na categoria ou desejam melhorar sua performance.

Palavras-chave: Inteligência Artificial, Robótica, Software.

I. INTRODUÇÃO

Os integrantes da categoria IEEE Very Small Size Soccer (VSSS)[1], vêm por meio deste estudo, incentivar pesquisas nas áreas de robótica e inteligência artificial, já que estas possuem um alto nível de complexidade. Isso ocorre devido a muitos conceitos envolvendo tanto *hardware*, *software* e *firmware*. Desde o ano passado, o modelo de competição simulado foi adotado graças aos conceitos de redes de computadores em cima do que já estava sendo usado. Este projeto foi desenvolvido pela equipe de robótica e inteligência artificial do Instituto Nacional de Telecomunicações - INATEL, a RobotBulls, com a logo da equipe mostrada na Figura 1. Nessa categoria existem, atualmente, cinco alunos atuando diretamente na pesquisa e três professores, cujo intuito é guiar os trabalhos da equipe.

Desde 2015, a categoria VSSS da equipe RobotBulls, participa das competições oficiais do LARC (Competição Latino-Americana de Robótica - *Latin American Robotics Competition*) que ocorreu em Uberlândia/MG. Desde então, as pesquisas foram se intensificando e os tornaram reconhecidos e obtendo o título de vice-campeão, em 2016, e depois sempre se mantendo no pódio desde 2016 até 2019. Devido

¹A.B.Lugli, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
Telefone: (35) 99969-9571; e-mail: baratella@inatele.br)

³D.Rosim, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99106-9046; davi.rosim@ges.inatel.br)

⁴E.H.de P.Silva, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99741-8810; edmundo.henrique@geb.inatel.br)

²E.R.Neto, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99949-0681; egidio.neto@inatele.br)

⁵J.H.S.Delfino, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99990-1230; joao.h@gec.inatel.br)

⁶J.R.T.dos Santos, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99749-0015; joao.roberto@gea.inatel.br)

⁷R.O.Salles, INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99202-7336; rodrigo.salles@gea.inatel.br)

⁸W.E.Saldanha INATEL, Santa Rita do Sapucaí, CEP 37540-000 Brasil
(Telefone: (35) 99808-6430; wsaldanha@inatele.br)

à pandemia de Covid-19, em 2020, a competição ocorreu de uma forma nunca antes vista, de maneira remota e simulada, através do FIRASim[2]. Este se trata de um simulador em que ocorre a disputa das partidas entre as equipes.

Os trabalhos de pesquisa da equipe são melhor detalhados na seguinte disposição. O segundo capítulo, é atribuído ao *software* dos robôs, bem como o conceito de máquina de estados, utilizado nas táticas e na navegação, bem como explica a nova pesquisa sobre *Behavior Tree*[3]. No capítulo III, é apresentado o software FIRASim, que é utilizado para a simulação, testes e também é o *software* a ser utilizado na competição deste ano. No capítulo IV, são vistos os testes e resultados que a equipe obteve ao longo do desenvolvimento do projeto. No capítulo V, conclui-se a pesquisa feita pela equipe.



Fig. 1. Logo da equipe RobotBulls

II. SOFTWARE DOS ROBÔS

O software dos robôs é desenvolvido de modo a obter o melhor desempenho ofensivo e defensivo. Dessa forma, fatores relacionados às estratégias e comportamentos dos robôs, dada as circunstâncias do jogo, são de extrema importância.

A. Estratégia

Toda a estratégia é dada a partir da maneira analítica do *software* e do jogo como um todo (movimentações dos robôs aliados e adversários e da bola), bem como o placar é considerado. Assim, o time é organizado em uma determinada formação e em um estilo mais agressivo ou defensivo, baseando-se no jogo. E, para impedir que uma equipe não seja pega de surpresa durante a partida, há, implicitamente, o tratamento dos modelos de jogo de cada time.

B. Máquina de estados

A máquina de estados[4] é utilizada para colocar em prática a estratégia, dado uma situação de jogo. E, por sua vez, após o *client* receber e interpretar os dados, ela atribui a cada robô uma função individual ou coletiva. É importante salientar que a bola está sempre sendo identificada e recebe uma prioridade maior, em relação aos jogadores, para que a tomada de decisão seja precisa.

C. Behavior Tree

Em paralelo, implementa-se o método *Behavior Tree* (árvore de comportamento). Ela separa o comportamento de seus Agentes (objetos de jogo), deixando a estrutura mais organizada, mais fácil de se testar novos comportamentos e mais simples, caso a inteligência de algum agente tenha que ser alterada. A *Behavior Tree* é uma estrutura de dados formada por Nós de Comportamento. Desses, existem 3 tipos de Nós, como mostrado na Figura 2:

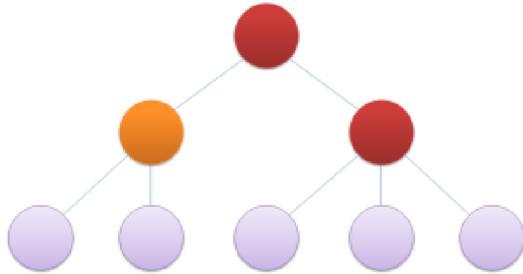


Fig. 2. *Behaviour Tree* - Árvore de Comportamento

- *Nó Folha ou Tarefa*: Esse nó sempre está presente nas pontas da árvore, e ele representa um comportamento do Agente. A tarefa é formada pela combinação de duas variantes: sua condição de execução e sua ação propriamente dita, exemplificado na Figura 3. Um exemplo de tarefa pode ser a seguinte: se a bola estiver indo para a esquerda e o adversário estiver na frente (condição), correr para a esquerda, na direção da bola e de modo a desviar do outro jogador (ação).

Task/Leaf Node

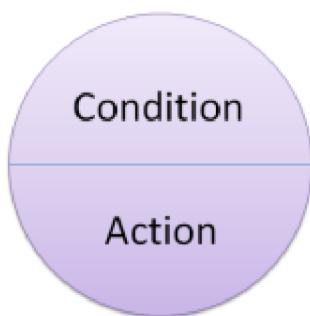


Fig. 3. *Nó folha - Tarefa*

- *Nó de Sequência*: Esse tipo de nó permite que cada nó filho seja verificado e executado um após o outro, evidenciado na Figura 4. Ele inicia sua tarefa verificando o seu primeiro nó filho (o nó filho pode ser de qualquer tipo). Caso o primeiro nó tenha sucesso quando executado, o nó de sequência executará o filho seguinte. Caso qualquer um dos nós filhos tenha fracassado ou sua condição não tenha sido alcançada, o nó de sequência retorna ao nível acima com a informação de fracasso. O nó de sequência terá sucesso apenas quando todos os filhos obtiverem sucesso na sua execução e condição.

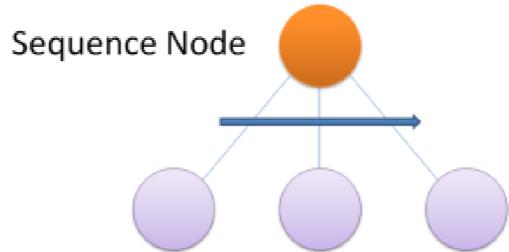


Fig. 4. *Nó de sequência*

- *Nó Seletor*: esse nó faz com que apenas uma ação filha seja efetuada iniciando similarmente ao de sequência, mas caso um nó filho tenha falhado, ele passa para o próximo até encontrar o primeiro nó filho que tenha sucesso. Caso o nó filho tenha sucesso em sua execução, o nó seletor retorna para o nível acima com o estado de sucesso e não executa os filhos restantes. O nó seletor só fracassa quando nenhum de seus filhos foram capazes de obter sucesso. Seu funcionamento está presente na Figura 5.

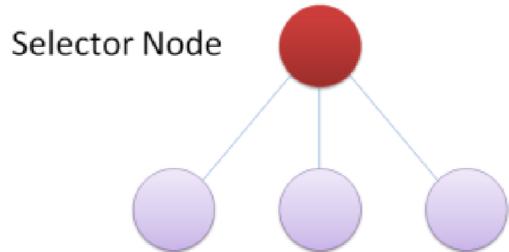


Fig. 5. *Nó seletor*

Com esse novo método para tomada de decisão, o time ganha uma enorme vantagem no processamento do código e resposta dos robôs, visto que é possível prever novas situações de jogo. Ainda assim, o comportamento dos robôs é mais acertivo, já que estes podem tomar as decisões mais cabíveis para cada situação.

D. Navegação

Para a navegação dos robôs em campo, a equipe utiliza o *UVF (Unit Vector Field)*[5], mostrado na Figura 6, ele faz com que o time apresente um resultado esperado durante a partida, igual ao apresentado durante a etapa de testes.

A fim de assegurar que a trajetória dos robôs seja mantida com mais precisão durante os jogos, faz-se uso de um tipo de controle de malha fechada utilizando conceitos de PID (Proporcional Integral Derivativo), como descrito nos estudos de Minorsky[6], e adaptado através do *software*. O controle pode ser configurado e ajustado através do código da inteligência, que adota a angulação do robô que está em campo como erro.

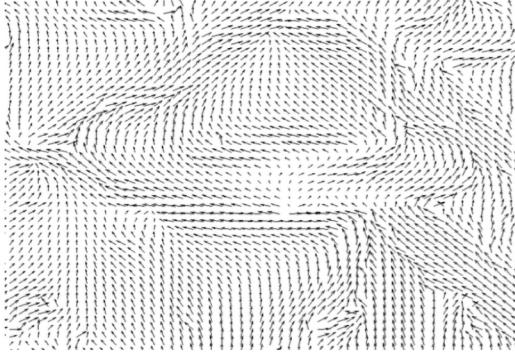


Fig. 6. Sistema *UVF* plotado na tela

III. SIMULADOR

O Simulador utilizado para o desenvolvimento de novas estratégias e que estará presente na categoria 5v5 é o *FIRASim*. Este *software* foi desenvolvido pela *ParsianLab*, o qual possibilita realizar testes em um ambiente virtual, simulando uma partida de VSSS com todos os requisitos necessários. Este ambiente permite que seja feito um desenvolvimento a nível de *software*, sem que haja o *hardware* dos robôs. Além disso, possibilitou a ocorrência de campeonatos durante a pandemia, mesmo que de maneira remota.

A. Comunicação

A comunicação[7] se assemelha à arquitetura cliente/servidor. Esta é uma das partes primordiais, visto que o aproveitamento dos serviços e dados fornecidos pelo simulador será efetivo, se a interação/comunicação estiver estabelecida. Dessa forma, utiliza-se o *Protocol Buffers*[8], sendo este um protocolo desenvolvido pelo Google, o qual possibilita realizar a serialização dos dados estruturados e obter um desempenho excelente.

Nesse sentido, o escopo da mensagem é definido por meio de arquivos “*.proto*”, os quais possuem todos os campos de informações que serão fornecidas pelo Simulador ou *Referee*[9]. As estruturas de dados a serem serializadas são descritas, de modo a identificar cada campo com seu respectivo nome, bem como seu tipo primitivo (*bool*, *int32*, *float*, *double* e *string*).

Uma vez que o escopo da mensagem foi definido, os arquivos “*.proto*” são compilados, gerando, automaticamente, as classes referentes às informações. Vale ressaltar, que estas possuem todos os métodos acessores e modificadores, os quais permitem acessar e modificar os dados. Feito isso, o simulador envia os pacotes na rede via *Multicast*, ou seja,

apenas para um grupo seletivo. Dessa forma, o cliente deve ser configurado a receber os pacotes *Multicast* referentes ao *FIRASim* para que possa ser feito o processamento das informações fornecidas por ele. A Figura 7 apresenta o esquema de comunicação do simulador, árbitro e o time de VSSS.

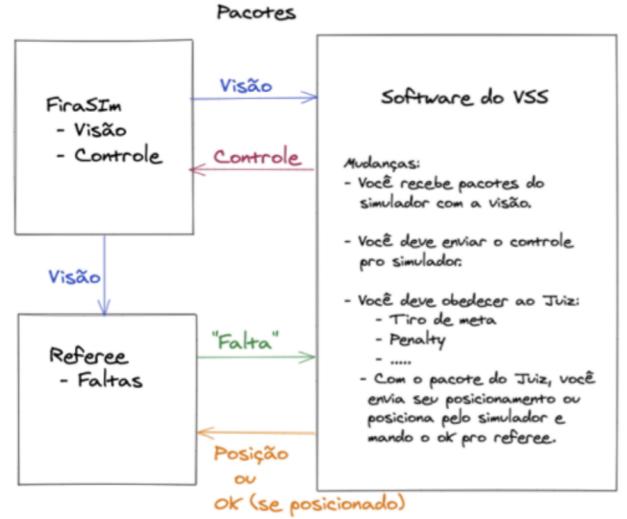


Fig. 7. Esquema de comunicação no simulador

B. Implementação entre os softwares

A implementação do *software* é realizada através da instalação das bibliotecas necessárias e o desenvolvimento do código referente ao time. A Figura 8 mostra a interface do simulador, seus parâmetros, bem como as informações sendo recebidas e impressas no terminal.

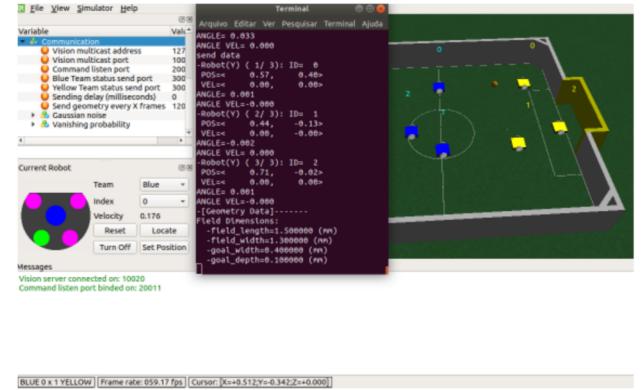


Fig. 8. Simulador testando os robôs

As subseções em sequência trazem cada tópico exibido individualmente.

1) Recursos necessários: Para o funcionamento correto do simulador, é necessário a instalação de algumas dependências. Entre os módulos necessários, vale ressaltar:

- ODE (Open Dynamics Engine):* Esta biblioteca[10] Open Source possibilita a simulação de corpos rígidos

e, assim, simular colisões reais, além de estabelecer o atrito.

- *OpenGL*: Esta *API (Application Programming Interface)*[11] permite desenhar os elementos do campo (robôs, linhas, marcações, etc) na interface do simulador.
- *Protobuf*: Esse pacote[8] permite que a comunicação ocorra corretamente, definindo as informações a serem trocadas, além de possibilitar a serialização desses dados.
- *QT*: Tal *Framework* possibilita realizar o desenvolvimento da interface, entre outros recursos.

2) *Client*: O *client*, desenvolvido por cada equipe, é responsável por receber as informações enviadas pelo Simulador, além de respeitar as situações de jogo enviadas pelo *Referee*. Entre algumas informações enviadas pelo Simulador, pode-se citar:

- Campo: Dimensões do campo, áreas, etc.
- Jogo: Tempo de jogo e o número de gols de cada time;
- Bola: Velocidade (V_x , V_y e V_z) e posição (x , y e z);
- Robôs: Posição (x , y e z) e orientação dos jogadores (radianos);

A partir das informações obtidas, cada time é responsável por implementar a sua própria estratégia. Dessa forma, é possível obter ganhos relacionados a jogadas ensaiadas, navegação, entre outros. De certa maneira, o desenvolvimento da inteligência de cada time torna-se exponencial, uma vez que não é necessária uma estrutura de *hardware* para realizar tais processos de teste.

C. Referee

O *software* do árbitro é um cliente do simulador, o qual é responsável por receber as informações de “visão”, processá-las e verificar se alguma infração foi cometida em jogo. Dessa forma, sua principal função é definir as situações de jogo e alertar aos times sobre elas. Assim que os times são informados, estes são responsáveis por analisar e definir as ações e posicionamentos dos robôs. Entre essas condições, podem ser:

- *KICKOFF*: Saída de bola, a qual é informado a qual time será fornecida.
- *FREE BALL*: Bola livre, a qual ocorre sempre que a bola fica parada em um determinado quadrante do campo por 5 segundos. A disputa ocorre no quadrante em que foi verificada tal situação.
- *GOAL KICK*: Tiro de meta é sempre marcado quando ocorre uma falta de ataque, por exemplo. A falta de ataque ocorre quando dois robôs atacantes invadem a área adversária com bola estando dentro desta.
- *PENALTY KICK*: Pênalti, o qual ocorre sempre que tiver uma falta defensiva ou a bola permanece paralisada na área defensiva. Dessa forma, é informada a situação de jogo, além de qual time deve cobrá-lo.
- *GAME ON*: Tal condição ocorre e é enviada para os times quando a situação de jogo é permitida.
- *STOP*: Paralisação do jogo.

- *HALT*: Paralisação da partida, mantendo os robôs em suas respectivas posições e velocidades.

Na Figura 9 pode-se observar o ciclo de trabalho do *software* do árbitro:

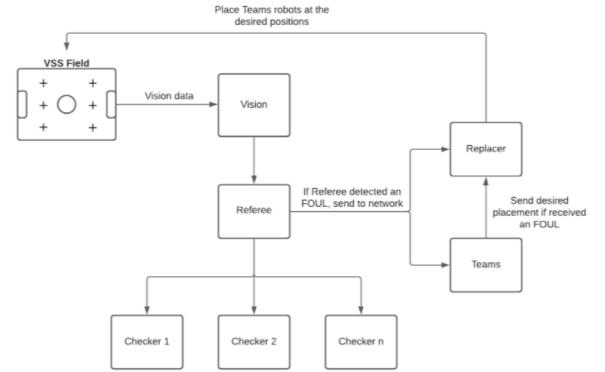


Fig. 9. Fluxo de trabalho do *Referee*

IV. TESTES E RESULTADOS

A fim de desenvolver o código e analisá-lo de maneira mais eficiente, a equipe testa diversas opções e configurações dentro do próprio *software FIRASim*, desafiando o próprio time e a própria estratégia. Por vezes, há confrontos amistosos em período de testes, é possível validar uma estratégia nova ou até fazer a otimização de uma antiga, além disso há o campeonato brasileiro da categoria, que também põe à prova todos os recursos implementados. Ademais, explora-se o ensejo de encontrar bugs no simulador enquanto disputam, de modo a também aprimorá-lo. Todos estes testes acontecem utilizando o juiz automático, cuja função é tomar todas as decisões pela interpretação do *software Referee*, sem nenhuma intervenção humana, exceto nos casos que seja necessário corrigir algum problema ou sinalização indevida do árbitro.

De modo a assegurar um maior aproveitamento dos robôs dentro de campo, utilizam-se os testes com o juiz manual, sendo este integrante de uma das equipes que se dispõe em ser o árbitro, logo ele sinaliza todos os eventos que julgar importantes ao decorrer da partida. Dessa forma, também é há a possibilidade de jogar com livremente sem ter de parar o jogo para marcar faltas, sempre verificando os robôs e seus comportamentos em situações inusitadas.

V. CONCLUSÃO

Através do simulador *FIRASim*, a equipe pôde concluir que de fato é possível testar diversas estratégias e jogadas. Com seu uso, os integrantes testaram o código em qualquer lugar que estivessem, evitando a necessidade de estar presente no laboratório da RobotBulls, uma vez que a pandemia do Covid-19 fez com que o acesso dos pesquisadores no campus do Inatel fosse reduzido.

Em razão da implementação e da experimentação dos novos métodos, além da máquina de estados, houve a nova

ideia, a *behavior tree*, o time de robôs vem se sobressaindo aos testes dentro de campo. Por fim, o objetivo da equipe, se tornar um dos pilares da categoria *VSSS* no país, está sendo cada vez mais possível, melhorando o time a cada partida e a cada teste feito, reduzindo as perdas e otimizando a navegação dos robôs.

REFERENCES

- [1] IEEE. *Very Small Size Soccer (VSSS) Rules*. URL: https://www.cbrobotica.org/wp-content/uploads/2022/05/vssRules_5x5.pdf (visited on 06/20/2022).
- [2] VSSSLeague. *FIRASim*. URL: <https://github.com/VSSSLeague/FIRASim> (visited on 06/29/2022).
- [3] Bazan A. *Inteligência Artificial e Behaviour Trees*. URL: <https://beattheplaguegame.wordpress.com/2012/05/25/inteligencia-artificial-e-behaviour-trees/> (visited on 05/20/2022).
- [4] M. Colledanchise. *Behavior Trees in Robotics and AI*. 1st. 2015.
- [5] Khan Academy. *Vector Fields*. URL: <https://www.khanacademy.org/math/multivariable-calculus/thinking-about-multivariable-function/ways-to-represent-multivariable-functions/a/vector-fields> (visited on 06/14/2022).
- [6] N. Minorsky. “Directional Stability Automatically of Steered Bodies”. In: *Naval Engineers Journal* 42 (1922), pp. 280–309. ISSN: 0028-1425. DOI: <https://doi.org/10.1111/j.1559-3584.1922.tb04958.x>. URL: <https://onlinelibrary.wiley.com/doi/10.1111/j.1559-3584.1922.tb04958.x>.
- [7] Robocin. *Introdução ao FIRASim*. URL: <https://drive.google.com/file/d/14MZ13T2ztmO7dDnpCYRy4KFDlzB1HYhx/view> (visited on 07/01/2020).
- [8] Google. *Protocol Buffers*. URL: <https://developers.google.com/protocol-buffers/> (visited on 06/09/2022).
- [9] Maracatronics Robotics. *VSSReferee*. URL: <https://github.com/MaracatronicsRobotics/VSSReferee> (visited on 07/13/2020).
- [10] Russ Smith. *Open Dynamics Engine*. URL: <http://www.ode.org/> (visited on 07/04/2020).
- [11] OpenGL. *OpenGL*. URL: <https://www.opengl.org/> (visited on 07/04/2021).
- [12] Neto Z. *VSSReferee*. URL: <https://github.com/VSSSLeague/VSSReferee#concept-of-play> (visited on 08/22/2021).