

UNIVERSIDADE FEDERAL DE SANTA MARIA
CENTRO DE TECNOLOGIA
PROGRAMA DE PÓS-GRADUAÇÃO EM CIÊNCIA DA COMPUTAÇÃO

Felipe Pedrozo Maia

**DESENVOLVIMENTO DE UMA LINGUAGEM BASEADA EM
ONTOLOGIAS PARA GERAÇÃO DE AÇÕES DE FUTEBOL DE ROBÔS**

Santa Maria, RS
2017

Felipe Pedrozo Maia

**DESENVOLVIMENTO DE UMA LINGUAGEM BASEADA EM ONTOLOGIAS PARA
GERAÇÃO DE AÇÕES DE FUTEBOL DE ROBÔS**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

ORIENTADOR: Prof. Dr. Giovani Rubert Librelotto

Santa Maria, RS
2017

Ficha catalográfica elaborada através do Programa de Geração Automática da Biblioteca Central da UFSM, com os dados fornecidos pelo(a) autor(a).

Maia, Felipe
DESENVOLVIMENTO DE UMA LINGUAGEM BASEADA EM
ONTOLOGIAS PARA GERAÇÃO DE AÇÕES DE FUTEBOL DE ROBÔS /
Felipe Maia.- 2017.
72 f.; 30 cm

Orientador: Giovani Rubert Librelotto
Dissertação (mestrado) - Universidade Federal de Santa
Maria, Centro de Tecnologia, Programa de Pós-Graduação em
Ciência da Computação, RS, 2017

1. Parser 2. Robôs 3. Ontologia 4. Comportamento I.
Rubert Librelotto, Giovani II. Título.

©2017

Todos os direitos autorais reservados a Felipe Pedrozo Maia. A reprodução de partes ou do todo deste trabalho só poderá ser feita mediante a citação da fonte.

Fone (0xx) 55 99989 9379; End. Eletr.: fmaia@inf.ufsm.br

Felipe Pedrozo Maia

**DESENVOLVIMENTO DE UMA LINGUAGEM BASEADA EM ONTOLOGIAS PARA
GERAÇÃO DE AÇÕES DE FUTEBOL DE ROBÔS**

Dissertação de Mestrado apresentada ao Programa de Pós-Graduação em Ciência da Computação, Área de Concentração em Ciência da Computação, da Universidade Federal de Santa Maria (UFSM, RS), como requisito parcial para obtenção do grau de **Mestre em Ciência da Computação**.

Aprovado em 29 de agosto de 2017:

Giovani Rubert Librelotto, Dr. (UFSM)
(Presidente/Orientador)

Jonas Bulegon Gassen, Dr. (UFSM)

Rafael Teodósio Pereira, Dr. (AMF)

Santa Maria, RS
2017

DEDICATÓRIA

Dedico esse trabalho aos meus pais que não mediram esforços para me ajudar e apoiar.

AGRADECIMENTOS

Agradeço ao meu orientador, Giovani, pela confiança, apoio e principalmente paciência nos momentos difíceis.

A Universidade Federal de Santa Maria, seu corpo docente, direção e administração que oportunizaram a janela que hoje vislumbro um horizonte superior.

Agradeço aos meus pais, que sempre estiveram presentes dando o apoio necessário para continuar essa caminhada até seu término.

Agradeço aos meus amigos pelos momentos de descontração, companheirismo e incentivo que sempre deram.

Agradeço a todos que direta ou indiretamente estavam torcendo para que as coisas fluíssem da melhor forma.

*A persistência é o menor caminho do
êxito.*

(Charles Chaplin)

RESUMO

DESENVOLVIMENTO DE UMA LINGUAGEM BASEADA EM ONTOLOGIAS PARA GERAÇÃO DE AÇÕES DE FUTEBOL DE ROBÔS

AUTOR: Felipe Pedrozo Maia

ORIENTADOR: Giovani Rubert Librelotto

O avanço da robótica para participação de robôs no cotidiano humano é assunto de diversas pesquisas em áreas da saúde, entretenimento, educação, entre outras. Em específico, a pesquisa e desenvolvimento de robôs para disputa de partidas de futebol tem ganhado força com a implantação da Liga RoboCup. A criação de comportamentos do robô em uma partida de futebol pode alcançar níveis de conhecimento interessantes para outras áreas onde robótica pode ser aplicada, assim sendo, ela se torna um aspecto chave na evolução da robótica em qualquer área de aplicação. No desenvolvimento de comportamentos, linguagens de programação são utilizadas para criar algoritmos nos quais o robô irá se basear para realizar ações. Nota-se que esta não é uma tarefa trivial de ser desenvolvida, e dado que diversas linguagens de programação possam ser utilizadas, a tarefa se torna ainda mais complexa. O presente trabalho aborda a criação de uma ontologia para o domínio de futebol de robôs como forma de descrever de forma semântica e lógica um conjunto de comportamentos do robô, dessa forma, ela busca facilitar a programação. Além disso, tal ontologia serve como forma de compartilhar conhecimento sobre as regras criadas e sobre o domínio utilizado nesse trabalho. Outro ponto abordado pelo trabalho é o desenvolvimento de uma linguagem para programação de comportamentos de robôs inseridos em uma partida de futebol. Essa linguagem, chamada Maia, é baseada na linguagem RS, isso se dá ao fato dela ser simples e ter o formato "Condição \Rightarrow Ação" o que facilita o raciocínio. Assim, ela visa facilitar o desenvolvimento de comportamentos para robôs, visto que esta é uma tarefa complexa de ser desenvolvida por um leigo no assunto. Dessa forma, o tempo necessário de aprendizado de uma linguagem pode ser um empecilho para o desenvolvedor. Desse modo, ontologia e Maia se complementam formando um papel importante na criação de comportamentos, visto que é possível mapear tal cenário e implementá-lo levando em consideração a dinamicidade do jogo.

Palavras-chave: Parser. Robôs. Ontologia. Comportamento.

ABSTRACT

CREATION OF A LANGUAGE FROM ONTOLOGIES TO GENERATE SOME FOOTBALL ACTIONS IN ROBOTS

AUTHOR: Felipe Pedrozo Maia
ADVISOR: Giovani Rubert Librelotto

The advances in robotics for everyday life are subject of research in several fields such as health, entertainment and education. Specifically, the research and development of robots for the dispute of soccer matches has gained strength with the RoboCup league. The development of robot behaviors for soccer matches can lead to applications in several areas of knowledge, becoming a key aspect in the evolution of robotics in any application. For the development of behaviors, programming languages are used to create algorithms that determine the robot's actions. It is noticeable that this is not a trivial task, specially when considered that several programming languages may be used. The current work approaches the creation of an ontology for the robot soccer domain, as a way of describing in a semantic and logic form the set of behaviors of a robot. Furthermore, the ontology facilitates the process of sharing the rules created under this domain. Another topic of this work is the development of a language for programming robot behaviors in the context of soccer matches. The programming language, called Maia, is based in the RS language due the fact that it is simple and has a format "Condition -> Action" that facilitates reasoning. In this context, the goal of the programming language is to ease the development of behavior for robots, once it is a complex task for non-experts, where the time necessary to learn a programming language can be limiting factor. Hence, the ontology and the Maia programming language complement themselves becoming an important factor in the creation of behaviors, once that it is possible to map each scenario and implement it taking into consideration the dynamism of the game.

Keywords: Parser. Robot. Ontology. Behavior.

LISTA DE FIGURAS

| | |
|--|----|
| Figura 2.1 – Suposto funcionamento do pato de Vaucason | 16 |
| Figura 2.2 – Máquina de fiar de Crompton | 17 |
| Figura 2.3 – Boneca de Henri Maillardet | 17 |
| Figura 2.4 – Robôs humanoides | 19 |
| Figura 2.5 – Jogo na RoboCup | 20 |
| Figura 3.1 – Arquitetura Taura Bots | 28 |
| Figura 3.2 – Programação do comportamento | 29 |
| Figura 4.1 – Áreas conceituais da ontologia | 35 |
| Figura 4.2 – Estruturação das classes da ontologia | 35 |
| Figura 4.3 – Relações conceituais | 36 |
| Figura 4.4 – Relações conceituais | 36 |
| Figura 4.5 – Propriedades de dados | 37 |
| Figura 4.6 – Instância de bola | 37 |
| Figura 4.7 – Propriedades de dados | 38 |
| Figura 5.1 – Abordagem do trabalho | 39 |
| Figura 6.1 – Posição inicial da jogada | 52 |
| Figura 6.2 – Robô goleiro chega até a bola | 53 |
| Figura 6.3 – Chute do robô goleiro | 53 |
| Figura 6.4 – Arquivo a utilizando linguagem Maia | 56 |
| Figura 6.5 – Arquivo de saída do programa | 57 |
| Figura 6.6 – Principais passos do processo | 57 |
| Figura 6.7 – Simulador | 58 |
| Figura 6.8 – Simulador executando comportamento | 58 |

LISTA DE TABELAS

| | |
|--|----|
| Tabela 2.1 – Trabalhos relacionados..... | 27 |
| Tabela 5.1 – Palavras Reservadas..... | 46 |

LISTA DE ABREVIATURAS E SIGLAS

| | |
|---------------------------|--|
| <i>ASEME</i> | <i>Agent Systems Engineering Methodology</i> |
| <i>DARwIn</i> – <i>OP</i> | <i>Dynamic Anthropomorphic Robot with Intelligence - Open Platform</i> |
| <i>CASE</i> | <i>Computer-Aided Software Engineering</i> |
| <i>CBR</i> | Competição Brasileira de Robótica |
| <i>FIFA</i> | <i>Fédération Internationale de Football Association</i> |
| <i>IA</i> | Inteligência Artificial |
| <i>JIRA</i> | <i>Japan Industrial Robot Association</i> |
| <i>JSON</i> | <i>JavaScript Object Notation</i> |
| <i>KSE</i> | <i>Kouretes Statechart Editor</i> |
| <i>RIA</i> | <i>Robotic Industries Association</i> |
| <i>RS</i> | Reativa Síncrona |
| <i>TAURA</i> | Tecnologia em Automação e Robótica Aplicada |
| <i>UDP</i> | <i>User Datagram Protocol</i> |
| <i>UFMS</i> | Universidade Federal de Santa Maria |

SUMÁRIO

| | | |
|----------|---|-----------|
| 1 | INTRODUÇÃO | 11 |
| 2 | REFERENCIAL BIBLIOGRÁFICO | 14 |
| 2.1 | ROBÓTICA | 14 |
| 2.1.1 | História da Robótica | 15 |
| 2.1.2 | Robôs Humanoides | 18 |
| 2.1.3 | RoboCup | 20 |
| 2.2 | ONTOLOGIA | 21 |
| 2.3 | LINGUAGEM RS | 23 |
| 2.4 | TRABALHOS RELACIONADOS | 25 |
| 2.5 | SUMÁRIO DO CAPÍTULO | 27 |
| 3 | TAURA BOTS | 28 |
| 3.1 | MÓDULO DE VISÃO | 29 |
| 3.2 | MÓDULO COMPORTAMENTO | 30 |
| 3.3 | MÓDULO AÇÃO | 30 |
| 3.4 | INTERFACE VISÃO-COMPORTAMENTO | 31 |
| 3.5 | INTERFACE COMPORTAMENTO - AÇÃO | 32 |
| 3.6 | SUMÁRIO DO CAPÍTULO | 32 |
| 4 | DESCRIÇÃO DO DOMÍNIO | 33 |
| 4.1 | ESPECIFICAÇÃO DA ONTOLOGIA | 33 |
| 4.1.1 | Classes | 33 |
| 4.1.2 | Propriedades e relações | 36 |
| 4.1.3 | Instâncias e Regras | 37 |
| 4.2 | SUMÁRIO DO CAPÍTULO | 38 |
| 5 | METODOLOGIA | 39 |
| 5.1 | ARQUITETURA DO PROCESSO | 39 |
| 5.2 | REQUISITOS | 40 |
| 5.2.1 | Requisitos de Entrada | 40 |
| 5.2.2 | Requisitos de Saída | 42 |
| 5.2.3 | Estrutura da Linguagem | 43 |
| 5.3 | DEFINIÇÃO DA GRAMÁTICA E IMPLEMENTAÇÃO | 45 |
| 5.3.1 | Gramática | 45 |
| 5.3.2 | Parser | 47 |
| 5.3.2.1 | Saída do Parser | 47 |
| 5.4 | SUMÁRIO DO CAPÍTULO | 51 |
| 6 | ESTUDO DE CASO E TUTORIAL DA FERRAMENTA | 52 |
| 6.1 | ESTUDO DE CASO | 52 |
| 6.2 | TUTORIAL DA FERRAMENTA | 56 |
| 6.3 | SUMÁRIO DO CAPÍTULO | 59 |
| 7 | CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS | 60 |
| | REFERÊNCIAS BIBLIOGRÁFICAS | 61 |
| | ANEXO A – GRAMÁTICA DA LINGUAGEM MAIA | 63 |

1 INTRODUÇÃO

Há muito tempo o ser humano usa ferramentas e utensílios com a finalidade de facilitar a execução de suas atividades diárias. No decorrer dos séculos, muitas invenções forneceram o necessário conhecimento tecnológico para a gradativa troca do homem pela máquina. Contudo, apenas quando ocorre de forma sistemática, a aplicação da ciência na indústria é que há efetivamente alteração do cenário (ROMANO; DUTRA, 2002).

É visível o avanço tecnológico nos dias atuais, além do grande fascínio dos humanos pelo campo da robótica (JÚNIOR et al., 2011). Essas máquinas buscam a autonomia e as características fomentadas a elas, seja uma funcionalidade bem específica, como montagem de uma determinada peça de carros em uma fábrica, ou algo mais complexo, como uma aproximação das características humanas (físicas e mentais). Em especial, robôs humanoides, os quais possuem aparência semelhante à humana, possuindo duas pernas, dois braços, um tronco e uma cabeça conectada a um pescoço (Sousa et al., 2011).

Esse fascínio e interesse pelo avanço da robótica levou, em 2002, a criação da RoboCup, competição mundial de robótica sediada cada ano em um país diferente e promove competições de futebol de robôs humanoides. Algumas das equipes participantes realizam apenas jogadas e movimentos pouco elaborados. Essa é a razão na qual realiza-se uma pesquisa no âmbito colaborativo com troca de conhecimentos entre as equipes do campeonato, utilizando os avanços adquiridos.

Outro item interessante é a perspectiva dos idealizadores da competição, onde na RoboCup de 2050, robôs completamente autônomos devem derrotar a atual equipe humana campeã do mundo do mesmo ano, utilizando regras da FIFA (KITANO et al., 1998). Com isso, surgiram vários desafios que envolvem diversas pesquisas sobre estratégias. Tais como, evitar obstáculos, métodos de interceptação da bola, formação de equipe, algoritmos de localização no campo, estudos biológicos sobre comportamento, ambiente de simulação para facilitar o estudo de estratégias.

Na Universidade Federal de Santa Maria (UFSM) não é diferente o interesse em robôs humanoides de competição, tanto que há um grupo de pesquisa, Taura Bots, voltado principalmente para programação de robôs humanoides. Grupo esse que teve sua primeira participação na RoboCup em 2015. Para isso, utilizou-se a plataforma open-source DARwIn-OP (*Dynamic Anthropomorphic Robot with Intelligence - Open Platform*), sendo assim é possível alterar *hardware* e *software* do robô.

Desde então, foram feitas pesquisas para melhorar o desempenho do robô. A ideia do grupo vai muito além de fazer com que o robô apenas apresente comportamentos futebolísticos semelhantes ao de jogadores reais, mas sim aprender técnicas, adaptá-las e aplicá-las nas mais diversas áreas, como saúde e educação, para simplificar o trabalho dos humanos.

Com o propósito de facilitar essas melhorias no desenvolvimento do robô, a equipe Taura Bots foi dividida em três subequipes: caminhada, visão e comportamento. A primeira subequipe é responsável pelo equilíbrio e caminhada. O segunda subequipe é encarregada por processar informações capturadas no ambiente e afim de influenciar no comportamento. Por último está a subequipe encarregada pelo comportamento, a mente do robô.

Para facilitar a criação de comportamentos e como forma de minimizar testes diretamente no robô foi desenvolvido um simulador 2D para o robô. O simulador é implementado com a mesma linguagem de programação utilizada no desenvolvimento do robô, Python. Desta forma é possível usar o mesmo código tanto no robô quanto no simulador, o que facilita os testes envolvendo programação. Outro ponto interessante do simulador, para os desenvolvedores de comportamentos, é sua abstração das etapas de caminhada e visão.

Neste sentido, faz-se necessário, quando deseja-se modelar um comportamento, que o usuário ou programador necessite saber o conhecimento técnico do funcionamento do simulador e do robô real, bem como a linguagem de programação utilizada. A necessidade de aprendizado de uma linguagem, assim como a necessidade de aprendizado do funcionamento dos sistemas de simulação e do robô podem ser considerados inviáveis para novos desenvolvedores, pois o emprego de trabalho de desenvolvimento se torna maior, assim como o tempo reservado para tais atividades.

Outro ponto a ser considerado é o tipo de linguagem usada, isso seria outro empecilho visto que é possível utilizar as mais diversas linguagens de programação nos mais diversos tipos de robôs humanoides, assim para criar um comportamento se faz necessário o aprendizado de cada uma delas. Isso é algo muito trabalhoso, tornando-se muito difícil para pessoas com pouco conhecimento em programação.

Assim, muitas vezes o usuário que está desenvolvendo um comportamento deseja que o robô tenha o conhecimento X, no entanto o usuário não tem conhecimento dos comandos necessários para execução de determinada ação. Ou seja, não sabe como desenvolver comportamentos para que o robô caminhe, por exemplo. Outro item são os movimentos, é necessário que o usuário saiba quais são os possíveis movimentos que o robô pode efetuar, visto que isso é um ponto importante para determinar certa jogada.

Dessa forma, o presente trabalho visa, através do uso de uma linguagem, chamada Maia, facilitar essa programação. A escolha da criação de Maia se deu pelo fato dela ser baseada na linguagem RS, o que segundo Toscani (1993) ela é mais intuitiva para programação de robôs que interagem com ambiente externo. Sendo assim, devido Maia ser bem delimitada para que possam aprender apenas os comandos e funcionalidades voltados especificamente para o futebol de robôs.

Com esse objetivo é utilizado um parser para validação de tal linguagem além de um tradutor para Python. Como mencionado anteriormente, simulador e robô utilizam Python como forma de descrever o comportamento. Assim, caso alguém queira criar comporta-

mentos será necessário aprender apenas uma linguagem.

Outro item abordado pelo trabalho é a criação de uma ontologia, onde a mesma visa facilitar a programação utilizando a linguagem Maia. Isso se dá uma vez que a ontologia descreve o cenário do futebol de robôs humanoides bem como faz a utilização de algumas palavras reservadas da linguagem, além de conter representações gráficas o que facilita o aprendizado do domínio. Mais uma vantagem da utilização da ontologia é facilidade da forma de compartilhar conhecimento sobre as regras criadas e sobre o domínio utilizado nesse trabalho.

Apesar da ontologia e Maia formarem um importante papel para programação, vale ressaltar que elas são independentes. Sendo assim, uma vez que é acrescentado um novo item na ontologia o mesmo deve ser feito manualmente para a linguagem Maia para que ambas estejam coerentes.

A partir disso, esse trabalho apresenta no Capítulo 2 conceitos fundamentais para a compreensão do estudo realizado. Já o Capítulo 3 aborda a estrutura e funcionamento da equipe TauraBots. No Capítulo 4 é abordada a ontologia utilizada para fazer a descrição de tal linguagem. A descrição da linguagem, seus requisitos, o funcionamento do parser e implementação são expostos no Capítulo 5. Um estudo de caso, é apresentado no Capítulo 6. Por último, o Capítulo 7, que aborda as considerações finais do trabalho, bem como trabalhos futuros.

2 REFERENCIAL BIBLIOGRÁFICO

Neste capítulo são abordados os principais assuntos que facilitarão a compreensão dessa dissertação. São eles: robótica, ontologias e linguagem RS. Além disso, serão apresentados trabalhos que em algum aspecto estão relacionados com o propósito desse trabalho.

2.1 ROBÓTICA

A robótica nada mais é que um campo envolvendo tecnologia e educação, para isso engloba computadores e robôs. Ela preocupa-se com as partes mecânicas, cujo as mesmas são comandadas por circuitos integrados, assim, tem-se um sistema mecânico motorizado, que pode ser tanto controlado de forma automática quanto manual (OTTONI, 2010).

Assim, essas máquinas podem até mesmo serem consideradas vivas devido estarem sempre em busca de autonomia de acordo com as características designadas a elas. Essas características podem ser algo bem específico, como colocar um parafuso em uma roda, quanto algo muito complexo, como aproximar as características humanas (OTTONI, 2010).

“A robótica, bem como a automação industrial são tecnologias muito relacionadas. A robótica é uma ciência da engenharia aplicada que é tida como uma combinação da tecnologia de máquinas operatrizes e ciência da computação” (TSAI, 1999).

Segundo Groover (1989), robôs são uma tecnologia manipulável projetada para manusear matérias, ferramentas ou dispositivos através de movimentos pré-definidos pelo programador com o intuito de realizar diversas tarefas. Para a realização dessas atividades, o robô precisa de dois aspectos: movimentação e programação. Estes aspectos estão melhores especificados a seguir:

- Movimentação do robô - Também chamado de ação ou comportamento, a movimentação é produzida a partir de uma sequência de movimentos básicos, chamados graus de liberdade (DOF – degrees of freedom) (ARMADA et al., 1998). A conexão dos membros do robô é feita através de juntas, que permitem o movimento do mesmo de acordo com o grau de liberdade.
- Programação do robô – É responsável por captar e processar várias características do mundo, de acordo com a funcionalidade do robô, e definir suas ações a partir

disso. Como coletores de dados para processamento, podem utilizar diversos tipos de sensores como: sensores de temperatura e sensores luminosidade (GROOVER, 1989).

A robótica, uma vez que engloba vários campos de conhecimentos científicos, ou seja, transdisciplinar, tem por objetivo desenvolver e incluir técnicas e algoritmos para criação do robô. É essa mistura de áreas de conhecimento que tem proporcionado o avanço que está ocorrendo hoje (HALFPAP et al., 2005).

Algumas definições de robótica possuem origem mais abstratas e vêem os robôs como sistemas que interagem com o mundo real. Outras, mais técnicas, os consideram como verdadeiras máquinas animadas ou autônomas. Na verdade, diversas definições têm surgido como é o caso desta baseada na ideia francesa de robô: “Robô é um dispositivo automático adaptável a um meio complexo, substituindo ou prolongando uma ou várias funções do homem e capaz de agir sobre seu meio” (MARTINS, 1993). Este conceito pode ser comparado com a moderna interpretação do pesquisador canadense, Marshall McLuhan, ao afirmar que: “todo produto da tecnologia, de alguma forma, faz estender nossos sentidos e nervos” (MARTINS, 1993).

Ainda de acordo com Martins (1993), alguns conceitos parecem não satisfazer os pesquisadores da área da robótica, argumentando que são por demais simplificados e incompletos e por não se referirem às características fundamentais dos robôs atuais, ou seja:

- a) sensibilidade;
- b) capacidade de excluir por inspeção;
- c) capacidade de identificar peças;
- d) capacidade de posicionar peças.

No entanto, existem controvérsias por parte da Japan Industrial Robot Association (JIRA), que defende que máquinas operadas pelo homem podem ser consideradas robôs, independentemente da complexidade delas. O que se pode adiantar, é que a robótica entre nós se mostra incipiente e o caminho a ser percorrido é longo e árduo.

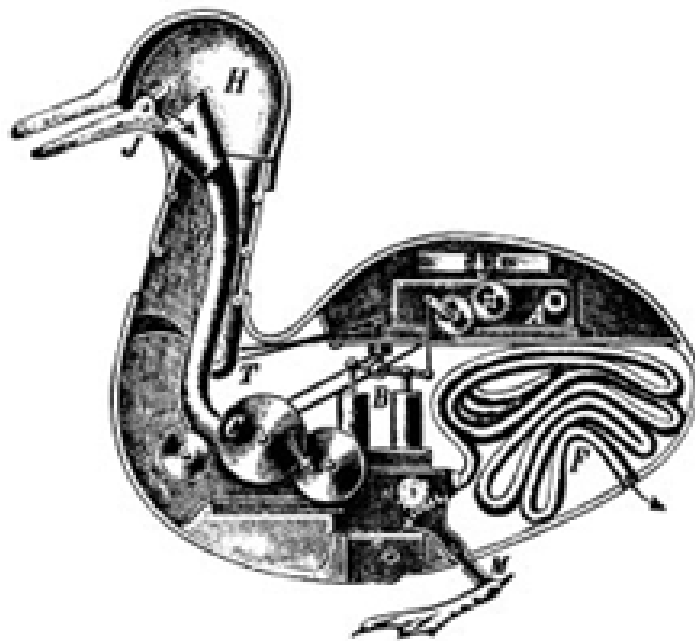
2.1.1 História da Robótica

O interesse pela construção de robôs já existe há algum tempo, tanto que entre 1495 e 1497 Leonardo da Vinci idealizou o primeiro robô articulado antropomórfico, sendo esse um cavaleiro dotado de armadura, com partes em madeira, couro e bronze, e com

acionamento através de cabos. Ele era capaz de fazer movimentos similares aos humanos, como sentar-se, mover os braços, pescoço e maxilar (ROSHEIM, 1997).

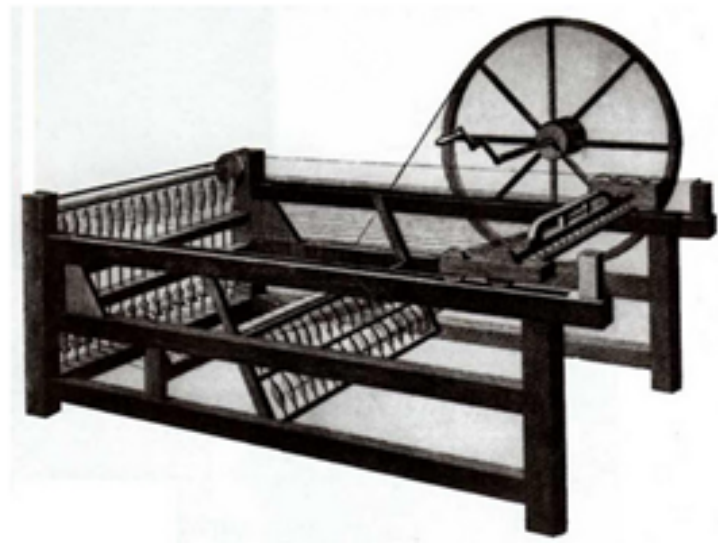
Pouco tempo depois começaram a surgir lendas e mitos sobre seres inanimados. Como o caso do flautista mecânico, do célebre “pato de Vaucanson” (Figura 2.1), que visava simular um pato de verdade. Sendo assim, ele conseguia fazer alguns movimentos. No entanto, devido ao conhecimento que tinha-se na época essas máquinas ficavam um pouco restritas quanto as suas funcionalidades, desempenhando um número pequeno de tarefas.

Figura 2.1 – Suposto funcionamento do pato de Vaucason



Já evolução industrial foi um momento que houve muitas invenções mecânicas, principalmente direcionadas a produção têxtil. Em Pazos (2002) há como exemplo, a máquina de fiar de Crompton de 1779, como pode ser vista na Figura 2.2. Ela possibilitou a manufatura em larga escala de fios com alta qualidade.

Figura 2.2 – Máquina de fiar de Crompton



Em 1805 Henri Millardet criou uma boneca que escrevia e desenhava com precisão (Figura 2.3) (PAZOS, 2002). Como pode ser observado, já começam a surgir máquinas com características físicas e motoras semelhantes a de seres humanos.

Figura 2.3 – Boneca de Henri Maillardet



A palavra "robô" foi inicialmente usada pelo cientista e escritor Isaac Asimov, em 1942, numa história chamada "Runaround". No entanto, desde 1939 ele já escrevia histó-

rias sobre robôs, embutidas de salvaguardas. Essas salvaguardas foram formalizadas em três leis para a robótica. Passado algum tempo uma quarta lei foi criada e elas atualmente são utilizadas como código de ética dos profissionais da área (ALVES, 1988). As leis são as seguintes:

- 1ª lei: Um robô não pode ferir um ser humano ou, por omissão, permitir que o ser humano sofra algum mal.
- 2ª lei: Um robô tem de obedecer às ordens recebidas dos seres humanos, a menos que contradigam a Primeira Lei.
- 3ª lei: Um robô tem de proteger sua própria existência, desde que essa proteção não entre em conflito com a Primeira e a Segunda Leis.
- 4ª lei: Um robô não pode causar mal à humanidade nem permitir que ela própria o faça (Asimov, 1997).

Essas leis na época em que foram criadas tinham cunho totalmente voltado a ficção, visto que em tal momento era improvável prever o avanço desse área. No entanto, no início do século XX, o tão almejado desejo da construção de robôs, de fato ganha vida, uma vez que há certa carência pelo aumento da produção sem perda na qualidade dos produtos.

2.1.2 Robôs Humanoides

Robôs humanoides nada mais são que robôs autônomos, os quais realizam determinadas atividades em um ambiente sem influência humana, fazendo adaptações de acordo com o ambiente no qual o mesmo está inserido. Além disso, possuem características físicas semelhantes aos seres humanos, como tronco com cabeça, dois braços e duas pernas. Um exemplo pode ser visto na Figura 2.4.

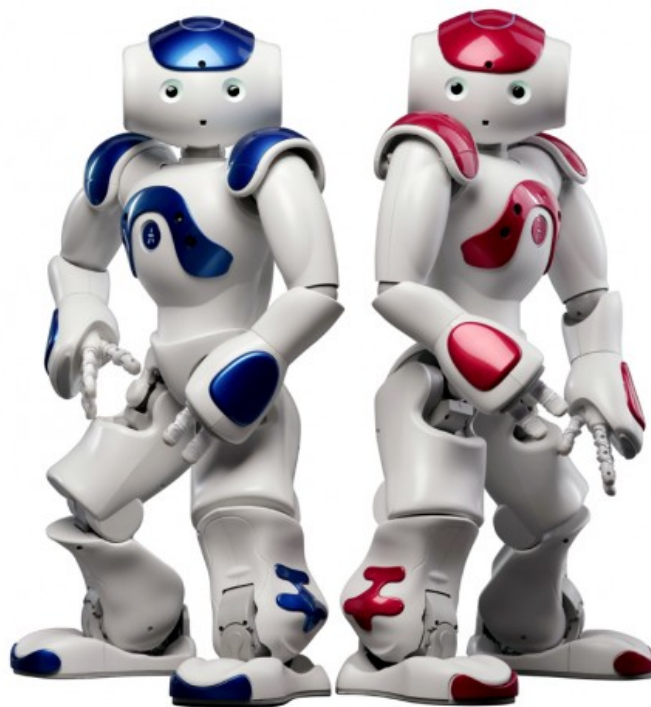


Figura 2.4 – Robôs humanoides

Para Akhtaruzzanab e Shafie (2010) é notável o avanço de pesquisas nessa área, pois sabe-se da importância que essas máquinas podem ter para auxiliar humanos quando elas estiverem em nível elevado de desenvolvimento. Esse nível elevado diz respeito a movimentos precisos, como: fazer com que eles tenham o mesmo movimento da caminhada dos humanos, e inteligência para que possam tomar decisões em qualquer situação e ambiente que sejam adequadas para tal.

Já Wolf et al. (2009) também acredita que robôs humanoides atraem atenção pela possibilidade de terem comportamentos semelhantes aos dos humanos, podendo tomar decisões, demonstrar emoções e ter relações afetivas. Outro ponto, que instiga pesquisas, é a possível adequação deles em vários ambientes.

O motivo pelo qual há bastante interesse na pesquisa sobre robôs humanoides, se dá pela forma de manipulação dos braços, de andar sobre dois pés e as mais diversas habilidades para comunicar-se, dessa forma há um benefício quando empregado de imediato em ambientes do seres humanos. Segundo Schwarz (2012) , para que os robôs possuam vantagens e sejam confiáveis, eles devem ter capacidade de acessar lugares de difícil acesso, e, entender e executar expressões e comunicação baseada em gestos.

2.1.3 RoboCup

A RoboCup é um evento internacional sediado anualmente em diferentes países e teve seu evento inaugural em 2002 em Fukuoka, no Japão. A RoboCup busca incentivar pesquisas nas áreas de robótica e Inteligência Artificial (IA). Para isso, são utilizados problemas do mundo real com o intuito de suscitar o interesse do grande público (ROBOCUP, 2017a).

Como proposta bastante ambiciosa da competição, ela almeja que em torno do ano 2050 o time campeão do torneio possa vencer o time campeão da copa do mundo FIFA do mesmo ano. Ou seja, demonstra-se interesse no avanço rápido das pesquisas nas áreas de robótica e IA, tendo um grande avanço durante as próximas décadas. No entanto, é possível estabelecer alguns paralelos, como é o caso do vôo do primeiro avião, onde passados 50 anos houve a missão Apollo que enviou e trouxe o homem à Lua em segurança. Outro exemplo dessa mesma linha, é a invenção do computador digital, o Deep Blue, que venceu o campeão mundial de xadrez, Garry Kasparov (ROBOCUP, 2017a).

Ainda para ROBOCUP (2017b), a RoboCup incentiva pesquisas de robôs móveis autônomos, sendo assim, sistemas de IA são um dos principais focos. Além disso outras áreas também são investigadas, como: eletrônica, computação, sistemas embarcados, visão computacional, sistemas de comunicação, mapeamento, sistemas em tempo real, aprendizado de máquinas, dentre outras. A Figura 2.5 foi tirada em um jogo da RoboCup, sendo assim representa o ambiente real dos robôs.

Figura 2.5 – Jogo na RoboCup



A RoboCup possui diversas ligas de competição, sendo a mais conhecida a de futebol de robôs (RoboCup Soccer). Nesse caso, o objetivo é desenvolver a cooperação de sistemas de robôs multi-agentes para atingir determinado fim. Existem várias categorias de disputa entre robôs. As categorias variam de acordo com as características dos robôs,

como tamanho e forma de locomoção (rodas ou bípedes)(ROBOCUP, 2017b).

Há também a liga de robôs para resgate (RoboCup Rescue), que visa cenários de desastres naturais (enchentes, terremotos, maremotos, furacão) e têm como finalidade desenvolver sistemas de mapeamento e localização das vítimas (ROBOCUP, 2017b). Esse é um propósito interessante pois facilita o trabalho dos profissionais que são responsáveis por tais funções, além de minimizar os riscos que os mesmos possam ter em determinados terrenos.

Além das ligas citadas, existe a liga RoboCup@home, destinada ao desenvolvimento de tecnologias assistivas. A RoboCup@home busca avanços que tornem robôs autônomos capazes de auxiliar seres humanos, principalmente pessoas com algum tipo de necessidade especial. Nesse caso, algumas das funcionalidades atribuídas ao robô estão relacionadas as atividades cotidianas como pegar determinado objeto, auxiliar na preparação de refeições, dentre outras que minimizem ou facilitem o trabalho humano.

Por último, existe a liga RoboCup Junior, desenvolvida especialmente para a inclusão de crianças no meio. Assim, desde cedo, crianças irão se aproximar do mundo da tecnologia através da robótica educacional, buscando novos talentos para a área. Das ligas apresentadas, a RoboCup Soccer é a qual o trabalho busca fazer contribuições científicas.

2.2 ONTOLOGIA

Ontologia nada mais é que um conceito que existe e é utilizado pela Filosofia há muito tempo, sendo definido como: estudo das coisas que existem. O termo original é a palavra aristotélica categoria, utilizada com o propósito de classificar e caracterizar qualquer entidade. Aristóteles ainda introduz o termo “differentia” para propriedades que distinguem diferentes espécies do mesmo gênero (SOWA, 2001).

Uma ontologia pode ser definida como uma especificação formal e explícita de uma conceituação compartilhada (GRUBER, 1995), onde:

- conceituação se refere ao modelo abstrato do mundo real;
- explícita significa que os conceitos e seus requisitos são definidos explicitamente;
- formal indica que a ontologia é processável por máquina, permite raciocínio automático e possui semântica lógica formal;
- compartilhada significa que uma ontologia captura o conhecimento apresentado não apenas por um único indivíduo, mas por um grupo.

De outra forma, pode-se dizer que ontologia é um modelo de dados que representa o conhecimento de um domínio através dos conceitos e relacionamentos entre eles.

Outra definição propõe o compartilhamento e reuso de ontologias. Esta definição fala de uma proposta de uso de ontologias para modelar problemas e domínios, onde elas fornecem uma biblioteca para fácil reutilização de classes e objetos para a modelagem. O objetivo desta proposta é o desenvolvimento de uma biblioteca de ontologias que poderia ser reutilizada e adaptada para diferentes classes de problemas e ambientes (GRUNINGER, 1996).

Atualmente, uma das definições mais citadas na área de computação, e que será adotada neste trabalho, é descrita por Borst (1997). Segundo o autor: “uma ontologia pode ser vista como uma especificação formal e explícita de uma conceitualização compartilhada”. Nessa definição: conceitualização refere-se ao modelo abstrato de algum fenômeno do mundo o qual identifica conceitos relevantes ao próprio fenômeno; formal significa o fato de a ontologia ser legível por computadores; especificação explícita diz respeito a conceitos, propriedades, relações, funções, restrições, axiomas definidos explicitamente; compartilhada quer dizer conhecimento consensual, ou seja, uma ontologia obtém o conhecimento apresentado não somente por um indivíduo único, mas sim por um grupo.

No que se refere a utilização de ontologias, pode-se afirmar que elas são aplicadas para proporcionar a comunicação entre diferentes pessoas e aplicações que fazem parte de um domínio do conhecimento em comum, mas que por vezes não compartilham de uma mesma conceitualização a respeito dos componentes deste domínio. Essa falta de entendimento compartilhado pode desencadear problemas na interoperabilidade e possibilidade de reuso e compartilhamento de conhecimento, o que é muito importante em função da grande variedade de métodos, paradigmas, linguagens e ferramentas existentes na área de computação. Logo, devido a natureza formal de uma ontologia, os conflitos conceituais e terminológicos são reduzidos (LOPES et al., 2007).

De acordo com o trabalho de Guimarães (2002), as principais vantagens na utilização de ontologias são:

- Ontologias fornecem um vocabulário para representação do conhecimento, o qual tem uma conceitualização bem definida, evitando assim interpretações ambíguas desse vocabulário.
- Ontologias permitem o compartilhamento de conhecimento, ou seja, caso exista uma ontologia que modele adequadamente certo domínio de conhecimento, essa pode ser compartilhada e usada por pessoas que desenvolvam aplicações dentro desse domínio. Por exemplo, considerando que existe uma ontologia para o domínio de livrarias, várias livrarias podem construir seus catálogos utilizando o vocabulário fornecido por essa ontologia sem a necessidade de refazer uma análise deste domínio.

- Fornece uma descrição exata do conhecimento. Diferentemente da linguagem natural em que as palavras podem ter semântica totalmente diferente conforme o seu contexto, a ontologia por ser escrita em linguagem formal, não deixa espaço para o gap semântico existente na linguagem natural. Por exemplo, quando uma pessoa fala para outra a palavra Globo, ela pode estar querendo falar a respeito de um corpo esférico, como também de um canal de televisão brasileiro. A interpretação da palavra pode ser atribuída a um conceito ou outro conforme o estado mental do indivíduo. Porém, se há uma conceituação comum entre essas duas pessoas, a possibilidade de mal entendido diminui consideravelmente. Por exemplo, se essas pessoas concordam em uma ontologia sobre o domínio de formas geométricas, possivelmente não haverá mal entendido.
- É possível fazer o mapeamento da linguagem da ontologia sem que com isso seja alterada a sua conceitualização, ou seja, uma mesma conceitualização pode ser expressa em várias línguas.
- Pode ser possível estender o uso de uma ontologia genérica de forma a que ela se adeque a um domínio específico. Por exemplo, se alguém precisa de uma ontologia sobre bicicletas para construir uma aplicação e só encontra uma ontologia sobre o domínio genérico de veículos, pode utilizar essa ontologia estendendo-a para o domínio específico da aplicação, que no caso são de bicicletas.

2.3 LINGUAGEM RS

Essa linguagem foi implementada por Simão S. Toscani (Toscani, 1993). Segundo Alnord et al. (2002) a linguagem RS (Reativa Síncrona) pode ser analisada como uma linguagem para programação de núcleos reativos. Sendo sistemas reativos, sistemas que tem integração com o meio externo. Além disso, mantém relacionamento dinâmico com ele.

Para Librelotto et al. (2008), RS, é orientada para especificação e implementação de sistemas que são regidos por dados de um ambiente externo. Ela é uma linguagem simples, com regras com o seguinte formato “Condição=>Ação”, dessa forma busca facilitar o raciocínio de programação, além de propiciar maior concentração, no que se refere a questões lógicas, no sistema em construção.

Ainda segundo Librelotto et al. (2008) , um programa RS, possui um conjunto de módulos, sendo cada módulo composto por um conjunto de caixas. Já cada uma dessas caixas, possuem um conjunto de regras de reação. Com módulos e caixas é possível

estruturar tal programa, no entanto, eles não necessariamente precisam ser usados, pois um conjunto de regras pode definir esse programa.

De acordo com Librelotto (2001) os sinais da linguagem RS têm a função de obter comunicação com o exterior para sincronização interna. Os sinais são identificados por nomes e contém valores ou não. Sendo assim, os sinais estão divididos em três grupos que são apresentados como:

- Sinais de entrada - são sinais provenientes do ambiente externo e somente esses sinais são capazes de desempenhar reações. A declaração desse tipo de sinal é representada por input.
- Sinais de saída - são sinais que tem por objetivo o ambiente externo, dessa forma, serve para indicar os resultados das reações. Como identificação do mesmo, é usado output.
- Sinais internos - utilizados na sincronização e comunicação interna de processos. Podendo ser subdivididos em sinais temporários e permanentes.

Como forma de exemplificar um programa RS, é demonstrado um programa de verificação de login. No exemplo a seguir são usados dois sinais de entrada, name(X) e password(Y), também dois sinais internos, gotName(X) e gotPassword(Y), e por último um sinal de saída, checkLogin(X, Y). Os sinais que são declarados em input tem acionamento somente pelo ambiente externo. Já os sinais *signal* e *output* são acionados de acordo com as regras.

```
module checklogin :
input  : name(X) , password(Y)
output: checkLogin(X,Y)
signal: gotName(X) , gotPassword(Y)
name(X) -> gotName(X)
password(Y) -> gotPassword(Y)
gotName(A) , gotPassword(B) -> checkLogin(A,B)
```

De acordo com Toscani e Monteiro (1994), nesse exemplo, na regra name(X) => gotName(X), o estímulo externo name(X) deve acionar o sinal interno gotName e instanciar o parâmetro X com o valor recebido. A segunda regra tem semelhança com a primeira, nessa, password(X) aciona o sinal interno gotPassword e instancia o parâmetro Y com o valor recebido. A última regra determina que quando os dois sinais internos, gotName(A) e gotPassword(B), forem ligados, o sinal checkLogin(A,B) deve ser emitido, sendo que A e B, representam os valores associados a X e Y, respectivamente.

A partir disso, do funcionamento de sistemas síncronos, pode-se afirmar que robôs tem um funcionamento muito similar. Assim sendo, os robôs recebem estímulos, input, da

visão e através de combinações de regras são capazes de reagir mandando informações para que tal robô interaja com o ambiente.

2.4 TRABALHOS RELACIONADOS

Os seguintes trabalhos relacionados são de pesquisas de algumas equipes que participam ou já participaram da RoboCup. O foco principal dos mesmos é apresentar as diferentes formas de criação ou manipulação de como os comportamentos dentro das mesmas.

Em Bonini (2015) propõe uma linguagem de domínio específico para facilitar a criação de estratégias de futebol de robôs. Também é feita uma tradução dessa linguagem para Python, que é utilizada na camada de comportamento do robô e simulador do grupo Taura Bots. No trabalho são apresentadas três funções básicas de ações que o robô pode efetuar.

No trabalho de Allgeuer e Behnke (2013) são apresentados dois frameworks para controle de comportamento: a Biblioteca de Controladores de Estado e o Framework de Controle de Comportamento. O primeiro serve para a implementação de comportamentos de agentes de complexidade de baixo a médio nível, especialmente comportamentos que tendem a exigir sequências de ações estruturadas. Demonstrou-se útil na implementação de diversas máquinas de estados finitos necessárias ao longo do código da plataforma NimbRo-OP. Uma limitação da estrutura, no entanto, é que não permite execução de um comportamento básico a qualquer momento, mesmo que este efeito possa ser teoricamente alcançado com o uso de multithreading.

Ainda para Allgeuer e Behnke (2013) este ponto, entre muitos outros, foram abordados pelo Framework de Controle de Comportamento, que utiliza uma árvore de inibições de comportamento para avaliar a cada instante que comportamentos devem ser ativados. Isso permite que vários aspectos de um agente sejam controlados simultaneamente por comportamentos independentes. O Framework de Controle de Comportamento foi destinado à implementação de comportamentos de agentes de média a alta complexidade. Assim, a Biblioteca de Controladores de Estado ainda pode ser usada dentro dos comportamentos individuais de uma arquitetura Framework de Controle de Comportamento para implementar sequências de ação com base em máquinas de estados finitos. Ambos os frameworks foram projetados com desempenho e eficiência em mente e formam uma base robusta na qual um sistema de controle de comportamento pode ser construído.

Um outro trabalho, de Topalidou-Kyniazopoulou, Spanoukadis e Lagoudakis (2013), apresenta uma ferramenta de Engenharia de Software Assistida por Computador (Computer-Aided Software Engineering - CASE), chamada Kouretes Statechart Editor (KSE), que permite que o desenvolvedor especifique o comportamento de um robô desejado como um

modelo de estado utilizando uma variedade de funcionalidades de robôs base (visão, localização, locomoção, habilidades de movimento). Um statechart é um modelo formal, independente de plataforma, amplamente utilizado em engenharia de software para projetar sistemas de software.

A KSE adota a abordagem orientada por modelo da Metodologia de Engenharia de Sistemas (Agent Systems Engineering Methodology - ASEME). Assim, a KSE orienta o desenvolvedor através de uma série de etapas de design dentro de um ambiente gráfico que leva à geração automática de código fonte. Assim, essa equipe usa a KSE para desenvolver o comportamento dos robôs humanoides para a competição (TOPALIDOU-KYNIASOPOULOU; SPANOUDAKIS; LAGOUDAKIS, 2013).

Bestmann(2014) faz a utilização do RatsLAM, cujo mesmo é um método utilizado na localização, que como vantagem, tem uma baixa necessidade de hardware. Esse algoritmo, tem por objetivo resolver problemas de localização e mapeamento. Para isso, usa como inspiração biológica os ratos, através de memorização da imagem e dos diferentes lugares. Dessa forma, ele opera de maneira similar a células do cérebro dos ratos, nas quais, representam um mapa de seu estado atual de atividade. No entanto, existem células específicas que são acionadas quando o rato encontra o local correspondente.

Nesse caso, baseadas em tais características, o autor utiliza o algoritmo fazendo integração com um software existente da RoboCup, assim os robôs têm a capacidade de se localizarem durante uma partida. Outro ponto do algoritmo, é que tem um bom funcionamento em ambientes parecidos com labirintos. No entanto, o campo de futebol há muitas semelhanças se observado de diferentes ângulos, o que pode gerar uma falsa correspondência de posição e conseqüentemente um erro na localização.

Uma técnica usada pelo autor para minimizar tal problema, é usando as próprias características conhecidas do ambiente para fazer essas distinções na localização, podendo ser distância para o gol, linhas, dentre outros. Porém, ainda assim, alguns problemas foram encontrados, mas há perspectivas de que essa localização possa ser melhorada com a criação de novos modelos desse algoritmo.

Já Wang (2015) faz a utilização de ontologias. Ele relata a importância na utilização de ferramentas que sejam capazes de captar, formalizar e verificar elementos comportamentais, bem como relações e interações entre eles, seja em forma qualitativa ou quantitativa. Para eles esses comportamentos são operações, ações ou eventos conduzidas por um único indivíduo ou pelo conjunto deles em determinados contextos.

Assim, ele objetiva modelar o comportamento, formando conceitos que facilitem a representação e raciocínio para uma equipe. Para isso, propôs dois modelos para representar comportamentos: visual e formal. OntoB, é uma modelagem baseada em ontologia, contendo classes, relações, instâncias e axiomas. Essa ontologia de comportamento possui quatro principais unidades: ator, operação, relação e contexto. Sendo que o contexto aborda relações de comportamento entre atores. Já as relações possuem três subcatego-

rias: temporais, inferenciais e baseada na partida.

A Tabela 2.1 mostra uma comparação entre os trabalhos apresentados. Demonstra resumidamente as diferentes aplicações entre as abordagens citadas. Onde pode-se notar que não existe um fator direto de comparação entre uma abordagem em relação a outra. Pode-se dizer também que cada abordagem teve a capacidade de resolver um problema em específico.

Ainda na Tabela 2.1, é notável que o presente trabalho traz duas das abordagens utilizadas nos outros trabalhos. Sendo elas a utilização de ontologias e uma proposta de linguagem. No entanto a linguagem proposta nesse trabalho apresenta um maior número de funcionalidades. Já no que se refere a ontologia, ela visa representar dentro do contexto de futebol de robôs as funcionalidades pela linguagem proposta.

Tabela 2.1 – Trabalhos relacionados

| | Proposta de linguagem | Máquina de Estados | Ontologia | Software Gráfico | Localização |
|--|-----------------------|--------------------|-----------|------------------|-------------|
| Bonini (2015) | X | | | | |
| Allgeuer e Behnke (2013) | | X | | | |
| Topalidou-Kyniazopoulou, Spanoukadis e Lagoudakis (2013) | | | | X | |
| Bestmann(2014) | | | | | X |
| Wang (2015) | | | X | | |
| Presente Trabalho | X | | X | | |

2.5 SUMÁRIO DO CAPÍTULO

O presente capítulo buscou relatar os principais conceitos necessários para uma melhor compreensão do trabalho, tais como ontologia, robótica e linguagem RS. Além disso, foram apresentados trabalhos relacionados a criação de comportamentos em robôs dentro das equipes participantes da RoboCup.

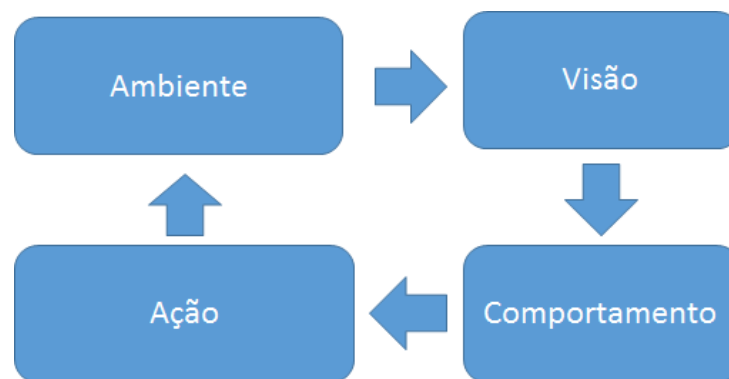
O próximo capítulo descreve o panorama global sobre a arquitetura utilizada na equipe Taura Bots, onde pode-se definir três grandes módulos, chamados de Visão que detecta objetos no mundo, Comportamento que define as ações a seres executadas e Ação que interage diretamente com os servos do robô.

3 TAURA BOTS

Este capítulo aborda, de uma maneira geral, a arquitetura usada pela pelo Grupo de Pesquisa TAURA (Tecnologia em Automação e Robótica Aplicada) da UFSM e sua equipe de competição em futebol de robôs humanoides chamada Taura Bots. A organização da equipe foi dividida em três grandes eixos: Visão, Comportamento e Ação (Montenegro, 2015).

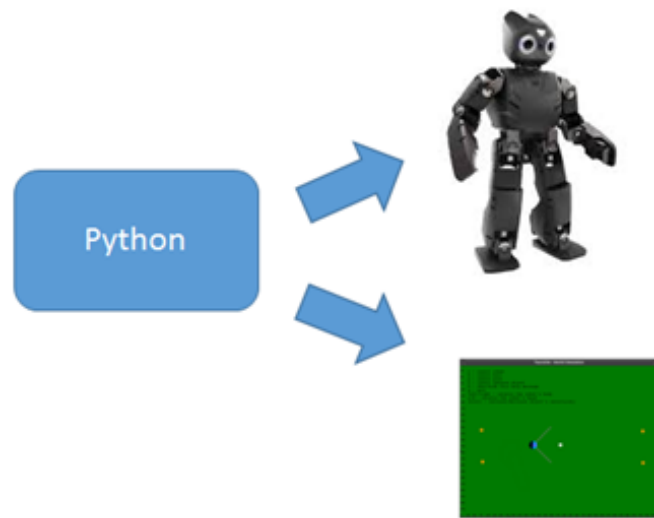
Onde o módulo de visão é responsável pela captação de dados do ambiente tratando-os e repassando para o próximo módulo. Já o Comportamento recebe estes, faz o tratamento e decide qual o que deve ser feito pelo robô através de ações. Sendo assim, o módulo de Ação executa tais comandos repassados para ele através do comportamento. Como pode ser visto na Figura 3.1.

Figura 3.1 – Arquitetura Taura Bots



Como linguagem de programação, foi pensado em optar por Python, visto que tem um alto grau de expressividade e boa parte dos membros da equipe tem algum conhecimento sobre a mesma. Porém, um código base open source na linguagem C++, presente no DARwIn-OP é disponibilizado pela Robotis. Assim sendo, algumas funcionalidades já estão desenvolvidas no mesmo, facilitando a implementação. Dessa forma, apenas o módulo de comportamento é programado na linguagem Python, como pode ser vista na Figura 3.2, onde tanto robô quanto simulador utilizam a mesma linguagem para criação de novas ações no robô.

Figura 3.2 – Programação do comportamento



A seguir são apresentados os módulos, propostos por Montenegro (2015), com maiores detalhes da equipe TAURA. Bem como acontece a comunicação entre essas camadas. Para isso, é necessária uma interface de comunicação de dados com os parâmetros essenciais para que haja robustez na comunicação das partes.

3.1 MÓDULO DE VISÃO

O módulo de Visão tem por objetivo a captação de dados do ambiente, sendo alguns deles: bola, robôs, linhas e intersecções do campo, além de postes. Outra função importante, é a auto localização que serve, por exemplo, para saber qual lado da quadra o robô está e qual o lado em que deve tentar fazer gols.

Vale ressaltar que, além do simulador, foi utilizada a plataforma open-source DARwIn-OP, sendo assim é possível alterar *hardware* e *software* além de possuir preços acessíveis e equipamentos computacionais avançados com potência, sensores sofisticados, alta capacidade de carga e capacidade de movimento dinâmico para permitir muitas atividades de pesquisa e educação.

Tal módulo segue a estrutura base original, linguagem C++ para implementá-lo. Assim, usa duas abordagens para detecção de objetos do ambiente, através do OpenCV¹ e outra fazendo a detecção de cores. Essas escolhas se dão pela conjuntura da equipe, onde há mudanças das características da bola de acordo com a competição, principalmente em

¹<https://opencv.org/>

relação a cor. Como é o caso da Competição Brasileira de Robótica (CBR) e da RoboCup uma vez que no mesmo ano as mesmas não possuíram iguais características da bola.

Ainda nesse módulo, a detecção dos objetos se dá através de coordenadas polares, sendo assim, possuem dois parâmetros que são: posicionamento angular do objeto em relação ao centro da câmera e distância em que esse foi detectado. A partir disso é possível obter a localização precisa na qual o mesmo se encontra.

3.2 MÓDULO COMPORTAMENTO

O módulo Comportamento possui o papel de definir jogadas mediante o contexto que está sendo captado em tal momento. Ou seja, é responsável pela tomada de decisões, criar estratégias definindo o que o robô deve fazer. No entanto, tal módulo não tem muitos comportamentos previamente definidos. Com isso, o trabalho abordará a possibilidade de criação de alguns comportamentos de acordo com as preferências do programador ou usuário. Assim sendo, o robô funciona como a mente do robô, processando informações e definindo o que o mesmo deve fazer.

3.3 MÓDULO AÇÃO

Esse módulo tem a função de efetuar mudanças nos servos dos robôs, isto é, faz os movimentos do robô. Há estudos sendo desenvolvidos pela equipe com o intuito de melhorar algoritmos de caminhada, dessa forma o robô terá uma maior agilidade(Montenegro, 2015).

Também é possível a execução de movimentos estáticos, dessa forma é feita uma sequência de movimentos de movimentos dos servos para atingir um objetivo específico. Algumas dessas ações podem ser levantar do chão, andar de lado, levantar o braço ou até mesmo fazer uma coreografia de dança.

No entanto, como algoritmo de caminhada, é usado o código base proposto pela Robotis, presente no DARWIN-OP. Porém, por vezes tal código deixa a desejar, um exemplo é quando o comportamento consegue repassar informações precisas para que o robô faça um giro em torno da bola para ficar de frente para o gol, o módulo de caminhada ainda não consegue executar tal ação. No entanto como forma de minimizar isto, são criados movimentos de forma estática.

3.4 INTERFACE VISÃO-COMPORTAMENTO

A Visão do robô tem a função de captar o ambiente de futebol de robôs e repassar essas informações, tais como dos objetos, para que sejam tomadas as decisões de o que pode ser feito a partir delas. É a visão que vai informar a localização exata de objetos como postes, robôs, dentre outras informações que o mesmo possui.

Para isso, é feita uma comunicação entre Visão e Comportamento que segue o protocolo de comunicação UDP¹ (User Datagram Protocol), e cada um dos pacotes envia um objeto no formato JSON² (JavaScript Object Notation) que é o formato de serialização de dados e formatação de mensagens.

Dessa forma, o Comportamento também passa a reconhecer tal formato e pode implementar suas estratégias em linguagem Python sem que existam problemas na comunicação. A seguir é apresentado um exemplo de entrada que o comportamento pode receber (Montenegro, 2015).

```
{
  "kind" : "ball",
  "position": [
    239.9031999919439,
    -0.6876952725551101
  ]
}
```

Especificando um pouco mais os detalhes desse exemplo, existe um atributo "kind" que representa cada objeto captado pela câmera, que no caso possui apenas a bola. Uma segunda informação que pode ser analisada é a posição desse objeto, cujo a mesma possui duas posições que são representações das coordenadas polares raio (r) e theta (θ).

O r é a distância que o objeto está. Já o θ é o ângulo em relação ao centro da câmera que está localizada na cabeça do robô. O valor de θ é representado como negativo para a direita e positivo para esquerda. Um outro ponto que vale ressaltar, é que nesse exemplo podem ter mais objetos no cenário, que no caso é um campo de futebol, no entanto só estão sendo repassada informações dos que estão no campo de Visão do robô no momento em que essas informações são enviadas.

¹www.cbpf.br/sun/pdf/udp.pdf

²www.json.org/

3.5 INTERFACE COMPORTAMENTO - AÇÃO

Assim como o Comportamento, o módulo de Ação também recebe informações através do uso do protocolo UDP utilizando o formato JSON, porém, esses dados são repassados do Comportamento para ele. O módulo Ação fica então responsável por executar movimentos como chute com uma das pernas, caminhar, desviar de obstáculos, dentre outros tipos de comportamentos recebidos como mensagem.

Um ponto que merece destaque é que o Comportamento trabalha com alto nível, dessa forma, ele envia para o módulo de ação um comando, onde esse possui três parâmetros. A seguir há um exemplo de entrada de dados da Ação.

```
“movement_vector”: [
    5,
    -0.9776952725551101,
    -0.9776952725551101
]
```

Como pode ser analisada, a mensagem transmitida possui três parâmetros, que é um vetor de três coordenadas (r , θ , ϕ). Sendo assim, os dois primeiros valores, r e θ , representam coordenadas polares que definem um movimento. Já ϕ é o ângulo de rotação que o robô deve fazer em torno do próprio eixo.

3.6 SUMÁRIO DO CAPÍTULO

Neste capítulo relatou-se um panorama global sobre a organização da arquitetura presente no Grupo de Pesquisa TAURA para atuais e futuros projetos. Tratou-se detalhadamente sobre os módulos existentes, e os meios de comunicação entre esses módulos utilizando o protocolo UDP e mensagens estruturadas no formato JSON.

O próximo capítulo trata especificamente da descrição do domínio de futebol de robôs humanoides da equipe Taura Bots e contém os detalhes sobre a estrutura utilizada no presente projeto. Apresenta-se classes e objetos, inferências e gráficos sobre a organização da Ontologia para a representação do conhecimento e descrição lógica sobre o ambiente de futebol de robôs humanoides.

4 DESCRIÇÃO DO DOMÍNIO

Há diversas pesquisas envolvendo ontologias com o intuito de facilitar o aprendizado de linguagens. Devido a ontologia funcionar em estrutura conceitual e possuir uma interface de navegação visual para os objetos de aprendizagem pode funcionar muito bem como uma ferramenta mental para o aprendizado (GANAPATHI; LOURDUSAMY; RAJARAM, 2011). Em Ganapathi; Lourdusamy; Rajaram (2011) é discutida uma abordagem para o desenvolvimento da ontologia projetada para o ensino de programação JAVA.

Para Pierrakeas; Solomou; Kameas (2012) a compreensão de linguagens requer desenvolvimento de representações sistemáticas das construções léxicas dos princípios semânticos da língua. Assim, ontologias se tornam úteis devido serem ticas estruturas de representação do conhecimento, podendo ser utilizadas para modelar conceitos do domínio.

Assim sendo, como forma de descrever o domínio de futebol de robôs não foi diferente, foi criada uma ontologia, vale ressaltar que o intuito da criação da mesma é facilitar a programação da linguagem Maia, uma vez que a ontologia limita o domínio bem como faz a utilização de algumas palavras reservadas da linguagem. Um outro ponto relevante da criação da ontologia é facilidade da forma de compartilhar conhecimento de tal domínio. Assim a Ontologia facilita a implementação de estratégias de jogadas de futebol de robôs humanoides. Para isso, é feito uma classificação de objetos do ambiente. São também apresentados definições das relações entre esses objetos, bem como a estruturação do agrupamento das classes e os conceitos envolvidos para tal.

4.1 ESPECIFICAÇÃO DA ONTOLOGIA

A especificação da Ontologia se dá, principalmente, de acordo com a classificação dos objetos que podem ser identificados para o domínio de futebol de robôs. A seguir são apresentadas a classificação desses objetos dentro das classes, bem como outras características provenientes da ontologia.

4.1.1 Classes

As ontologias na linguagem OWL têm sempre como topo hierárquico a classe Thing que é o domínio total. O domínio deste trabalho é o futebol de robôs na Liga RoboCup, mas o propósito desta ontologia não é representar todos os conceitos e termos associados ao futebol em geral. O domínio associado a uma partida de futebol apresenta muitos

tipos diferentes de conceitos, nesse caso, esses conceitos estão agrupados por áreas conceituais, onde conceitos do mesmo tipo podem se encaixar. A seguir são descritas alguns dos principais conceitos ou classes:

- *Actions* - indica inferências realizadas, ou seja, estratégia definida pelo robô perante o estado atual do jogo. As ações que podem ser controladas são *Kick* (chute), *Walk* (caminhar) e *Neck* (pescoço). No entanto, há submódulos inseridos em alguns desses. No caso do chute os submódulos são: *PassS*, que é um chute fraco para um alvo destino próximo; *PassM*, passe com força média; *PassL*, passe longo onde o alvo está em uma distância maior; por último o *KickToGoal* que é um chute cujo destino é a goleira. Já referente a caminhada os módulos podem ser: *WalkBall*, caminhar em direção a bola; *WalkTurnBall*, girar em torno da bola, esse comando é utilizado para posicionar-se no lugar correto antes do chute; *RotateAround*, giro em torno do próprio eixo, ou seja, girar o corpo; *WalkLeft*, para o robô goleiro ir para a esquerda; *WalkRight*, para o robô goleiro ir para a direita; *Walk_*, caminhar para uma determinada direção.

- *Object* - Abreu(2009) definiu os objetos como como abstratos e concretos. Sendo abstratos o time ou equipe. Já os concretos podem se subdividir em fixos e não fixos. Os objetos não fixos são aqueles que mudam sua localização. Por exemplo, a bola, outros robôs e objetos desconhecidos que estiverem no campo durante uma partida. Os objetos fixos são aqueles que não alteram a sua posição, tais como os postes e as linhas do campo.

- *Time* - essa classe representa o tempo decorrente de jogo, é bastante importante, pois de acordo com o resultado que é preciso, pode-se fazer estratégias levando em consideração o tempo. Ou seja, se há pouco tempo e o resultado desejado é uma vitória, o posicionamento do time deve ser no ataque.

- *Region* - correspondem as definições no domínio do espaço, regiões. As regiões são basicamente áreas do campo onde os jogadores estão colocados ou onde determinadas ações ocorrem.

- *State* - nessa classe é possível definir a estratégia inicial adotada no jogo. Pode ser ataque ou defesa.

- *Scoreboard* - essa classe corresponde ao placar com que o jogo está.

A Figura 4.1 apresenta um esquema das áreas conceituais abordadas na ontologia, para a representação de um jogo de futebol de robôs na RoboCup. Todos os conceitos estão agrupados segundo uma hierarquia de classes. A Figura 4.2 apresenta a hierarquia

das classes dos principais conceitos para tal domínio, como pode ser observado, algumas classes possuem um conjunto de subclasses, como é o caso da classe Action.

Figura 4.1 – Áreas conceituais da ontologia

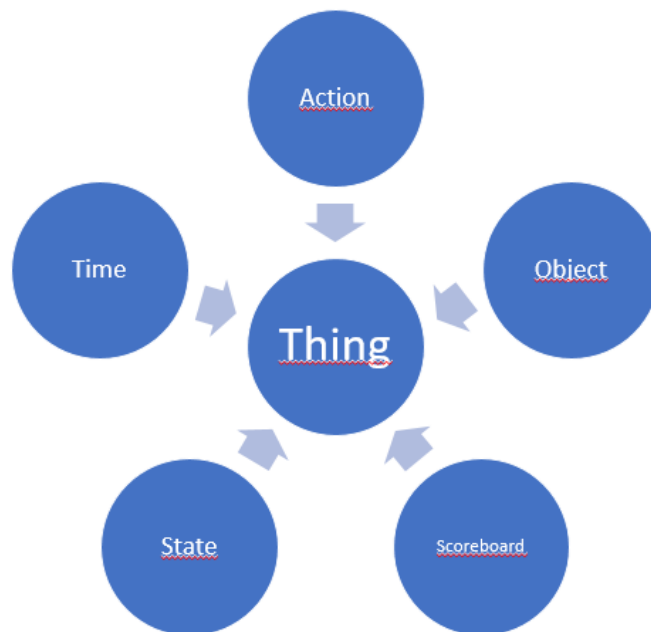
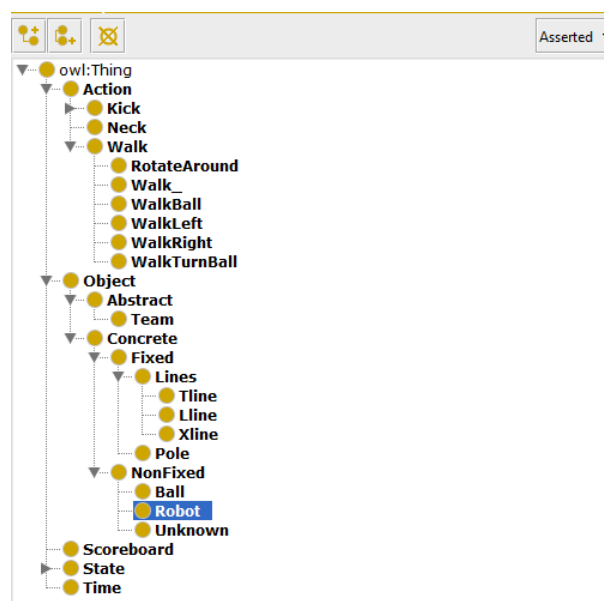


Figura 4.2 – Estruturação das classes da ontologia



Ainda dentro das classes, é possível fazer algumas definições a respeito das mesmas. A Figura 4.3 traz um exemplo disso, onde a classe *Robot* (robô), é subclasse de *NonFixed* (não fixo). Além disso, é representado *odomain* (domínio) e *range* (alcance) da propriedade de objeto *hasPartner*. Outro ponto representado ainda são duas instâncias de robôs, Juarez e Bender.

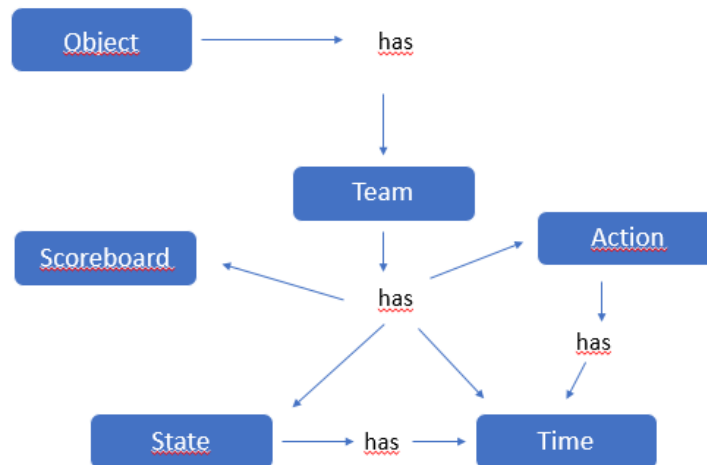
Figura 4.3 – Relações conceituais



4.1.2 Propriedades e relações

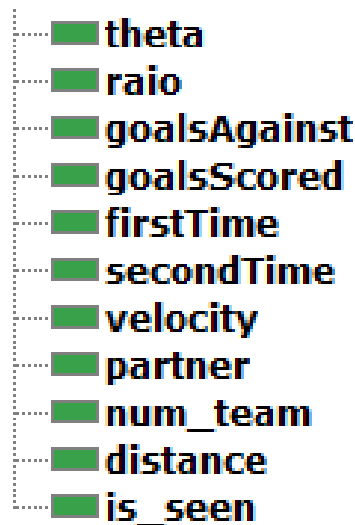
A descrição de relações em uma ontologia se dá através da propriedade "*Object properties*". Na Figura 4.4 são apresentadas essas propriedades presentes na ontologia através de um pequeno mapa conceitual. Onde numa análise minimalista dessa figura é possível observar que o *Team* (time) possui um *State* (estado), ou seja, o time estará em um estado de jogo que no caso pode ser ataque, por exemplo.

Figura 4.4 – Relações conceituais



Os atributos de uma ontologia referem-se às relações entre dados e conceitos. Estes dados na linguagem OWL são representados através de "*Data properties*". A Figura 4.5 representa algumas das propriedades de dados presente na ontologia, destaco "raio" e "theta" que são muito utilizados para descobrir saber a localização que outro objeto se encontra.

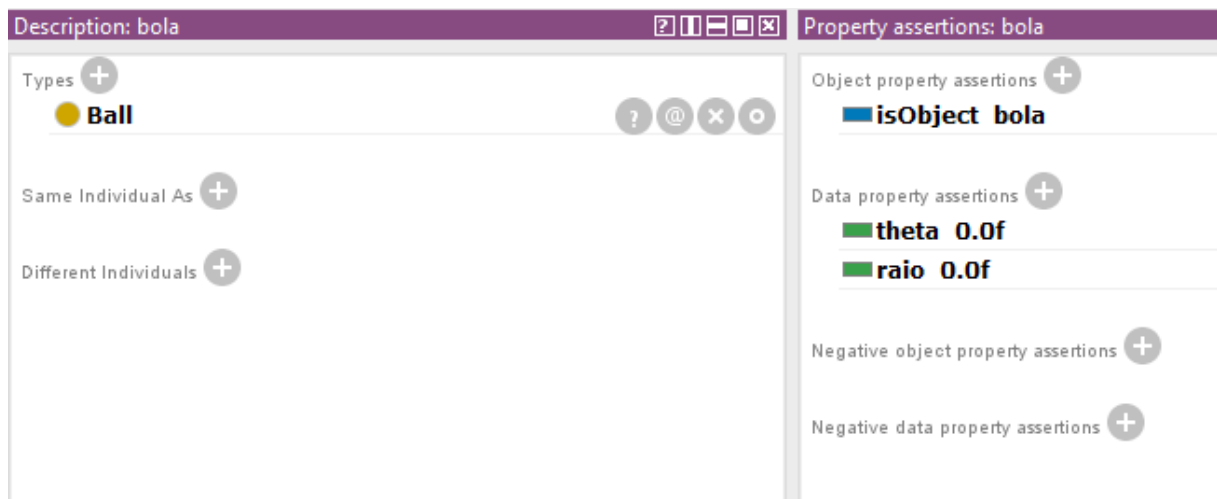
Figura 4.5 – Propriedades de dados



4.1.3 Instâncias e Regras

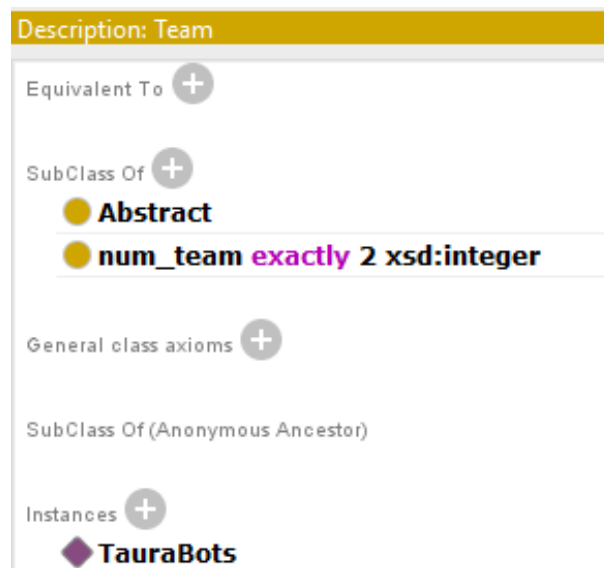
Instâncias nada mais são que dados reais da base de conhecimento. Na Ontologia desenvolvida a bola pode ser classificada conforme a Figura 4.6, onde existe uma instância de Ball a qual possui suas propriedades de dados e de objetos. As propriedades de objetos na Ontologia definem a quais domínios este objeto pertence e com isso tem-se as formas de manipulação deste objeto da classe Ball, tal como "isObject" que define que pertence ao domínio do "Object". Já as propriedades de dados tratam dos parâmetros ou atributos existentes neste objeto bola, tais como: "raio", valor o qual define a distância em relação ao objeto; "theta", determina o ângulo de direção até o objeto.

Figura 4.6 – Instância de bola



Já referente a regras criadas na ontologia são destacadas as que englobam a dinâmica do jogo diretamente, como as próprias regras da competição da RoboCup, como é o caso do número de objetos que pode haver em campo. No caso de bola é apenas uma, já postes são até quatro (sendo que dois postes equivalem a uma goleira), robôs podem existir até dois de um mesmo time e o número de total de times em uma partida é de dois. A Figura 4.7 mostra o número máximo de times que a classe *Team* (time) pode possuir, que no caso são dois, tal qual o futebol de robôs.

Figura 4.7 – Propriedades de dados



4.2 SUMÁRIO DO CAPÍTULO

Neste capítulo apresentou-se os conceitos fundamentais para compreensão do domínio utilizado na presente pesquisa. Ademais, foi apresentada uma Ontologia para descrever um campo de futebol, onde relatou-se a metodologia aplicada, com isso, facilitará a programação de comportamentos do robô em uma partida de futebol.

O próximo capítulo trata especificamente do módulo de criação do comportamento da equipe Taura Bots e os detalhes sobre a modelagem proposta no presente projeto. São apresentadas a arquitetura geral do trabalho, requisitos de entrada e saída da linguagem, estrutura da linguagem, dentre outros itens.

5 METODOLOGIA

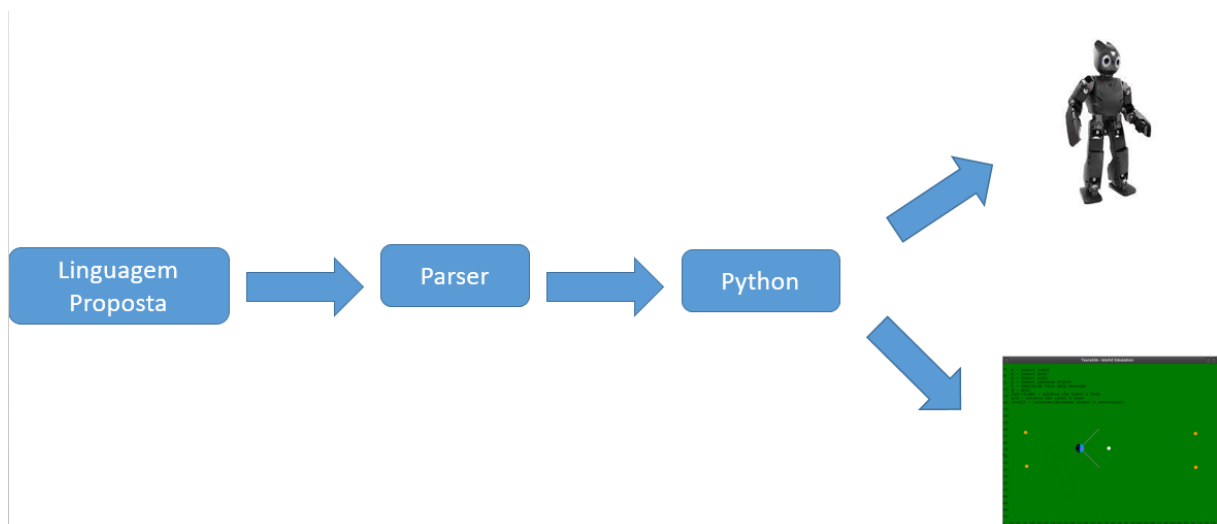
Nesse capítulo são abordados algumas das especificações da linguagem Maia além do parser. Como forma de facilitar a compreensão para programação foi definida a ontologia apresentada anteriormente que define conceitualmente o robô e o contexto de futebol no qual o mesmo está inserido.

5.1 ARQUITETURA DO PROCESSO

Na Figura 5.1 pode ser observado um diagrama, onde há a linguagem Maia, cujo a mesma possui estrutura bastante parecida com a linguagem RS, que segundo Toscani (1993) é mais intuitiva para programação de robôs que interagem com ambiente externo. Isso se dá ao fato dela ser simples e ter o formato "Condição \Rightarrow Ação" o que facilita o raciocínio. Além disso, robôs também tendem a ter esse mesmo formato, dessa forma a linguagem é bem interessante para o presente trabalho. Outro item positivo de Maia é o fato de ser bem delimitada para que possam aprender apenas os comandos e funcionalidades voltados especificamente para o futebol de robôs.

Para a criação de Maia foi definida uma gramática e um conjunto de regras de produção que definem as condições de geração das palavras da linguagem. A partir disso, é feito um parser que traduzirá Maia para Python sendo a contribuição na qual o grupo Taura Bots irá beneficiar-se. Visto que há uma nova camada para geração de comportamentos. Vale ressaltar que a linguagem Python é utilizada pelo grupo para programação de comportamentos.

Figura 5.1 – Abordagem do trabalho



A partir disso, são criados comportamentos na linguagem proposta, sendo que foi utilizado o ANTLR¹ uma única vez para fazer análise léxica, sintática e semântica. Posteriormente é utilizado o NetBeans², que com o conjunto de classes geradas pelo ANTLR, recebe um arquivo de entrada dessa nova linguagem ocorrendo a validação e tradução em Python.

5.2 REQUISITOS

A definição de requisitos teve como base o trabalho de Montenegro (2015), onde no simulador 2D, os objetos de entrada para o comportamento são oriundos da visão em formato JSON. Após isso, são processados pelo módulo de comportamento e enviados também em JSON para a caminhada executar as ações necessárias.

5.2.1 Requisitos de Entrada

Os requisitos de entrada do simulador provém da visão. Assim são objetos que a mesma consegue captar. Como pode ser visto no exemplo a seguir:

```
{
  "head_angle": 0.0, // angulo da cabeca
  "objects_list": [
    {
      "kind": "pole_1", // tipo do objeto
      "position": [ // posicao objeto
        239.9031999919439, // r
        -0.6876952725551101 // theta
      ]
    }
  ]
}
```

Os objetos que a visão reconhece são de oito tipos: ball, pole, robot, line, X_line, T_line, L_line e unknown. No entanto, foram criados outros três objetos que servirão como entrada para a linguagem. Na sequência é feita uma descrição com cada um deles.

- Tipo ball - representa a bola e segundo as regras da RoboCup é possível ter apenas uma em campo.

¹<http://www.antlr.org/>

²<https://netbeans.org/downloads/>

- Tipo pole - representa as traves das goleiras e possuem cor branca, tal qual do jogo de futebol de humanos. A linguagem permite até quatro traves, sendo que a identificação da mesma é denotada por poleX, sendo que X é um índice variando de 1 até no máximo 4.
- Tipo robot - representa os robôs em campo. Os robôs possuem a identificação robotX_o e robotX_p. Sendo robotX_o o robô oponente. Já o robotX_p retrata parceiros do mesmo time. O X, assim como a representação das traves, funciona como um índice e também varia de 1 até 3.
- Tipo line - representa linhas do campo e é identificado pela cor branca e diferenciado do objeto poste pela variação entre a cor branca e a cor verde do gramado.
- Tipo X_line - variação do objeto line. É representada pela intersecção de duas linhas formando um ângulo de aproximadamente 90 graus entre qualquer uma delas. Esse é o caso da linha no meio de campo no encontro com a linha do círculo central.
- Tipo T_line - outra variação do objeto line. Como exemplo, são as linhas de intersecção entre a linha do meio de campo e da linha lateral. O outro caso, é a intersecção entre a linha de fundo e a linha da grande área.
- Tipo L_line - mais uma variação do objeto line, encontrado nas intersecções das linhas laterais com a linha de fundo e nas intersecções das linhas (cantos) da grande área.
- Tipo time - caracteriza o tempo decorrido de jogo.
- Tipo scoreboard - descreve o placar do jogo. O que dependendo do resultado que se faz necessário poderá mudar as estratégias de jogadas.
- Tipo neck - representa o pescoço do robô que é um objeto importante para saber a localização de outro objeto e o deslocamento que deve ser feito caso queira ir até ele.
- Tipo unknown - são os demais objetos. São objetos desconhecidos que não estão categorizados nessa lista apresentada.

Como sinais de entradas boa parte desses objetos possuem coordenadas polares, r e θ . No entanto, *time*, *scoreboard* e *neck* possuem uma diferente representação. Para *time* os parâmetros de entrada são minutos e segundos, que corresponde ao tempo de jogo. Já *scoreboard* tem o número de gols do meu time(p) e o número de gols que o oponente(o) tem. Já para *neck*, há variação de ângulo do robô na vertical(y) e horizontal(x). A seguir pode ser observado um exemplo de entrada.

```
in: [ball(r, theta), robot(r, theta), pole(r, theta),
     Tline(r, theta), Xline(r, theta), Lline(r,theta),
     line(r, theta), unknown(r,theta),
     time(min, seg), scoreboard(p, o), neck (x, y)]
```

5.2.2 Requisitos de Saída

Os requisitos de saída correspondem as ações que podem ser efetuadas pelo módulo da caminhada. Assim como os requisitos de entrada, os de saída também usam o formato JSON no trabalho descrito por Montenegro (2015). Na sequência é mostrado um possível código de saída em formato JSON.

```
{
  "movement_vector": [
    5, // r
    -0.9758561608856633, // theta
    -0.9758561608856633 // phi
  ],
  "index": 0,
  "head_angle": -0.9758561608856633 // angulo da cabeca
}
```

A partir disso foram definidos alguns itens como ações para execução. A seguir podem ser visualizadas as saídas possíveis, bem como a estrutura das mesmas.

```
out: [walk(r, theta, phi), kick(r, theta), defend(x),
      passS(r, theta), passM(r, theta), passL(r, theta),
      walkBall (r, theta), walkLeft (r, theta), walkRight(r, theta),
      walkTurnBall(r, theta), rotateAround (r, theta)
    ]
```

Onde a definição de cada uma dessas ações está representada como:

- Ação walk - comando utilizado para que o robô caminhe. Diferente das demais ações, essa possui três parâmetros: r, theta e phi. Sendo r a distância no qual o objeto se encontra e theta o ângulo em relação a visão. Já o phi é utilizado no robô real, que indica o ângulo que o robô gira no seu próprio eixo, que se dá uma vez que a posição da cabeça pode ser diferente do corpo, assim o corpo do robô deve ficar de acordo com a visão para não haver discrepâncias na movimentação e consequentemente

na chegada até o alvo. Além disso, é o movement vector que é setado com esses valores para indicar a nova posição do robô.

- Ação walkBall - comando utilizado para o robô caminhar até a bola.
- Ação walkLeft - usado quando deseja-se mover o robô para a esquerda, no entanto esse movimento é executado de lado.
- Ação WalkRight - com esse movimento o robô anda de lado, sendo da esquerda para a direita.
- Ação walkTurnBall - essa ação serve para que o robô se posicione em volta da bola em um ângulo desejado.
- Ação kick - movimento para que o robô chute a bola.
- Ação defend - são os movimentos que somente o robô goleiro vai efetuar, que são defesas. Andar para esquerda, representado por -1. Ir para a direita, simbolizado pelo número 1. Por último manter-se parado, denotado pelo zero.
- Ação passS - movimento de chutar, no entanto é um chute muito fraco, é mais usado para tocar para um parceiro que está bem próximo.
- Ação passM - também é um movimento de chutar, a variação desse caso é com força intermediária, quando o companheiro está em um nível intermediário entre intermediário e próximo de distância em relação a posição do que está com a bola.
- Ação passL - outra variante da ação de chutar, é usado quando o parceiro está um pouco longe. É um pouco mais fraco que o chute a gol.

5.2.3 Estrutura da Linguagem

A linguagem proposta possui uma estrutura bastante similar a da linguagem RS. Sendo assim possui: sinais de entrada, sinais de saída, variáveis, estados e estado inicial. No entanto a principal mudança se dá pelo maior número de funcionalidades que estão sendo abordadas. Assim, a seguir são elucidados tais itens.

- Sinais de entrada - correspondem aos objetos aos quais a linguagem aborda. São objetos provenientes da visão juntamente com o pescoço do robô, ou seja, são somente esses itens que podem conter, respeitando as peculiaridades de cada um, como o caso das traves onde é possível ter no máximo quatro itens. É representado por in:[objects].

- Sinais de saídas - referem-se aos movimentos ou ações que o robô pode executar, isto é, repassa as informações para o módulo da caminhada. Como forma de caracterizar tal sinal é usado `out:[actions]`.
- Variáveis - são elementos definidos pelo usuário para que ele possa utilizar na formulação de suas jogadas. Somente as variáveis definidas nesse item que poderão ser usadas na criação de comportamentos. Exprimidas como `var:[variables]`. Além disso, faz-se necessária a colocação do tipo da variável que será usada.
- Estados - representa os estados internos do robô. As opções possíveis são defesa (defense), ataque (attack) e goleiro (goalkeeper). É representado por `states:[state]`.
- Estado inicial - é o estado no qual o robô deve começar o jogo. Os robôs não podem estar em mais de um estado simultaneamente. Sendo assim, deve selecionar um dos estados declarados. Como caracterização é usado a instrução `initially:[state]`.

Na sequência é apresentada a estrutura básica utilizando algumas das funcionalidades desses conjuntos.

```
in: [ ball(r, theta), robot(r, theta), pole(r, theta),
      Tline(r, theta), Xline(r, theta), Lline(r, theta),
      line(r, theta), unknown(r, theta),
      time(min, seg), scoreboard(p, o), neck (x, y) ]
out: [ walk(r, theta, phi), kick(r, theta), defend(x),
       passS(r, theta), passM(r, theta), passL(r, theta) ]
var: [ float cont ];
states: [ attack, defense, goalkeeper, outro ];
initially: [ attack ];
```

Após esse bloco básico são necessárias o conjuntos de instruções para determinar alguma ação. Tais instruções ou regras possuem, seguindo a linguagem RS, a seguinte estrutura de Condição -> Ação. Onde a condição nada mais é que o conjunto de objetos recebidos pela entrada (in), assim essas, ativam regras e disparam ações. Já as ações correspondem ao conjunto de instruções que terão como resultado a ação que deve ser executada.

Para a execução de qualquer atividade, deve estar setado o estado interno do robô, este pode variar no decorrer do jogo, porém somente um estado pode estar sendo usado por vez. Para mudar o estado do robô é usado o comando `up(state)`. Supondo que o robô estivesse com o comando `initially:[attack]`; nas regras é possível mudá-lo para que o mesmo mude seu estado para defesa (`up[defense];`)

Quanto as ações que serão executadas, são representadas pelo comando `emit[actions]` e somente uma ação pode ser executada por vez, tal qual os estados internos do robô. Assim, é possível criar regras para definir a ação.

Um exemplo mais elaborado, com mais funcionalidades, dessa linguagem pode ser observado na sequência. Como pode ser analisado, há algumas condições e ações.

5.3 DEFINIÇÃO DA GRAMÁTICA E IMPLEMENTAÇÃO

Para desenvolver a linguagem Maia, se fez necessária a definição de uma gramática bem como o conjunto de regras de produção, que juntos definem as condições de geração das palavras, ou seja, as regras semânticas, que irão estabelecer o significado das estruturas da linguagem.

Ademais, se faz fundamental a construção de um tradutor que irá gerar o código final. Como linguagem base para a criação da gramática foi a linguagem Maia e a ferramenta que irá auxiliar na construção do tradutor foi o ANTLR.

5.3.1 Gramática

Para a criação da gramática, inicialmente precisou-se definir os *tokens*. Esses *tokens* estão categorizados como: números palavras reservadas e identificadores. Os números são inteiros, representados por DIG, ou flutuantes (*float*), representados por NUM.

DIG: `[0-9]+;`

NUM: `('-')?[0-9]+'.'[0-9]+;`

Já as palavras reservadas correspondem ao conjunto de palavras que não podem ser usadas como identificadores. Por exemplo, se têm-se a palavra *module* como sendo palavra reservada, logo não poderá ser criada nenhuma variável com mesmo nome, pois o uso dessas palavras é de uso exclusivo da gramática da linguagem. A Tabela 5.1 apresenta as palavras reservadas, sendo boa parte delas utilizadas pela ontologia na definição do domínio.

Tabela 5.1 – Palavras Reservadas

| | | | | |
|--------|------------|---------|--------------|--------------|
| ; | # | [|] | passM |
| (|) | { | } | passL |
| < | > | == | >= | walkBall |
| != | phi<= | := | , | walkLeft |
| & | + | - | * | walkRight |
| / | boolean | true | false | walkTurnBall |
| module | in | out | var | passS |
| states | initially | ball | pole | kick |
| robot | line | Xline | Tline | defense |
| Lline | r | theta | phi | walk |
| time | scoreboard | unknown | rotateAround | defend |
| if | else | up | emit | int |

Os identificadores são todos os nomes de variáveis que podem ser definidos pelo programador. Por exemplo, float cont, "cont" é um identificador. Como forma de representação dessas variáveis existe o token ID.

ID: [_a-zA-Z] [_a-zA-Z0-9]*;

Após isso foi definido o restante da gramática, parte dela é apresentada a seguir, vale enfatizar a similaridade com alguns itens da ontologia, como é o caso do número máximo de postes que podem ter no campo de futebol, representado por "POLE: 'pole'[1-4]", ou seja, limita que podem existir até quatro postes. A gramática completa encontra-se no Apêndice A.

```

expr: name body;
/* module x: */
name: 'module' ID DP;
/* { in out var states initially rules */
body : AC input output var stts init rules* FC;
input : 'in' DP OC parametersIn pInput;
/* ball */
parametersIn : 'ball' | POLE | (ROBOT_O | ROBOT_P | ROBOT) | 'Xline' |
               'Tline' | 'Lline' | 'line' | 'unknown' | 'time' | 'scoreboard'
POLE: 'pole'[1-4];

```

A partir disso, com a utilização do ANTLR pode-se testar e validar tal gramática. Para isso, são geradas cinco classes java na execução do ANTLR que são:

- MaiaParser.java - parser que reconhece a gramática.

- `MaiaLexer.java` - o lexer para gramática, esta classe realiza o tokenizing do input, através do reconhecimento dos símbolos gramaticais.
- `MaiaTokens.java` - arquivo que armazena os tipos de tokens específicos da gramática.
- `MaiaListener.java` - arquivo que pode-se usar para navegar na árvore de resultados do parser, possui eventos que são disparados ao se percorrer a árvore.
- `MaiaBaseListener` - interface para o listener da árvore de resultados.

Depois dessa etapa, é necessário apenas selecionar o arquivo de entrada para que o mesmo seja validado de acordo com a gramática proposta.

5.3.2 Parser

Para implementação do tradutor para linguagem Python, foram utilizadas as classes java geradas pelo ANTLR a partir da gramática de entrada. A linguagem se deu pelo fato de tanto o simulador 2D quando o robô real usarem Python como linguagem do módulo do comportamento.

Foi utilizado o plugin versão 4.5 do ANTLR para a construção de tal implementação. A plataforma de desenvolvimento utilizada foi o NetBeans, a linguagem responsável por fazer a tradução.

Vale ressaltar que nem todos os comandos da linguagem proposta, se traduzidos para Python irão funcionar no simulador ou robô. Isso dá-se uma vez que a linguagem possui um número maior de funcionalidades, visando uma maior combinação de elementos na criação de comportamentos.

Um outro fator se dá a dificuldade em que o robô possui no módulo de Caminhada, sendo que o mesmo utiliza o código original do robô com pequenas modificações. Assim, não há movimentos muito elaborados. Porém, a ideia é que assim que esses problemas sejam solucionados, essas funcionalidades da linguagem proposta possam ser usadas.

5.3.2.1 Saída do Parser

Como responsável por percorrer as estruturas montadas e concatená-las é utilizado o método `getOutput`. Para isso, tem-se um trecho de código padrão, que independente da estrutura montada para o comportamento, esse trecho estará presente, que é o cabeçalho do código. Ele contém as instruções que servem para importar os módulos do simulador e por carregar as informações percebidas do mundo.

```
# cabecalho
from MindInterface import Simulation
from MindInterface.config import *

import time
from math import pi

robot = Simulation.start()

while robot.updateSimulation():
    world = robot.perceiveWorld()
    if not world:
        sys.exit("No world received")
```

Após isso são impressas as variáveis declaradas pelo usuário. Pode ser analisado que há tipagem na declaração das mesmas, apesar do Python não possuir, caso seja feito um tradutor para outra linguagem que tenha tipagem, a mesma já está sendo abordada.

Código na linguagem proposta:

```
var: [float x, boolean y, int w];
```

Saída em Python:

```
x = None
y = None
w = None
```

Na sequência aparecem os objetos de entrada (in). Para isso se faz necessário percorrer a lista deles de acordo com o que o módulo de visão do robô captou no ambiente.

Código na linguagem proposta:

```
in: [ball(r_ball, t_ball), robot1_p(r_robot1_p, t_robot1_p),
pole1(r_pole1, t_pole1), pole2(r_pole2, t_pole2)]
```

Saída em Python:

```
ball = None
robot1_p = None
pole1 = None
for obj in world.objects_list:
    if obj.kind == "ball":
        ball = obj
        r_ball = ball.position.r
        t_ball = ball.position.a
    if obj.kind == "robot1_p":
```

```

robot1_p = obj
r_robot1_p = robot1_p.position.r
t_robot1_p = robot1_p.position.a
if obj.kind == "pole1":
    pole1 = obj
    r_pole1 = pole1.position.r
    t_pole1 = pole1.position.a
if obj.kind == "pole2":
    pole2 = obj
    r_pole2 = pole2.position.r
    t_pole2 = pole2.position.a

```

Dando continuidade, são processadas as estratégias do jogo seguindo as Condições -> Ações. Primeiramente as condições são traduzidas, levando em conta detectados pela visão.

Código na linguagem proposta:

```
ball(r_ball , t_ball) & pole1(r_pole1 , t_pole1)#attack ->
```

Saída em Python:

```
if ball and pole1 and attack:
```

Logo após, são processadas as instruções up(state), caso ela exista na definição do comportamento. No entanto apenas um estado pode estar ativo, sendo assim, o mesmo é setado como true e os demais como false.

Código na linguagem proposta:

```
up(defense);
```

Saída em Python:

```

attack = false
defense = true
goalkeeper = false
outro = false

```

Na sequência são processadas as demais instruções que fazem parte da linguagem. Essas são responsáveis pela programação das estratégias e também pela emissão das saídas, ou seja, dos movimentos que serão executados. Existem if e else para tomada de decisão, sendo que o if pode aparecer sozinho ou com um else. No entanto não é possível aparecer um else sozinho, ele deve ser precedido de um if.

Código na linguagem proposta - instrução if :

```

if (r_ball > 1){
    ...

```

```
}
```

Saída em Python:

```
if r_ball > 1:
    ...
```

Código na linguagem proposta - instrução if...else:

```
if (r_ball > 1){
    ...
}
else { ... }
```

Saída em Python:

```
if r_ball > 1:
    ...
else
    ...
```

Além dessas funcionalidades, é possível também fazer atribuição a variáveis. Para isso é utilizado `:=` que é equivalente ao sinal de igual.

Código na linguagem proposta - instrução de atribuição :

```
cont := cont + 10 + 1.2;
b := false;
```

Saída em Python:

```
cont = cont + 10 + 1.2
b = false
```

Para execução dos movimentos é utilizado o `emit` para sinalizar que um movimento será efetuado, além disso, também deve ser sinalizado tal movimento. Outro item necessário é o restante dos atributos necessários para execução do movimento de acordo com as definições de cada um.

Código na linguagem proposta - instrução emit :

```
emit(walk(r_ball , 1.2 , 1));
emit(kick(t_ball , 1));
emit(defend(1));
emit(walkBall(r , theta));
emit(walkLeft(r , theta));
emit(walkRight(r , theta));
emit(walkTurnBall(r , theta));
emit(rotateAround(r , theta));
```

Saída em Python:

```
robot.setMovementVector(Point2(r_ball, 1.2, 1))
robot.setKick(t_ball, 1)
robot.setDefend(1)
robot.setWalkBall(r, theta)
robot.setWalkLeft(r, theta)
robot.setWalkRight(r, theta)
robot.setWalkTurnBall(r, theta)
robot.setWalkRotateAround(r, theta)
```

Após essa fase é possível obter um código executável utilizado no simulador de acordo com os critérios definidos em Montenegro (2015). Uma vez que no presente trabalho é feito apenas a parte do comportamento, logo, o código gerado é correspondente ao mesmo, sendo assim, é inserido dentro do arquivo IA.py do simulador de Montenegro (2015).

5.4 SUMÁRIO DO CAPÍTULO

Este capítulo apresentou a contribuição que Maia tratá ao grupo Taura Bots, bem como a quem deseja usá-la para criação de comportamentos. Para isso, foram abordados os requisitos e a estrutura dela. Além disso, foi apresentado o parser que fez a tradução dessa linguagem para Python.

O próximo capítulo traz estudo de caso utilizando várias funcionalidades da linguagem proposta, além disso, é feita uma tradução dela para Python e testes desse código gerado.

6 ESTUDO DE CASO E TUTORIAL DA FERRAMENTA

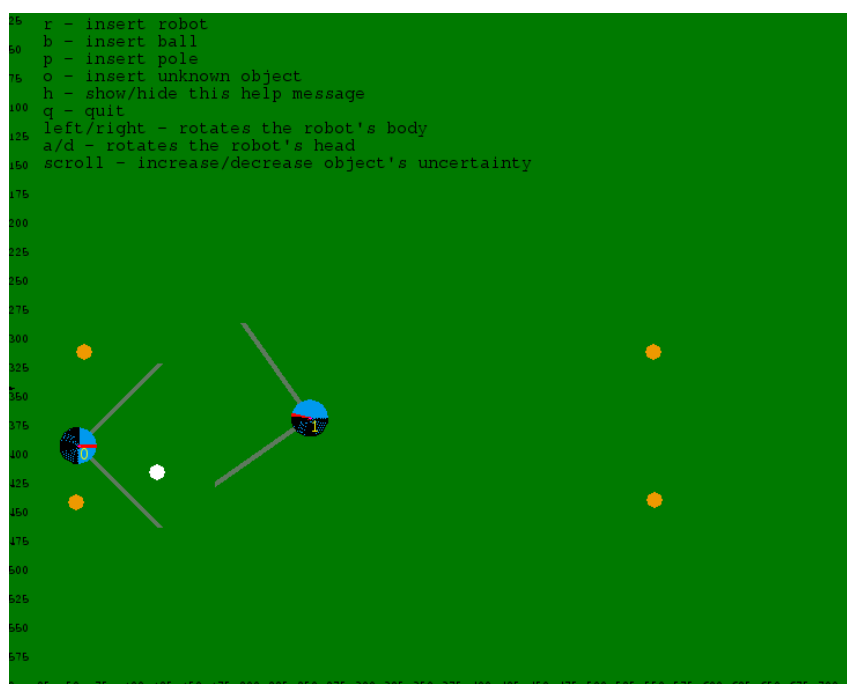
O presente capítulo aborda um estudo de caso utilizando as funcionalidades da linguagem criada. Além disso é demonstrado um tutorial para que facilite o uso da linguagem Maia, para isso, são especificadas as etapas executadas para que o código possa ser usado no simulador.

6.1 ESTUDO DE CASO

Esse estudo de caso possui uma entrada, correspondente a linguagem proposta no trabalho, e uma saída, linguagem Python utilizada no simulador e robô. Um problema ocorrido se dá pelo fato da Caminhada não suportar todos os movimentos propostos na linguagem. No entanto, para aqueles robôs que já desempenham tais movimentos, a linguagem torna-se mais dinâmica devido poder executar mais funcionalidades.

O exemplo a seguir é uma possível combinação de elementos feito utilizando Maia e foi testado no simulador 2D. Nesse caso o robô goleiro verifica se têm um robô adversário na sua frente, e se a bola está em seu campo de visão que está sendo representado na Figura 6.1. Onde os círculos laranja representam as goleiras, o círculo branco é a bola e os outros círculos com mais de uma cor e antenas representam os robôs. Como pode ser observado, tais antenas delimitam o ângulo do campo de visão do robô.

Figura 6.1 – Posição inicial da jogada



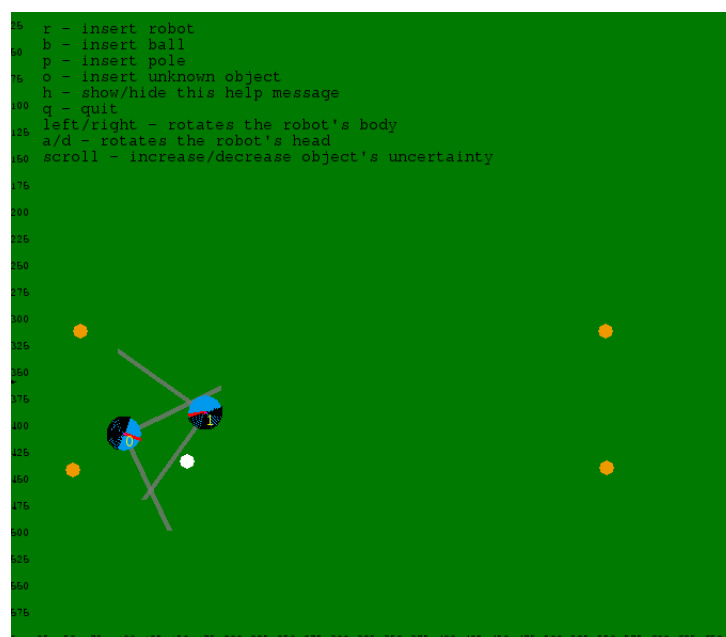
Já a Figura 6.2 os dois robôs se movimentaram e direção a bola e como pode ser observado o robô goleiro, que estava mais próximo, chegou primeiro que o robô adversário. Dessa forma ele tem a opção de chutar a bola.

Figura 6.2 – Robô goleiro chega até a bola



Por último, através da Figura 6.3 é o momento em que a jogada é finalizada, pois o chute do robô é executado. Nesse caso, o robô adversário perdeu contato visual com a bola. Assim, esses três momentos são considerados uma das muitas jogadas possíveis que o robô pode efetuar.

Figura 6.3 – Chute do robô goleiro



A seguir, mostra-se o funcionamento da linguagem proposta para execução dessa

jogada. Como pode ser observado, como entrada (*in*) tem-se uma bola e um robô opo-
nente. Já como saída (*out*) são apresentados todos os movimentos propostos pela lingua-
gem. Também são declaradas duas variáveis (*var*) do tipo *float*. O estado inicial (*initially*)
indica que o robô é goleiro (*goalkeeper*). Após esses passos, são feitas as regras (Condi-
ção \Rightarrow Ação) que combinadas vão informar qual ação o robô deve executar.

Como condição (linha 12) nota-se que faz-se necessário que o robô seja um goleiro
e tenha no seu campo de visão a bola e o robô adversário. Após isso, é feito um cálculo
(linha 13) com o intuito de saber qual a distância que o robô adversário está em relação a
bola. A linha 15 analisa quem está mais próximo, o robô goleiro ou o robô adversário, a
partir disso são executados comandos de acordo com essa condição. As linhas 15, 17 e
21 sinalizam qual ação deverá ser feita.

```
module robo1: {
  in: [ball(r_ball, t_ball), robot1_o(r_robot1_o, t_robot1_o)];
  out: [walk(r_w, theta_w, phi_w), defend(x),
        kick(r_k, theta_k), passS(s_r, s_t),
        passM(m_r, m_t), passL(l_r, l_t),
        walkBall(r_wb, theta_wb), walkLeft(r_wl, theta_wl),
        walkRight(r_wr, theta_wr), walkTurnBall(r_wt,
        theta_wt), rotateAround(r_ra, theta_ra)];
  var: [float r_pos, float t_pos];
  states: [attack, defense, goalkeeper, outro];
  initially: [goalkeeper];
    ball(r_ball, t_ball) & robot1_o(r_robot1_o,
    t_robot1_o)#goalkeeper ->
      r_pos := r_robot1_o - r_ball;
      if (r_pos < r_ball) {
        emit(walk(0, r_ball, t_ball));
        if (r_ball+5 < 10){
          emit(kick(1, 1));
        }
      }
      else {
        emit(defend(0));
      }
}
```

Dando continuidade é apresentada a tradução dessa linguagem para Python. As
primeiras linhas representam o cabeçalho do código, sendo padrão para qualquer entrada.
Posteriormente, são inicializadas as variáveis. Dentro do laço de repetição são analisados

os objetos de entrada e, após isso, são executadas as regras para do comportamento. A linha 27 equivale a "Condição" da linguagem recém apresentada. As demais linhas, abaixo dessa, representam a "Ação" que deverá ser executada pelo robô.

```
import time
from math import pi

robot = Simulation.start()

while robot.updateSimulation():
    world = robot.perceiveWorld()
    if not world:
        sys.exit("No world received")

    r_pos = None
    t_pos = None
    goalkeeper = True

    ball = None
    robot1_o = None
    for obj in world.objects_list:
        if obj.kind == "ball":
            ball = obj
            r_ball = ball.position.r
            t_ball = ball.position.a
        if obj.kind == "robot1_o":
            robot1_o = obj
            r_robot1_o = robot1_o.position.r
            t_robot1_o = robot1_o.position.a

    if ball and robot1_o and goalkeeper:
        r_pos = r_robot1_o - r_ball
        if r_pos < r_ball:
            robot.setMovementVector(Point2(0,
            r_ball, t_ball))
        if r_ball + 5 < 10:
            robot.setKick(1, 1)
        else:
            robot.setdefend(0)
```

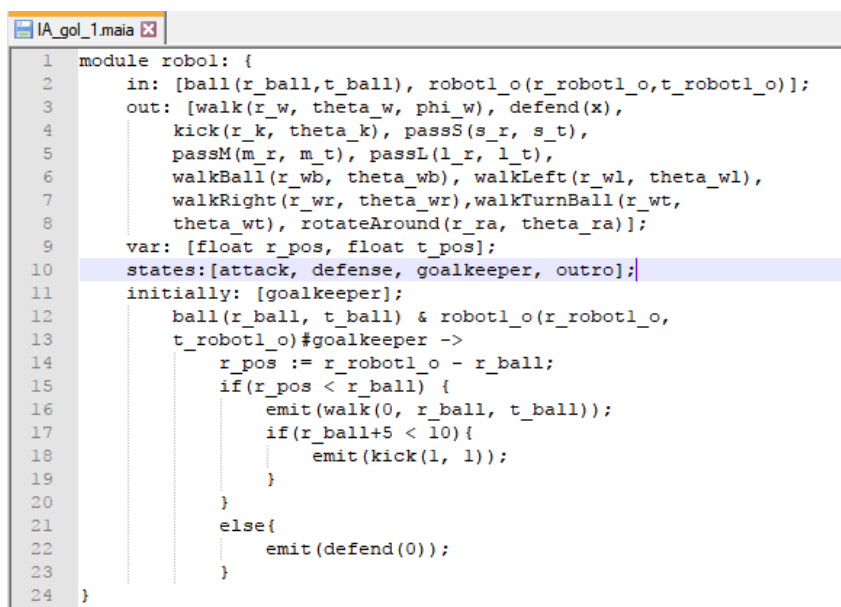
Esse estudo de caso demonstrou um pouco da diversidade de jogadas que podem ser criadas utilizando essa linguagem. Apesar de o futebol de robôs não possuir exatamente as mesmas características de futebol de humanos, além de estar em um nível inicial de pesquisas, já é possível notar a complexidade e a variação de jogadas que podem ser feitas.

6.2 TUTORIAL DA FERRAMENTA

Como apresentado anteriormente é feita uma gramática e validada com o ANTLR. A partir disso, com as classes geradas pelo ANTLR, é utilizado o NetBeans para a programação do tradutor. Sendo essa programação responsável pelas etapas de validação e tradução do código Maia para Python.

Inicialmente se faz necessária a criação de um arquivo com extensão ".maia" correspondente as especificações da linguagem proposta no trabalho. A Figura 6.4 apresenta um possível comportamento criado com a utilização de Maia. Como pode ser observado, no canto superior esquerdo há o nome do arquivo juntamente com a extensão, no caso "IA_gol_1.maia".

Figura 6.4 – Arquivo a utilizando linguagem Maia

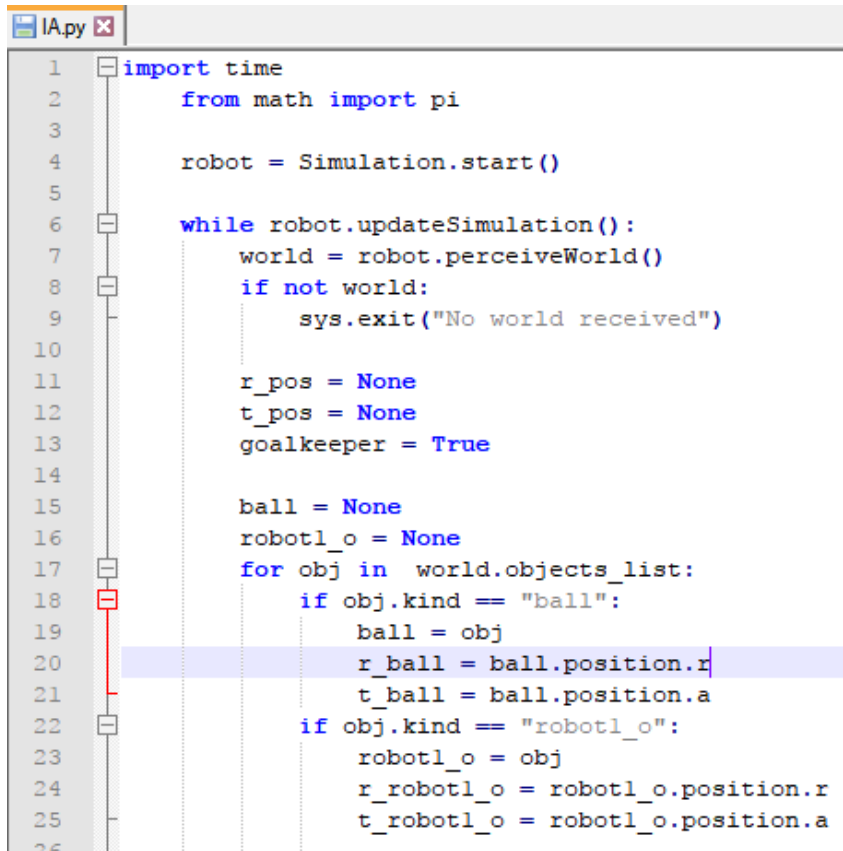


```

1 module robot: {
2   in: [ball(r_ball,t_ball), robotl_o(r_robotl_o,t_robotl_o)];
3   out: [walk(r_w, theta_w, phi_w), defend(x),
4         kick(r_k, theta_k), passS(s_r, s_t),
5         passM(m_r, m_t), passL(l_r, l_t),
6         walkBall(r_wb, theta_wb), walkLeft(r_wl, theta_wl),
7         walkRight(r_wr, theta_wr), walkTurnBall(r_wt,
8             theta_wt), rotateAround(r_ra, theta_ra)];
9   var: [float r_pos, float t_pos];
10  states:[attack, defense, goalkeeper, outro];
11  initially: [goalkeeper];
12  ball(r_ball, t_ball) & robotl_o(r_robotl_o,
13      t_robotl_o)#goalkeeper ->
14      r_pos := r_robotl_o - r_ball;
15      if(r_pos < r_ball) {
16          emit(walk(0, r_ball, t_ball));
17          if(r_ball+5 < 10){
18              emit(kick(1, 1));
19          }
20      }
21      else{
22          emit(defend(0));
23      }
24 }
  
```

Após essa etapa, deverá ser selecionado o diretório onde esse arquivo se encontra. Feito isso, deverá selecionar o diretório em que será salva a versão em Python. A Figura 6.5 mostra parte da saída gerada pelo programa, sendo que foi criado um arquivo com nome de "IA.py".

Figura 6.5 – Arquivo de saída do programa



```

1 import time
2     from math import pi
3
4     robot = Simulation.start()
5
6     while robot.updateSimulation():
7         world = robot.perceiveWorld()
8         if not world:
9             sys.exit("No world received")
10
11         r_pos = None
12         t_pos = None
13         goalkeeper = True
14
15         ball = None
16         robotl_o = None
17         for obj in world.objects_list:
18             if obj.kind == "ball":
19                 ball = obj
20                 r_ball = ball.position.r
21                 t_ball = ball.position.a
22             if obj.kind == "robotl_o":
23                 robotl_o = obj
24                 r_robotl_o = robotl_o.position.r
25                 t_robotl_o = robotl_o.position.a
26

```

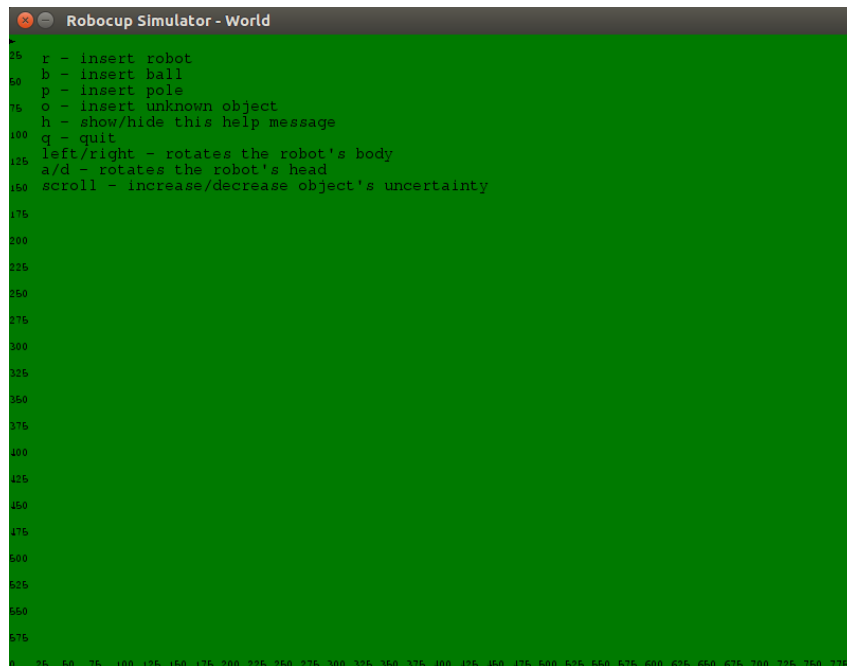
A Figura 6.6 apresenta de forma simplificada os passos utilizados nesse processo. Ou seja, o programa recebe um arquivo de entrada com formato ".maia", o processa fazendo validação e tradução, gerando como saída um arquivo com formato ".py"(Python).

Figura 6.6 – Principais passos do processo



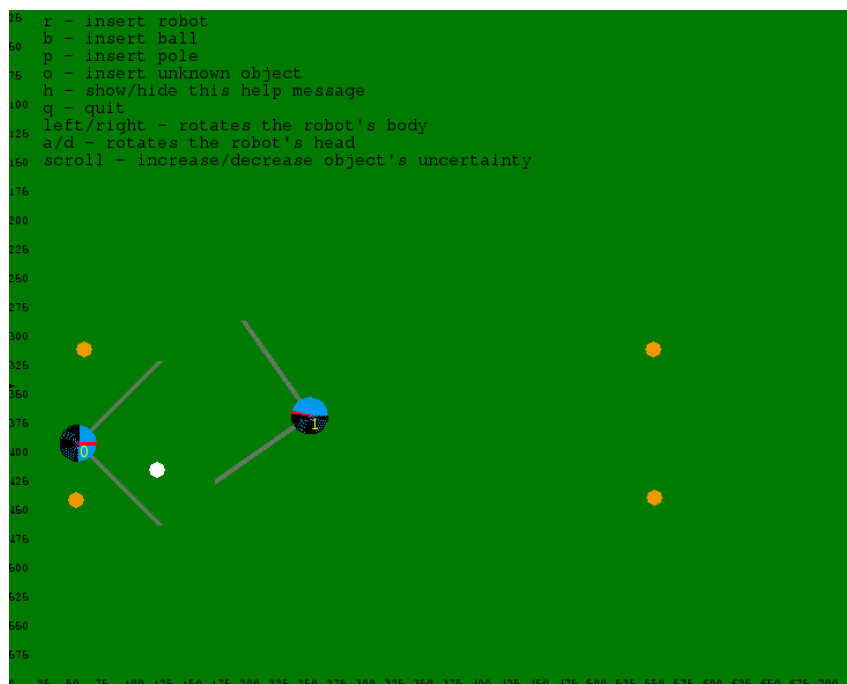
A partir disso, é gerado o arquivo que pode ser utilizado pelo simulador na geração de comportamentos. No entanto, é primordial saber o funcionamento do simulador para que seja possível testar e usufruir de uma melhor forma a contribuição que o trabalho traz. Para execução do simulador, deve-se abrir a pasta dele pelo terminal e utilizar o comando "python3 WorldSimulation.py". A Figura 6.7 apresenta a interface do simulador bem como alguns comandos para inserção de objetos no campo.

Figura 6.7 – Simulador



Um outro comando, esse fundamental para execução dos comportamentos criados é "python3 nomedoarquivogerado.py". Onde "nomedoarquivo" deve ser substituído pelo nome escolhido como saída do programa no NetBeans, pegando o exemplo anterior o comando é "python3 IA.py". A Figura 6.8 traz a interface do simulador, após a inserção de objetos no campo e da execução desse comando.

Figura 6.8 – Simulador executando comportamento



6.3 SUMÁRIO DO CAPÍTULO

Esse capítulo abordou um estudo de caso onde pode ser observado um pouco das possibilidades de jogadas que podem ser criadas utilizando essa linguagem. Ademais, foi feito um tutorial especificando os passos da transformação da linguagem Maia em Python e da execução dessa saída no simulador.

O próximo capítulo traz as considerações finais, ressaltando as principais contribuições dessa pesquisa, enfatizando algumas dificuldades encontradas, além de citar possíveis trabalhos futuros.

7 CONSIDERAÇÕES FINAIS E TRABALHOS FUTUROS

Este capítulo traz uma conclusão da pesquisa realizada e sobre a utilização da proposta da mesma. Além disso, também são descritas as principais limitações encontradas para o desenvolvimento do trabalho. Também são apresentados alguns possíveis trabalhos futuros que podem ser feito a partir desse.

As principais dificuldades encontradas foram para realização de testes tanto no robô real quanto no simulador, visto que não há uma quantidade muito grande de funcionalidades, isso se dá pelas limitações que o robô ainda tem. Limitações essas voltadas ao módulo da caminhada, onde não são feitos movimentos muito elaborados pelo robô.

No entanto foram criadas mais funcionalidades visando que em um futuro próximo as mesmas possam ser implantadas no robô. Ainda essas funcionalidades podem facilitar equipes que estejam fazendo pesquisa na área, talvez já possam utilizá-las.

Outro problema foi delimitar quais seriam os próximos passos, essas funcionalidades que deviam ser criadas, uma vez que há uma vasta quantidade de possibilidades. Porém, foram escolhidas as que ampliem a capacidade de criação de comportamentos levando em consideração as restrições que o robô possui.

Já como motivação para desenvolvimento desse trabalho se deu pela dificuldade na criação de comportamentos, visto que é necessário saber a linguagem que está sendo usada, e, como há diversas modelos de robôs, além de simuladores com as mais variadas linguagens, a criação dessa linguagem foi uma forma de tentar minimizar tal problema, aprendizado de várias linguagens.

Este trabalho também abordou uma Ontologia, cujo objetivo é facilitar a implementação de jogadas de futebol, visto que ela tem representações gráficas, o que facilita o aprendizado de seu domínio. Sendo assim, ela descreve a semântica e a lógica do cenário de futebol de robôs, classificando os objetos do campo de acordo com suas características.

Assim sendo ontologia e linguagem se complementam formando um papel importante na criação de comportamentos, visto que é possível mapear tal cenário e implementá-lo levando em consideração a dinamicidade do jogo. Logo, a equipe Taura Bots pode usufruir de tal trabalho, bem como as demais equipes.

Todavia, ainda há melhorias a serem feitas buscando aprimorar esse trabalho. Como um possível trabalho futuro pode ser a criação de mais funcionalidades para a linguagem bem como para a Ontologia. Um outro trabalho é reutilizar e adaptar a linguagem para utilização em outros ambientes, visto que a criação da mesma foi para resolução de um problema eminente.

REFERÊNCIAS BIBLIOGRÁFICAS

- ALLGEUER, P.; BEHNKE, S. Hierarchical and state-based architectures for robot behavior planning and control. In: **Proceedings of 8th Workshop on Humanoid Soccer Robots, IEEE-RAS Int. Conf. on Humanoid Robots, Atlanta, USA**. [S.l.: s.n.], 2013. p. 3–5.
- ALVES, J. B. M. **Controle de robô**. [S.l.]: Editora Cartgraf, 1988.
- ARMADA, M. et al. Arquitectura mecánica y modelos cinemáticos. **II Jornadas Iberoamericanas de Robótica**, 1998.
- ARNOLD, G. V. et al. Uso da linguagem rs em robótica. **Electrónica e Telecomunicações**, v. 3, n. 6, p. 501–508, 2002.
- BESTMANN, M. **Biologically Inspired Localization On Humanoid Soccer Playing Robots**. 2014. Tese (Doutorado), 2014.
- BONINI, J. A. **A Definição de uma Linguagem para a Programação do Comportamento de Robôs Dentro do Contexto da RoboCup**. 2015. Trabalho de Conclusão de Curso, UFSM, Rio Grande do Sul, Brasil, 2015.
- BORST, W. N. **Construction of engineering ontologies for knowledge sharing and reuse**. [S.l.]: Universiteit Twente, 1997.
- GANAPATHI, G.; LOURDUSAMY, R.; RAJARAM, V. Towards ontology development for teaching programming language. v. 3, 07 2011.
- GROOVER, M. P. **Robótica: tecnologia e programação**. [S.l.]: McGraw-Hill, 1989.
- GRUBER, T. R. Toward principles for the design of ontologies used for knowledge sharing? **International journal of human-computer studies**, Elsevier, v. 43, n. 5, p. 907–928, 1995.
- GRUNINGER, M. Designing and evaluating generic ontologies. In: **Proceedings of the 12th European Conference of Artificial Intelligence**. [S.l.: s.n.], 1996. v. 1, p. 53–64.
- GUIMARÃES, F. J. Z. Utilização de ontologias no domínio b2c. **Mestrado em Informática, Pontifícia Universidade Católica do Rio de Janeiro**, 2002.
- HALFPAP, D. M. et al. Um modelo de consciência para aplicação em artefatos inteligentes. Florianópolis, SC, 2005.
- HURST, W. J.; MORTIMER, J. W. **Laboratory Robotics-A Guide to Planning, Programming, and Applications**. 1987.
- JÚNIOR, O. d. R. N. et al. Desenvolvimento da fluência tecnológica em programa educacional de robótica pedagógica. Florianópolis, SC, 2011.
- KITANO, H. et al. Robocup: A challenge problem for ai and robotics. In: **RoboCup-97: Robot Soccer World Cup I**. [S.l.]: Springer, 1998. p. 1–19.
- LIBRELOTTO, G. R. Um compilador para a linguagem rs distribuída. 2001.

LIBRELOTTO, G. R. et al. A semântica formal da linguagem rs. **XII Simpósio Brasileiro de Linguagens de Programação**, p. 216–225, 2008.

LOPES, J. L. et al. Uma abordagem baseada em ontologias para sensibilidade ao contexto na computação pervasiva. In: **Artigo do WPUC 2007–I Workshop on Pervasive and Ubiquitous Computing**. [S.l.: s.n.], 2007.

MARTINS, A. **O que é Robótica**. [S.l.]: Brasiliense, 1993.

MONTENEGRO, F. J. C. **Criação de um Simulador para Auxiliar no Desenvolvimento de Estratégias de Comportamento para Futebol de Robôs**. 2015. Trabalho de Conclusão de Curso, UFSM, Rio Grande do Sul, Brasil, 2015.

OTTONI, A. L. C. **Introdução à Robótica**. [S.l.]: Olimpíada de Robótica do Campo de Vertentes, 2010.

PAZOS, F. Automação de sistemas e robótica. **Editora Axcel Books do Brasil**, 2002.

PIERRAKEAS, C.; SOLOMOU, G.; KAMEAS, A. **An Ontology-Based Approach in Learning Programming Languages**. 2012. 393-398 p.

ROBOCUP. **O que se pesquisa na RoboCup?** Disponível em <<http://robocup.org.br/pesquisa.php>>. Acesso em 10-07-2017. Disponível em: <<http://robocup.org.br/objetivo.php>>.

ROBOCUP, B. **Objetivo da RoboCup Federation**. Disponível em <<http://robocup.org.br/objetivo.php>>. Acesso em 10-07-2017. Disponível em: <<http://robocup.org.br/objetivo.php>>.

ROMANO, V.; DUTRA, M. Introdução a robótica industrial. **Robótica Industrial: Aplicação na Indústria de Manufatura e de Processo**, São Paulo: **Edgard Blücher**, p. 1–19, 2002.

ROSHEIM, M. E. In the footsteps of leonardo [articulated anthropomorphic robot]. **IEEE Robotics & Automation Magazine**, IEEE, v. 4, n. 2, p. 12–14, 1997.

SCHWARZ, M. et al. Nimbro-op humanoid teensize open platform. In: **In Proceedings of 7th Workshop on Humanoid Soccer Robots, IEEE-RAS International Conference on Humanoid Robots, Osaka**. [S.l.: s.n.], 2012.

SOWA, J. F. Building, sharing, and merging ontologies. **web site: <http://www.jfsowa.com/ontology/ontoshar.htm>**, 2001.

TOPALIDOU-KYNIASOPOULOU, A.; SPANOUDAKIS, N. I.; LAGOUDAKIS, M. G. A case tool for robot behavior development. In: **RoboCup 2012: Robot Soccer World Cup XVI**. [S.l.]: Springer, 2013. p. 225–236.

TSAL, L.-W. **Robot analysis: the mechanics of serial and parallel manipulators**. [S.l.]: John Wiley & Sons, 1999.

WANG, C.; CAO, L.; CHI, C.-H. Formalization and verification of group behavior interactions. **IEEE Transactions on Systems, Man, and Cybernetics: Systems**, IEEE, v. 45, n. 8, p. 1109–1124, 2015.

WOLF, D. F. et al. Robótica móvel inteligente: Da simulação às aplicações no mundo real. In: **Mini-Curso: Jornada de Atualização em Informática (JAI), Congresso da SBC**. [S.l.: s.n.], 2009. p. 13.

ANEXO A – GRAMÁTICA DA LINGUAGEM MAIA

```
/*
* Felipe Maia
* Orientador: Giovani Librelotto
*/

grammar diss;

@header{
    import java.util.ArrayList;
}

@members{
    ArrayList<String> states = new ArrayList<>();
    ArrayList<String> input = new ArrayList<>();
    ArrayList<String> output = new ArrayList<>();
    ArrayList<String> var = new ArrayList<>();
    ArrayList<String> bool = new ArrayList<>();
    ArrayList<String> flt = new ArrayList<>();
    ArrayList<String> inteiro = new ArrayList<>();
    ArrayList<String> in = new ArrayList<>();
    int cont=0;
}

expr: name body;

/* module x: */
name: 'module' ID DP;

/* { in out var states initially rules */
body : AC input output var stts init rules* FC;

input : 'in' DP OC parametersIn pInput;

/* ball */
parametersIn : 'ball' | POLE | (ROBOT_0 | ROBOT_P | ROBOT) | 'Xline' |
               'Tline' | 'Lline' | 'line' | 'unknown' | 'time' | 'scoreboard'
               ;
```

```

/* (r, theta) */
pInput: AP a=ID ',,' b=ID FP listIn
    {if(!var.contains($a.text)){
        var.add($a.text); in.add($a.text);}
    else if(!in.contains($a.text))
        System.out.println("[Linha "+$a.getLine()+" Coluna "+$a.pos+"]
        INPUT: Variável '"+$a.text+"' já declarada.");
    if(!var.contains($b.text)){
        var.add($b.text); in.add($b.text);}
    else if(!in.contains($b.text))
        System.out.println("[Linha "+$b.getLine()+" Coluna "+$b.pos+"]
        INPUT: Variável '"+$b.text+"' já declarada.");
    flt.add($a.text); flt.add($b.text);
}
;

listIn :      ',,' parametersIn pInput
            |      CC PV
            ;

/* out: [parameters] */
output : 'out' DP OC parametersOut CC PV;

/* walk(r, theta, phi) | defend(r, theta, phi) | kick(r, theta, phi) */
parametersOut : 'walk' AP a=ID COMMA b=ID COMMA c=ID FP COMMA
    'defend' AP a=ID FP COMMA
    'kick' AP a=ID COMMA b=ID FP COMMA
    'passS' AP a=ID COMMA b=ID FP COMMA
    'passM' AP a=ID COMMA b=ID FP COMMA
    'passL' AP a=ID COMMA b=ID FP COMMA
    'walkBall' AP a=ID COMMA b=ID FP COMMA
    'walkLeft' AP a=ID COMMA b=ID FP COMMA
    'walkRight' AP a=ID COMMA b=ID FP COMMA
    'walkTurnBall' AP a=ID COMMA b=ID FP COMMA
    'rotateAround' AP a=ID COMMA b=ID FP
    {if(!var.contains($a.text))
        var.add($a.text);
    else

```

```

        System.out.println("[Linha "+$a.getLine()+" Coluna "+$a.pos+"]
        OUTPUT: Variável '"+$a.text+"' já declarada.");
    if(!var.contains($b.text))
        var.add($b.text);
    else
        System.out.println("[Linha "+$b.getLine()+" Coluna "+$b.pos+"]
        OUTPUT: Variável '"+$b.text+"' já declarada.");
    if(!var.contains($c.text))
        var.add($c.text);
    else
        System.out.println("[Linha "+$c.getLine()+" Coluna "+$c.pos+"]
        OUTPUT: Variável '"+$c.text+"' já declarada.");
    flt.add($a.text); flt.add($b.text); flt.add($c.text);});

/* var: [type id...] */
var :   'var' DP OC CC PV
      |   'var' DP OC decl listVar
      ;

listVar :   COMMA decl listVar
          |   CC PV
          ;

/* float, bool and int */
decl: 'float' ID
      {flt.add($ID.text);
      if(!input.contains($ID.text) && !output.contains($ID.text) &&
      !states.contains($ID.text) && !var.contains($ID.text))
          var.add($ID.text);
      else
          System.out.println("[Linha "+$ID.getLine()+" Coluna
          "+$ID.pos+"] ATR: Variável '"+$ID.text+"' já declarada.");
      }
      | 'boolean' ID
      {bool.add($ID.text);
      if(!input.contains($ID.text) && !output.contains($ID.text) &&
      !states.contains($ID.text) && !var.contains($ID.text))
          var.add($ID.text);
      else

```

```

        System.out.println("[Linha "+$ID.getLine()+" Coluna
        "+$ID.pos+"] ATR: Variável '"+$ID.text+"' já declarada.");
    }
    | 'int' ID
    {inteiro.add($ID.text);
    if(!input.contains($ID.text) && !output.contains($ID.text) &&
        !states.contains($ID.text) && !var.contains($ID.text))
        var.add($ID.text);
    else
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        ATR: Variável '"+$ID.text+"' já declarada.");
    }
;

/* states: [parameters]*/
stts : 'states' DP OC parametersStts CC PV;

/* walk, defend, kick */
parametersStts : 'attack' COMMA 'defense' COMMA 'goalkeeper' COMMA unknown
;

/* defined state by programmer */
unknown : ID
    {if(!var.contains($ID.text))
        states.add($ID.text);
    else
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        STATE: Já existe uma variável com mesmo nome '"+$ID.text+"'.");
    }
;

/* initially: [state]*/
init: 'initially' DP OC parametersInit CC PV;

/* variable in state */
parametersInit: 'attack'
    | 'defense'
    | 'goalkeeper'
    | ID

```

```

    {if(!states.contains($ID.text))
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        INITIALLY: Variável '"+$ID.text+"' não declarada em 'states'.");
    }
;

/* rules: condition -> action */
rules:  condition action;

/* one input object*/
condition: parametersIn AP a=ID COMMA b=ID FP and HAS parametersInit RARROW
    {if(!var.contains($a.text)){
        var.add($a.text); in.add($a.text);}
    else if(!in.contains($a.text))
        System.out.println("[Linha "+$a.getLine()+" Coluna "+$a.pos+"]
        INPUT: Variável '"+$a.text+"' já declarada.");
    if(!var.contains($b.text)){
        var.add($b.text); in.add($b.text);}
    else if(!in.contains($b.text))
        System.out.println("[Linha "+$b.getLine()+" Coluna "+$b.pos+"]
        INPUT: Variável '"+$b.text+"' já declarada.");
    flt.add($a.text); flt.add($b.text);
    }
;

/* more than one input object*/
and:    AND parametersIn AP a=ID COMMA b=ID FP and
    {if(!var.contains($a.text)){
        var.add($a.text); in.add($a.text);}
    else if(!in.contains($a.text))
        System.out.println("[Linha "+$a.getLine()+" Coluna "+$a.pos+"]
        INPUT: Variável '"+$a.text+"' já declarada.");
    if(!var.contains($b.text)){
        var.add($b.text); in.add($b.text);}
    else if(!in.contains($b.text))
        System.out.println("[Linha "+$b.getLine()+" Coluna "+$b.pos+"]
        INPUT: Variável '"+$b.text+"' já declarada.");
    flt.add($a.text); flt.add($b.text);
    }
;

```

```

|
;

/* If | If Else | Atr | Up */
action: emit term
      | atr term
      | condIf term
      | condIfElse term
      | up term;

/* if condition { term } */
condIf: 'if' AP (oprLflt | oprLbool) FP AC term FC;

/* if condition { term } ... */
condIfElse: condIf condElse;

/* else {term} */
condElse: 'else' AC term FC;

/* If | If Else | Atr */
term: emit term
     | atr term
     | condIf term
     | condIfElse term
     |
     ;

/* emit(walk); or emit(defend); or emit(kick); */
emit:  'emit' AP 'walk' AP parameter COMMA parameter COMMA parameter FP FP PV
      | 'emit' AP 'defend' AP (NUM|DIG) FP FP PV
      | 'emit' AP 'kick' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'passL' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'passM' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'passS' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'walkBall' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'walkLeft' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'walkRight' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'walkTurnBall' AP parameter COMMA parameter FP FP PV
      | 'emit' AP 'rotateAround' AP parameter COMMA parameter FP FP PV

```

```

;

parameter:  (NUM|DIG)
|  ID
{if(!var.contains($ID.text))
    System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
    EMIT: Variável '"+$ID.text+"' não declarada em 'var'.");
}
;

/* up(attack); or up(defend); or up(goalkeeper);
or up(estado definido pelo usuario); */
up :      'up' AP parametersInit FP PV;

/* assignment */
atr:      ID DPI (oprA | member) PV
{if(!var.contains($ID.text))
    System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
    ATR: Variável '"+$ID.text+"' não declarada em 'var'.");
if(!flt.contains($ID.text))
    System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
    ATR: Erro de tipo.");
}
|      ID DPI bool PV
{if(!var.contains($ID.text))
    System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
    ATR: Variável '"+$ID.text+"' não declarada em 'var'.");
if(!bool.contains($ID.text))
    System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
    ATR: Erro de tipo.");
}
;

/* booleano */
bool:      'true'
|      'false'
;

/* condition if */

```



```

oprLflt:    (oprA | member) operatorL (oprA | member)
;

/* arithmetic operations */
oprA :    member operatorA listaA
        ;

listaA :    member
            |    member operatorA listaA
        ;

member: ID
    {if(!var.contains($ID.text))
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        ATR: Variável '"+$ID.text+"' não declarada em 'var'.");
    if(!flt.contains($ID.text))
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        ATR: Erro de tipo.");
    }
    | (NUM | DIG);

/* + or - or * or / */
operatorA:  ADD
            |  SUB
            |  MTP
            |  DIV
            ;

operatorL:  LT // <
            |  GT // >
            |  LEQ // <=
            |  GEQ // >=
            |  EE // ==
            |  DIF // !=
            ;

/* condition if */
oprLbool: ID EE bool
    {if(!var.contains($ID.text))

```

```

        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        ATR: Variável '"+$ID.text+"' não declarada em 'var'.");
    if(!bool.contains($ID.text))
        System.out.println("[Linha "+$ID.getLine()+" Coluna "+$ID.pos+"]
        ATR: Erro de tipo.");
    }
;

POLE: 'pole'[1-4];
ROBOT_0: 'robot'[1-3] '_o';
ROBOT_P: 'robot'[1-3] '_p';
ROBOT: 'robot'[1-3];
ID: [_a-zA-Z][_a-zA-Z0-9]*;
DIG: [0-9]+;
COMMA: ',';
SUB: '-';
NUM: ('-')?[0-9]+'.'[0-9]+;
DP: ':';
AC: '{';
FC: '}';
OC: '[';
CC: ']';
AP: '(';
FP: ')';
DPI: ':=';
ADD: '+';

MTP: '*';
DIV: '/';
LT: '<';
GT: '>';
LEQ: '<=';
GEQ: '>=';
EE: '==';
DIF: '!=';
HAS: '#';
RARROW: '->';
AND: '&';
PV: ';';

```

```
WSPC : [ \r\t\n]+ -> skip;
```