Descrição do Time de Futebol de Robôs da Equipe ARARABOTS Azul - 2022 - Categoria IEEE Very Small Size Soccer

Juan Carlos C. de Lima Sales[1], Allan Menchik da Cunha[2], Ana Karolina da Silva Barbosa[3], Mizuki Katsukawa[4], Amanda Ottoni Petini[5], Natália Silva Duarte[6], Fábio Huang[7], Vinicios Faustino de Carvalho[8], Edson Takashi Matsubara[9]

***Abstract*— A categoria IEEE Very Small Size Soccer (VSSS) é atualmente uma das categorias mais disputadas na Competição Brasileira de Robótica e consiste em um jogo de futebol de robôs de três robôs por time, controlados de maneira autônoma. Este documento tem por objetivo descrever o sistema de controle do time de futebol de robôs ARARABOTS Azul, desenvolvido no LIA-UFMS (Laboratório de Inteligência Artificial da Universidade Federal de Mato Grosso do Sul). O sistema é composto por: visão manipulada pela biblioteca OpenCV, comunicação via Bluetooth, implementação da estratégia de controle utilizando ROS, navegação utilizando `univector` para direcionamento dos robôs e *hardware* constituído de componentes de fácil acesso, além da simulação para treinamento de estratégias. Este artigo tem por objetivo apresentar uma recapitulação do funcionamento do time VSSS da ARARABOTS e apresentar novos elementos do mesmo.**

I. INTRODUÇÃO

A categoria de futebol de robôs *IEEE Very Small Size Soccer* propicia aos alunos um ambiente único de desafios e desenvolvimentos técnico e interpessoal. No desenvolvimento técnico, os alunos colocam em prática os conhecimentos adquiridos em disciplinas como Inteligência Artificial, Programação e Estrutura de Dados, Geometria Analítica, Engenharia de Software, Laboratório de Hardware, Computação Gráfica entre outras. No desenvolvimento pessoal, os alunos trabalham a autoconfiança, raciocínio lógico, resolução de conflitos e principalmente o trabalho em grupo. Os benefícios são inúmeros aos participantes da competição.

Apoio: Universidade Federal de Mato Grosso do Sul.

[1,2,3,4,5]Alunos de graduação da Universidade Federal de Mato Grosso do Sul (Laboratório de Inteligência Artificial, Faculdade de Computação). Campus Universitário da UFMS, S/N, Bairro Universitário, Caixa Postal: 549. CEP: 79070-900. Campo Grande – Mato Grosso do Sul – Brasil. {juan_carlos[1], allan.m[2], ana.k[3], mizuki.katsukawa[4], amanda.petini[5], natalia.duarte[6], fabio_huang[7]}@aluno.ufms.br

[8]Aluno de mestrado ciência da computação da Universidade Federal de Mato Grosso do Sul (Laboratório de Inteligência Artificial, Faculdade de Computação). Campus Universitário da UFMS, S/N, Bairro Universitário, Caixa Postal: 549. CEP: 79070-900. Campo Grande – Mato Grosso do Sul – Brasil. [8]vinicios.carvalho@aluno.ufms.br

[9]Professor da Universidade Federal de Mato Grosso do Sul (Laboratório de Inteligência Artificial, Faculdade de Computação). Campus Universitário da UFMS, S/N, Bairro Universitário, Caixa Postal: 549. CEP: 79070-900. Campo Grande – Mato Grosso do Sul - Brasil. [9]edsontm@facom.ufms.br

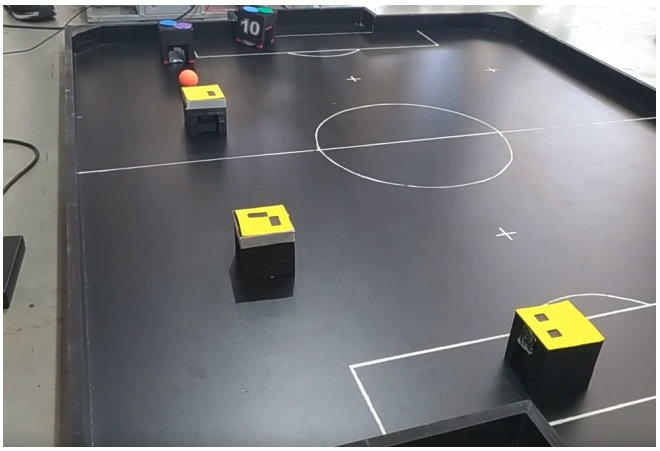


Fig. 1: Jogo da ARARABOTS durante a LARC 2018

Atualmente o sistema da ARARABOTS é distribuído nos módulos: visão, estratégia, interface, treinador e comunicação. Todos esses módulos executam de maneira paralela e a comunicação entre eles é feita através do *Robot Operating System* (ROS). Embora o processamento distribuído insira complexidade no desenvolvimento, ele torna possível o compartilhamento de um mesmo recurso entre dois times, isso é útil na simulação de jogos pois o mesmo sistema de visão e comunicação pode ser utilizado por duas instâncias de times diferentes.

A principal entrada para o sistema que controla os robôs durante a partida é o sistema de visão. Ele é constituído por uma câmera posicionada na parte superior do campo, que possibilita a visão de todos os jogadores e da bola, e um sistema de localização. Em virtude do período de quarentena e competições simuladas, o sistema também funciona a partir das informações de posição de tanto os robôs quanto a bola obtidas do simulador *FiraSim*.

Tendo informações de posicionamento dos robôs, cada time implementa a sua estratégia que gera comandos que definem os comportamentos de seus robôs. Com os comandos definidos pela estratégia, os sistemas de transmissão enviam os comandos de controle para cada robô. Em um jogo feito através de um simulador, tais comandos devem ser dirigidos para o mesmo.

Enfim, nas seções seguintes serão feitas descrições dos principais componentes do sistema implementados pelos ARARABOTS. Também será descrito como o sistema integra o simulador necessário para a competição virtual de 2021.

II. Visão computacional

A visão computacional utiliza como entrada a imagem gerada pela câmera que é posicionada acima da arena. A câmera utilizada é do tipo *webcam*, marca ELP, modelo FHD01M-SFV. A resolução máxima da câmera é 1920x1080 *pixels* a uma taxa de 30 quadros por segundo. Visando uma taxa de atualização maior, a câmera foi configurada para operar em uma resolução inferior, 1280x720 *pixels*, porém à 60 quadros por segundo.

Devido à inclinação da câmera e a distorção radial da lente, é preciso realizar correção para que a detecção dos objetos não tenha erros de posição. Para isso, são capturadas várias imagens de um padrão quadriculado, como um tabuleiro de xadrez. A variabilidade do padrão nas imagens (inclinação, rotação, distancia para a câmera) é muito importante para a qualidade da correção. Essas imagens são utilizadas para gerar uma matriz de correção, que serve de parâmetro para uma função da biblioteca de visão computacional OpenCV, que elimina distorção inerente à lente da câmera. Para corrigir o erro perspectiva na imagem, são definidos 4 pontos que indicam os vértices da arena, sendo assim feita a correção de perspectiva. Desse modo, a imagem final é livre de distorções significativas, o que permite um melhor mapeamento dos objetos.

Além disso, para reduzir o número de *pixels* a serem verificados e evitar que objetos fora da arena interfiram na visão, há um módulo que permite fazer um recorte da imagem original de tal maneira que apenas o conteúdo da arena estará visível para o próximo passo no processamento de imagem.

A imagem que ao final do processo de correção de distorções tem como espaço de cores o RGB (*Red, Green e Blue*) é convertida para o espaço HSV (*Hue, Saturation e Value*). Após essa conversão de espaço de cores é feito o processo de segmentação, que irá classificar os *pixels* entre as cores utilizadas. Para que a segmentação seja o mais fidedigna possível, é preciso realizar a calibração do espectro, feita através da interface do sistema onde é possível escolher e, com o clique do mouse, definir e identificar na imagem quais *pixels* pertencem àquela cor, selecionando o melhor intervalo de valores para sua representação.

Com a imagem separada em segmentos nas cores de interesse, é feita a identificação de cada um dos objetos. Para identificação da bola, é detectada a posição do contorno encontrado no segmento de cor laranja, visto que esse é o único objeto dessa cor dentro da arena.

Para identificar os robôs inimigos, é utilizado o algoritmo de agrupamento k-médias e o número de centroides passados para o algoritmo é o número de inimigos em campo. Desse modo, o algoritmo retorna a posição dos centroides como sendo a posição de cada um dos inimigos. O modelo de estampa da identificação do time adversário pouco influencia no resultado, visto que somente a cor do time é levada em consideração.

Já para a identificação dos robôs do time é usada uma estampa específica, composta por dois círculos, um cuja cor identifica o time ao qual o robô pertence e outro cuja cor identifica unicamente o robô. A partir da disposição dos círculos é extraído o vetor de orientação do robô.

Um exemplo de *tags* pode ser observado na Figura 2.

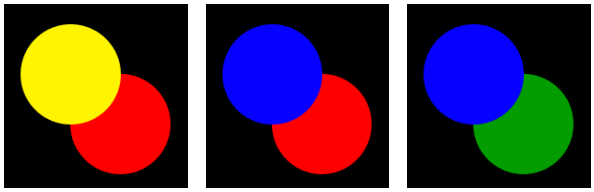


Fig. 2: Exemplos de *tags*

III. Simulador

No âmbito da pandemia de Covid-19, novas atitudes foram necessárias para manter o distanciamento e a segurança da população. Competições de robótica passaram a usar simuladores para a realização dos jogos e eventos e, neste contexto, *FiraSim* passou a ser o principal simulador utilizado. Ele foi desenvolvido com o objetivo de simular o futebol de robôs da categoria IEEE VSSS. Os principais componentes do seu funcionamento são: *Qt5* [1], *OpenGL* [2], *Open Dynamics Engine* (ODE) [3] e *Google Protobuf* [4]. *Qt* consiste em um *framework* de desenvolvimento focado na criação de aplicativos e interfaces de usuário para plataformas *desktop*, embarcadas e móveis. *OpenGL* é o principal ambiente para o desenvolvimento de aplicações gráficas 2D e 3D. ODE é uma biblioteca de código aberto de alto desempenho voltada para a simulação da dinâmica de corpos rígidos e detecção de colisões entre os mesmos. Finalmente, o *Google Protobuf* define um protocolo de comunicação extensível e de linguagem neutra para serializar dados estruturados.

O simulador tem como responsabilidade fornecer as mesmas informações de posicionamento de objetos que o sistema de visão baseado em câmeras do time AraraBots. Desta forma, o simulador integra o sistema como sendo uma ”Visão Simulada”, tomando o lugar da visão baseada em câmeras e visão computacional.

Atualmente, mesmo com a competição sendo realizada na modalidade presencial, o simulador continua sendo utilizado com o intuito de simplificar implementações de melhorias no sistema e testar as mesmas de maneira controlada, em um ambiente ideal. Após serem validadas no ambiente simulado, as implementações são passadas para o ambiente físico.

IV. Arquitetura de Comunicação

A conexão entre os módulos do sistema é feita pelo ROS. O pseudo-sistema operacional possuí diversas ferramentas e bibliotecas para auxiliar desenvolvedores na criação de aplicações de automação e robótica. O ROS fornece gerenciamento de pacotes, abstração de hardware, bibliotecas com

[1] https://doc.qt.io/qt-5/index.html
[2] https://www.opengl.org/
[3] https://www.ode.org/
[4] https://developers.google.com/protocol-buffers

diversos algoritmos utilizados nas áreas de robótica e inteligência artificial, simuladores, mecanismos de comunicação, processamento inerentemente distribuído e *scripts*. Organizada em nós, pacotes, tópicos e serviços, a comunicação entre os módulos é facilmente implementada por meio de interfaces disponíveis. Um módulo (nó) pode se comunicar com outro de forma direta através de um serviço, ou de forma indireta pela publicação em um tópico, que vai gerenciar as informações recebidas e as informações a serem enviadas. Ao enviar a mensagem para o nó alvo, o tópico realiza o processo de assinatura (Figura 3).

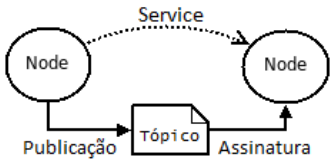


Fig. 3: Esquema de publicação e assinatura.

Com isso, é possível realizar a comunicação entre os diversos módulos do software utilizando o esquema para controle, gerenciamento e sincronização de informações entre os módulos. O caminho principal de processamento é denotado pela Figura 4, que indica da esquerda para a direta o fluxo de atualizações feitas até que se chegue no destinatário final, o robô físico.

A. Nós de processamento

1) Visão: O módulo de visão realiza a captura e processamento da imagem utilizando algoritmos de correção de distorção, corte de áreas de interesse e separação de cores. Após o tratamento e identificação de objetos, o módulo publica mensagens contendo as informações dos objetos no tópico de informações. Ao utilizar a "Visão Simulada", as informações de posicionamento são obtidas do *FiraSim* e retransmitidas para o tópico.

2) Robô: O módulo Robô é responsável por abstrair tudo que um robô é capaz de fazer, por exemplo, seguir um ponto, ir para um ponto, olhar para um ponto, chutar a bola, criando assim um encapsulamento das tarefas relativas ao robô. Após o cálculo de tais tarefas, os nós publicam as mensagens a serem enviadas para o robô físico por meio do tópico de comunicação.

3) Treinador: É a camada superior de controle, que é responsável por determinar qual estratégia deve ser usada, utilizando informações de várias fontes, como por exemplo, o juiz. Essa camada também é responsável por alterar a estratégia dinamicamente durante a partida, caso a estratégia atual não esteja obtendo bons resultados.

4) Interface: Como os nós de robôs e treinador, o nó da interface também lê as mensagens publicadas no tópico de informações para desenhar o campo virtual, que é a representação da situação atual de jogo, na tela. O nó da interface é responsável por alterar o estado em que o jogo se encontra, trocar parâmetros dos nós de robôs, além de realizar operações na visão por meio de serviços.

5) Mensageiro: O nó mensageiro é responsável por criar e monitorar os *sockets* que se comunicam com o meio físico, escutar o tópico da comunicação aguardando novas mensagens de controle. Em caso de falha, o nó terá um número limitado de tentativas para entregar o pacote e, caso não consiga, tentará a reconexão.

V. Estratégia

A parte estratégica pode ser dividida superficialmente em três módulos, chamados de Treinador, àrvores de Comportamento (*Behaviour Trees*) e Resolução de Caminho respectivamente.

A. Treinador

Assim como em um time de futebol convencional, cada um dos robôs têm um papel específico em uma partida. No entanto, as interações entre os robôs em campo, considerando seus papéis, não se dá de maneira orgânica como entre jogadores humanos. Supondo, por exemplo, uma situação em que o robô zagueiro encontra-se mais avançado no campo que os atacantes aliados e mais próximo à bola. Neste cenário, seria conveniente que este robô assumisse o papel de atacante e algum de seus companheiros assumisse seu papel com zagueiro.

Neste sentido, introduziu-se uma entidade, independente dos robôs, denominada treinador. Dentro do sistema, o treinador assume o papel de administrar o time de robôs, assim como um treinador em uma equipe esportiva.

Dentre suas atribuições, além da supracitada capacidade de alterar os papéis dos robôs, o treinador é capaz instanciar e destruir nós dos robôs no ROS. Assim sendo, este determina os robôs que participarão de uma partida.

B. Behaviour Trees

Para coordenar o comportamento dos robôs e a estratégia de jogo usamos um modelo matemático chamado *Behaviour Trees* (no português Árvore de Comportamento). Esse modelo consegue descrever alternância entre um conjunto finito de tarefas armazenadas em nós.

A execução de uma árvore começa pela sua raiz e desce os filhos começando pela esquerda. Dessa forma, somente nós folhas resultam diretamente na produção de um comportamento ou checagem de uma condição enquanto nós interiores são usados para direcionar o fluxo de processamento para seus filhos usando selectors (seletores) e sequences (sequenciadores). Selectors executam cada um de seus filhos até algum ter sucesso enquanto os sequences executam até um falhar. Ademais, alternativamente os dois também param quando não possuem mais filhos. Um exemplo de *Behaviour Tree* utilizada pelo AraraBots é representado na Figura 5.

C. Resolução de Caminho

Para o planejamento de rota optou-se pela utilização do algoritmo de navegação para ambientes dinâmicos `univector` [1]. Tal algoritmo consiste na geração de um campo vetorial em determinado ambiente, considerando cada objeto presente no local. A finalidade do campo é conduzir

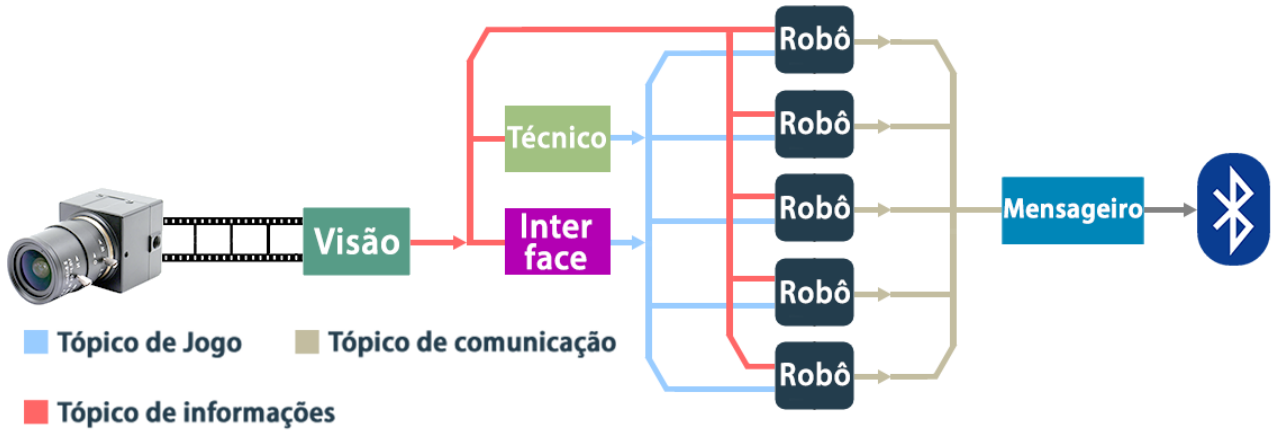


Fig. 4: Fluxo de controle do sistema.

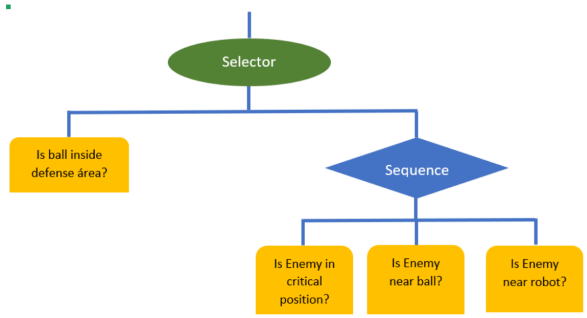


Fig. 5: Exemplo de *behaviour tree*. O seletor representado possui dois filhos: um nó que verifica se a bola está dentro da área do goleiro e uma sequência de nós, executados caso a verificação do nó anterior retorne falso.

um móvel à um ponto de interesse, através de um caminho desobstruído. Para isto, os vetores próximos aos obstáculos divergem radialmente do centro dos mesmos, repelindo o móvel, de forma a evitar possíveis colisões. Os demais vetores convergem para o ponto de interesse, atraindo o móvel.

Uma das vantagens do emprego do `univector` é sua adaptação à alterações no meio. Isto é, o algoritmo considera as posições do móvel e dos obstáculos. Desta forma, quando os objetos no ambiente alteram suas posições, o campo é recalculado com base nas posições atuais, evitando possíveis colisões que pudessem ser causadas devido à posições virtuais discrepantes das reais.

Além disso, o `univector` também baseia-se nos vetores velocidade dos elementos, sendo assim capaz de predizer posicionamentos futuros. Com isso, o móvel desvia-se não da posição em que o obstáculo encontra-se, mas da posição que o mesmo assumirá com base em seu deslocamento, como mostra a Figura 6.

Por fim, considerando o fato de que o algoritmo conduzirá o robô até a bola, há casos em que a bola não está alinhada com o gol adversário, como pode ser visto na Figura 7a, de forma que quando o robô atinge a bola, esta não se moverá

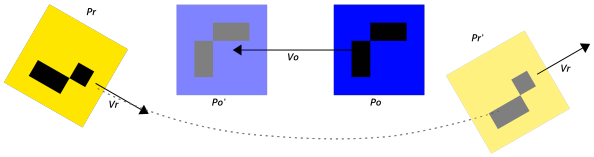


Fig. 6: Desvio do obstáculo predito

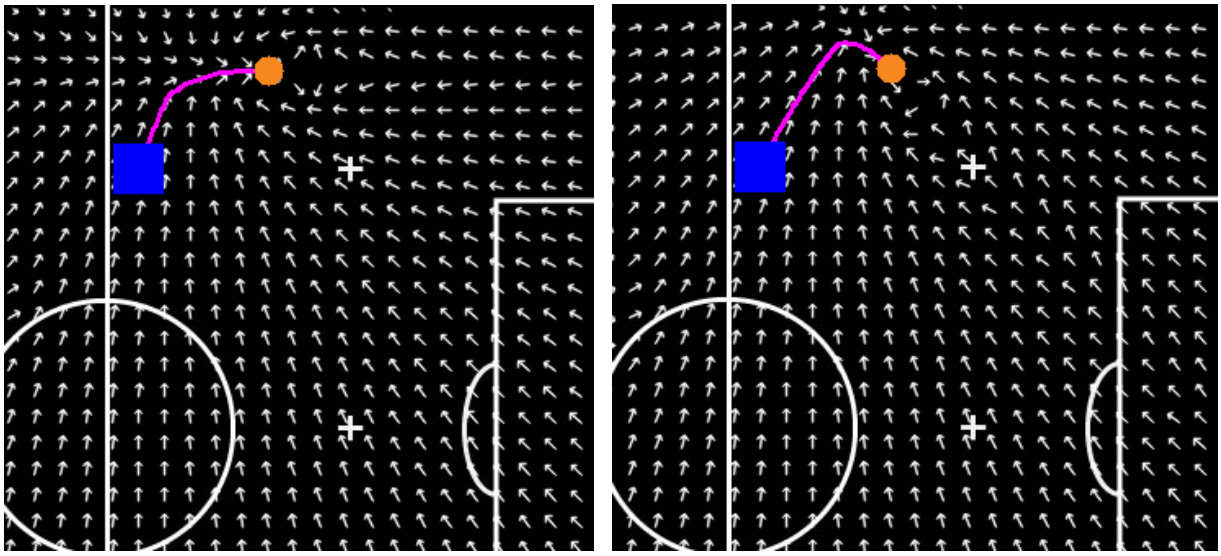
até o interior do gol. Para solucionar este problema, utilizou-se matrizes de rotação para rotacionar os eixos do plano em torno da origem, alinhando-se o eixo X do campo com o eixo formado pela direção **bola - gol adversário**, de maneira que o robô atinja a bola com uma angulação que a desloque ao centro do gol adversário, à direita na Fig. 7a e Fig. 7b.

VI. *Hardware*

O desenvolvimento do *hardware* foi orientado ao baixo custo e à disponibilidade de componentes no mercado.
A estrutura do robô é construída em impressão 3D utilizando acrilonitrilo-butadieno-estireno (ABS) como material. Todas as peças do robô com exceção dos motores são impressas. A utilização de tecnologia 3D permite diminuir o custo da estrutura e também torna a construção mais simples, pois não depende de materiais/peças que muitas vezes são difíceis de encontrar no mercado.

A. *Estrutura*

A estrutura principal é constituída de duas partes que são unidas por parafusos para formarem o bloco principal do robô. A escolha de dividir a estrutura principal em duas partes foi motivada pela dificuldade de imprimir peças altas, devido ao fenômeno de retração do ABS, além de uma melhor qualidade de impressão pois menos suportes precisaram ser utilizados. As peças B e C da Figura 8 correspondem às duas metades do bloco principal.



(a) Campo não rotacionado (b) Campo rotacionado

Fig. 7: Representação do campo não rotacionado e do campo rotacionado

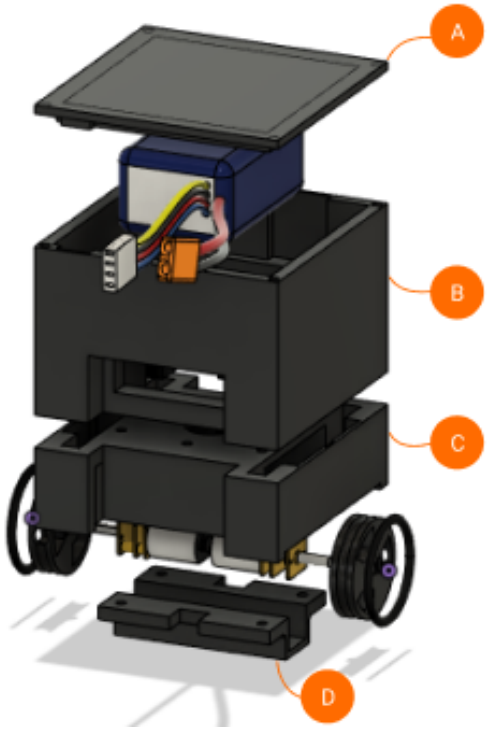


Fig. 8: Vista explodida do robô.

Na parte inferior (peça C) são fixados os motores com o auxílio do suporte D. Escolheu-se modelar um único suporte para os dois motores, ao contrário de um suporte para cada um, para garantir que os eixos dos dois motores fiquem ao longo de uma mesma linha.
Na peça superior (peça B), está presente o compartimento da bateria assim como o suporte da placa controladora.

A tampa do robô (peça A) é fixada na parte superior da estrutura com ímãs de neodímio de $3mm$, e também possui duas saliências que se encaixam de forma única em dois sulcos presentes na estrutura.

O corpo central das rodas também foi impresso em 3D e como pneu utilizou-se anéis de borracha para cano PVC. Vale notar que o *design* permite que o eixo do motor seja fixado utilizando-se porca e parafuso de *2mm*, essa característica foi inspirada pelo modelo da equipe Projeto Neon.

B. Motores

O motor utilizado encontra-se na Figura 9, escolheu-se a opção *high-power* por possuir tensão nominal relativamente baixa, em torno de $6V$. Esse aspecto é importante pois reguladores de tensão e baterias que atendam essa faixa de tensão são mais disponíveis no mercado.

Fig. 9: Micromotor *high-power* com redução 30:1

C. Placa controladora

A placa controladora do robô foi reformulada em 2018, em relação aos anos anteriores. O microcontrolador foi alterado, assim como o circuito integrado de ponte H dos motores e também o circuito de potência dos motores.
Inseriu-se um circuito divisor de tensão para junto com o

conversor analógico-digital (ADC) monitorar a tensão da bateria, para a mesma não ser utilizada quando a tensão for abaixo da tensão de corte. A notificação é feita através de um *buzzer* presente na placa. Como incremento para o *hardware*, foi adicionado suporte para giroscópio em 2019.
Também foi inserido um diodo de proteção para impedir a circulação de corrente quando a bateria é conectada com a polarização invertida.

1) Microcontrolador: Utilizou-se o microcontrolador ESP32 da marca Espressif, montado no módulo ESP-WROOM-32. Este chip foi escolhido por conta de seu alto custo-benefício, pois apresenta em um único componente dois núcleos de processamento de 32 bits de até $240MHz$ e conexão *bluetooth*.
Dessa forma foi possível substituir o microcontrolador ATmega328-PU e o módulo *bluetooth* HC-05, anteriormente utilizados, por um único componente. Dessa forma foi possível diminuir o custo e também o espaço ocupado na placa.

2) Ponte H e Circuito de Potência: Usou-se o *driver* duplo TB6612FN, este circuito integrado (CI) possui duas pontes H, dessa forma é possível ligar os dois motores do robô no mesmo. Ele é capaz de alimentar dois canais com corrente contínua de $1A$ em cada um, com picos de até 3A. Além disso o circuito desde CI é baseado em tecnologia MOSFET e portanto muito mais eficiente que CIs baseados em transistores de junção bipolar (BJTs).
Para alimentar os motores optou-se por utilizar o circuito regulador de tensão *step down* DC-DC MP1584EN para abaixar a tensão da bateria para a tensão nominal dos motores. Diferentemente dos reguladores de tensão linear (família 78xx, por exemplo) este circuito é mais eficiente e portanto, menos energia é perdida no processo de conversão de tensões.

D. Bateria

Escolheu-se bateria LiPo de alta descarga de 3 células com carga de $800mAh$. Esse modelo de bateria permite jogar mais que um jogo sem a necessidade de recarregar ou trocar a bateria do robô. Esse fator também é muito importante no processo de desenvolvimento de estratégias, pois é possível desenvolver durante longos períodos sem interrupção.

VII. Conclusão e Trabalhos Futuros

Desde a última competição, o sistema foi alterado, para a integração dos ambientes físico e simulado, de maneira que fosse possível alternar com facilidade entre ambos. Com isso, é possível traçar estratégias em um ambiente ideal e identificar as limitações não relacionadas ao software.

Foram identificados problemas que demandam tempo para serem solucionados, não podendo ser resolvidos antes da competição. Uma das limitações do sistema é dada pela visão computacional física. Dessa forma, estuda-se a possibilidade de refatoração, ou mesmo migração para sistemas unificados de visão, utilizados por outras equipes. Além disso, pretende-se revisitar a estratégia de controle, dado que este também é um ponto crítico do sistema. Atualmente, os robôs tem um giroscópio em seu circuito, com o qual é possível ajustar a trajetória com mais precisão que a obtida através da visão. Essa funcionalidade, no entanto, ainda não foi implementada.

Agradecimentos

Agradecemos ao Laboratório de Inteligência Artificial da Faculdade de Computação da Universidade Federal de Mato Grosso do Sul, por prover o espaço e os recursos necessários para o desenvolvimento de nossas pesquisas e atividades.

References

[1] Y. Lim, S.-H. Choi, J.-H. Kim, and D.-H. Kim, "Evolutionary univector field-based navigation with collision avoidance for mobile robot," *IFAC Proceedings Volumes*, vol. 41, no. 2, pp. 12787–12792, 2008.

[2] "Opencv - open source computer vision.," 2000. Acesso em 15/06/2019.

[3] M. Quigley, K. Conley, B. Gerkey, J. Faust, T. Foote, J. Leibs, R. Wheeler, and A. Y. Ng, "Ros: an open-source robot operating system," in *ICRA workshop on open source software*, vol. 3, p. 5, Kobe, Japan, 2009.

[4] F. C. Vieira, A. A. Medeiros, P. J. Alsina, and A. P. Araújo Jr, "Position and orientation control of a two-wheeled differentially driven nonholonomic mobile robot.," in *ICINCO (2)*, pp. 256–262, 2004.

[5] R. Munoz-Salinas and S. Garrido-Jurado, "Aruco library," *URL: http://sourceforge. net/projects/aruco*, 2013.