RobôCIn VSSS Description Paper

Carlos Henrique Caloete Pena[1], Elisson Rodrigo da Silva Araújo[1], José Douglas Pontes Silva[1], Juliana do Nascimento Damurie da Silva[1], Lucas Dias Maciel[1], Lucas Henrique Cavalcanti Santos[1], Lucas Oliveira Maggi[1], Maria Isabel Fernandes dos Santos[1], Mariana da Silva Barros[1], Mateus Gonçalves Machado[1], Matheus Viana Coelho Albuquerque[1], Morgana Beatriz Feijó Galamba[1], Pedro Nogueira Coutinho[1], Tiago da Silva Barros[1], Hansenclever de Franca Bassani[1], Edna Natividade da Silva Barros[1]

***Resumo*—Este *Team Description Paper* (TDP) tem como objetivo descrever o projeto desenvolvido pela equipe RobôCIn, do Centro de Informática da UFPE, para participação na competição IEEE *Very Small Size Soccer*. Neste TDP são descritos os principais sistemas desenvolvidos: o sistema de localização por visão computacional, a técnica de planejamento de caminho baseada em campos potenciais, as tomadas de decisão, bem como a comunicação entre os robôs, o sistema de controle, o sistemas eletrônico e mecânico do robô projetado, e as mudanças feitas no nosso sistema e no simulador utilizado para a competição.**

I. INTRODUÇÃO

Para participar da competição da IEEE na categoria *Very Small Size Soccer*, são necessários cinco pilares essenciais: detecção de objetos, planejamento de caminho e decisões, mecânica, eletrônica e comunicação. O primeiro estágio do ciclo de execução do projeto é a detecção de objetos. Nesta etapa, a equipe optou por criar um módulo de detecção da bola e dos robôs utilizando visão computacional. Após a etapa de detecção, as informações de posição, direção e velocidade são passadas para o módulo de estratégia. O módulo de estratégia define um caminho a ser seguido pelos robôs, e assim calcula as velocidades a serem enviadas para o módulo de controle do robô (*hardware* embutido no robô). Com as velocidades definidas, o módulo de controle local do robô realiza o controle do robô.

Na seção II, é explicado o módulo de localização dos objetos em campo. Na seção III, é detalhado o planejamento de caminho e ações realizadas pelo módulo de estratégia. Em seguida as seções IV e V descrevem a mecânica e eletrônica dos robôs criados pela equipe e a seção VI que explica a comunicação entre os módulos. Finalizando, a seção VII explica o simulador utilizado e as mudanças feitas pela equipe para integração do mesmo com o sistema do time. Em seguida, são apresentadas as seções de conclusão, agradecimentos e referências.

[1]Todos os autores estão no RobôCIn no Centro de Informática, Universidade Federal de Pernambuco, Brazil robocin@cin.ufpe.br

II. LOCALIZAÇÃO

Para identificação e detecção dos robôs e bola em campo, foi desenvolvido pela nossa equipe um *software* de visão computacional em C++, utilizando a biblioteca *open-source* OpenCV [1] e baseado na detecção de pares de cores, localizadas na parte superior de cada robô. Para captura das imagens necessárias para detecção, é utilizada uma câmera posicionada a 2 metros de altura sobre o campo. O fluxo de técnicas de visão computacional pode ser visto como um *software* de arquitetura de Dutos e Filtros [2] com as seguintes etapas: pré-processamento, segmentação, identificação de objetos, e transformação de coordenadas, descritas a seguir.

A. PRÉ-PROCESSAMENTO

Na etapa de pré-processamento, são configurados os parâmetros internos da câmera, como controle de brilho e ajuste de foco automático. A etapa de pré-processamento foi embutida no software principal do RobôCIn, que conta com comunicação direta com o drive da câmera para manipular as configurações que são definidas em nível de *hardware*. Além disso, estão sendo desenvolvidas pesquisas em métodos para controlar mudanças na iluminação, além da aplicação de filtros padrões que melhoram os dados para etapa de segmentação.

B. SEGMENTAÇÃO

Na etapa de segmentação, é feita a classificação de cada *pixel* da imagem, removendo ruídos para facilitar a identificação dos objetos. A segmentação de cores se baseia na análise das intensidades dos canais de cor de cada pixel, onde o pixel se configura como um *pixel* de interesse se a intensidade dos canais de cor do *pixel* está dentro de um dos intervalos da cor característica de alguma etiqueta ou bola. Calculamos previamente a classe para todos os valores possíveis de *pixels*, isto é, definimos a classe de cor para cada cor possível. Nós utilizamos uma *Look Up Table* (LUT) [4] para armazenar esses resultados, de tal maneira que indexamos os pixels a partir da soma dos valores dos canais (R,G,B) multiplicados por 65536, 256, e 1, respectivamente. Dessa forma, em tempo de execução é necessário somente realizar uma consulta à LUT para saber a classe de um *pixel*. A LUT aparece como uma

otimização da técnica de segmentação através de cores, tornando o processo de segmentação eficiente, realizando apenas operações de leitura e escrita durante a partida.

Para tornar a segmentação mais eficiente, os valores guardados na LUT são de intensidades no espaço RGB. Este é o espaço de cor no qual recebemos a imagem da câmera. Porém, o processo em si de segmentação precisa ser realizado em um espaço de cor que seja capaz de ser robusto a variações na iluminação e nas cores.

A geração da LUT possui as seguintes etapas:

- Normalização do espaço RGB.
- Filtragem de tons de cinza.
- Conversão para o espaço de cor HSV
- Segmentação do espaço de cor HSV usando pivôs

No processo de Normalização do espaço RGB é utilizada uma normalização vetorial ponderada, fortalecendo a cromaticidade das cores (isto é, se a cor for avermelhada, ela será normalizada para uma cor mais próxima do vermelho). De forma similar, isto ocorre para as outras cores primárias e secundárias, porém fazendo com que a região de ambiguidade tenha um distanciamento maior entre os seus elementos iniciais. Já a filtragem em tons de cinza é utilizada para remover falsos positivos das partes que compõem o campo, que geralmente são regiões muito claras (muito saturadas), ou muito escuras (trazem pouca precisão na cor), mas apresentam ainda algum nível de cromaticidade. Esta filtragem é feita a partir de uma região cilíndrica orientada na diagonal principal do espaço RGB. Após isso, convertemos todas as cores resultantes em cores no espaço HSV. Este espaço de cor distribui as cores por cromaticidade no canal "Hue"(matiz), onde iremos realizar a segmentação das cores pela sua matiz, sendo descartados os valores dos canais S (Saturation) e V (Value). Associamos cada cor à cor do pivô que está mais próximo dela. Cada pivô está posicionado de tal forma que as fronteiras de decisão coincidem com as fronteiras da percepção humana, resultando em uma segmentação mais robusta à variação da iluminação, e torna-se mais fácil de segmentar por ter menos parâmetros. Temos um número para cada pivô: o Hue da cor que ele representa. Temos 7 cores para segmentar, portanto 7 parâmetros apenas para as cores. Além disso, temos mais um parâmetro para o filtro de tons de cinza, que delimita o quanto queremos filtrar com o cilindro no espaço RGB. Assim, há uma totalização de 8 parâmetros, e com isso é demonstrado que precisamos de menos parâmetros para configurar a segmentação do que uma segmentação por regiões retangulares em 3D, por exemplo, no espaço RGB. Após a etapa de segmentação, a matriz com os elementos de interesse (Figura 1) será enviada para a etapa seguinte.

Foram também criadas ferramentas na interface do programa de forma a acelerar o tempo de calibração da segmentação e de detecção de erros na mesma. Tais ferramentas incluem a possibilidade de salvar e carregar configurações de segmentação automaticamente em tempo de execução; e a adição de novas visualizações de depuração da imagem da câmera.

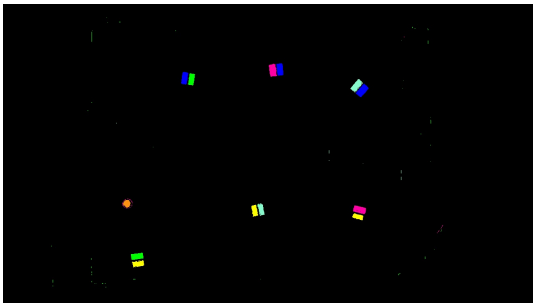

Figura 1. Elementos de interesse identificados.

C. IDENTIFICAÇÃO DE OBJETOS

De forma a otimizar o tempo de processamento, a imagem segmentada é comprimida através do algoritmo *run length encoding* [3], comprimindo *pixels* que sejam da mesma cor e estejam na mesma linha em uma única célula de informação, chamada de *run*, que contém a cor e a quantidade de *pixels* correspondentes. Essa compressão ajuda a otimizar o tempo para encontrar os centróides, pois possibilita utilizar o algoritmo *union-find* para juntar *runs* vizinhos da mesma cor, calculando o centróide de cada conjunto a cada junção nova. Por fim, o número de objetos será o número de conjuntos disjuntos formados cujo tamanho seja superior a um threshold.

A utilização da compressão *run length encoding*, em conjunto com o *union-find*, tem como principal vantagem a diminuição no número de iterações pela matriz segmentada completa. Além disso, ele permite a paralelização do processo de compressão, que pode ser fragmentado em um *run* para cada linha, e depois se fazer o *union-find* desses runs.

D. CONVERSÃO DE COORDENADAS

A última etapa de detecção de objetos é a conversão dos pontos adquiridos no plano da imagem para pontos em escala real, no sistema de coordenadas do campo. Para alcançar o resultado desejado, utilizamos métodos de interpolação de coordenadas. Esse método consiste em utilizar correspondências já conhecidas de pontos da imagem com os pontos reais para gerar uma transformada que execute a conversão, para entregar posições e velocidades no sistema de coordenadas desejado.

III. ESTRATÉGIA

O módulo de estratégia é responsável por determinar as melhores ações e caminhos a serem tomados pelos robôs, utilizando como fontes de informação a posição, orientação e velocidade dos robôs e da bola presentes em campo, informações essas que são fornecidas pelo módulo de visão. É importante que o módulo de estratégia seja rápido e preditivo, de forma a prover uma reação rápida dos robôs

em função das situações de jogo e assim garantir uma maior probabilidade de vitória.

O módulo de estratégia do RobôCIn pode ser separado em camadas, nomeadas: escolha de formação, escolha de ação, planejamento de caminho e controle de movimento. Tais camadas são executadas em ordem e têm como saída final as velocidades das duas rodas que devem ser enviadas para cada robô, de forma a executar o plano de jogo, como pode ser observado na Figura 2. As mesmas serão descritas nas subseções III-A, III-B, III-C e III-D, respectivamente.

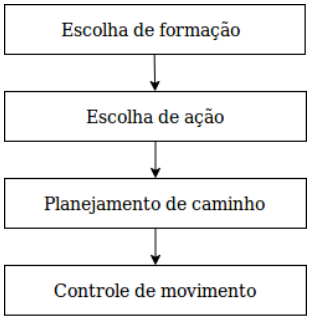

Figura 2. Fluxo das camadas do módulo de estratégia.

A. ESCOLHA DE FORMAÇÃO

Para explicar a camada de escolha de formação, é necessário primeiramente introduzir o conceito de comportamento. O comportamento de um robô é definido como o conjunto de relações entre as ações que o robô pode tomar e as situações em que cada ação é tomada. Sua definição é análoga à definição de função de um jogo de futebol. Por exemplo, semelhante a um goleiro de um time de futebol que, ao perceber que a bola está se aproximando de seu gol, deve ir em direção à bola com objetivo de desviar sua trajetória, um robô programado com o comportamento de goleiro deve executar ação semelhante em campo. O software do RobôCIn apresenta os seguintes comportamentos que são utilizados atualmente: Atacante, Defensor e Goleiro.

A camada de escolha de formação tem como função analisar as posições dos robôs e da bola no campo e assim decidir, para cada robô, qual o comportamento mais útil dada a condição atual de jogo. É importante observar que a escolha de formação é executada *frame* a *frame*, ou seja, um robô que começou o jogo sendo goleiro pode terminar sendo atacante, e vice-versa. Tal característica possibilita uma maior adaptabilidade ao time de robôs. Essa adaptabilidade pode ser observada na Figura 3. Nessa situação, o robô atacante, marcado pelo identificador 1, não está em boa posição para chegar na bola, pois está sendo marcado pelo robô inimigo de identificador 2. Já o robô defensor, marcado com o identificador 3, tem um caminho quase retilíneo em direção ao gol. Nessa situação, a troca de funções faz com que este defensor tome o comportamento de atacante e aproveite a chance para fazer o gol. Para decidir qual robô é o melhor para cada função, ordenamos os robôs de acordo com a sua distância para o ponto objetivo daquela função.

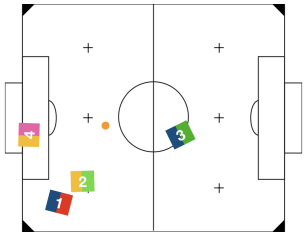

Figura 3. Situação de jogo onde a troca de funções entre os jogadores é vantajosa

B. ESCOLHA DE AÇÃO

A camada de escolha de ação, como é sugerido pelo nome, diz respeito à avaliação da melhor ação a ser tomada, dado o comportamento de cada robô. A escolha de ações é feita através de uma máquina de estados, onde os estados são ações e as arestas são situações de jogo. O exemplo de uma máquina de estados para um comportamento de atacante pode ser observada na Figura 4. Atualmente, no módulo de estratégia em questão, as seguintes ações podem ser tomadas: buscar a bola, conduzir a bola, girar e acompanhar a bola.

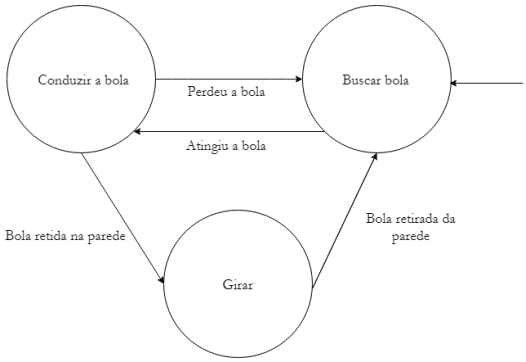

Figura 4. Máquina de estados que representa o comportamento do atacante

C. PLANEJAMENTO DE CAMINHO

A escolha da ação a ser executada implica na definição de uma coordenada e orientação definidas como objetivo, que deve ser alcançado o mais rápido possível para que a ação tenha sucesso. É trabalho da camada de planejamento de caminhos definir qual a melhor trajetória a ser seguida de forma a chegar no objetivo sem maiores problemas.

A abordagem usada pela equipe para o planejamento de caminho consiste em definir o caminho a ser seguido utilizando um campo de vetores unitários [10], que podem ditar a direção de movimento do robô. Duas espirais hiperbólicas são utilizadas para gerar os vetores, uma para fazer o robô se direcionar à bola e uma para fazer o robô

desviar de obstáculos como pode ser observado na figura 5.

O resultado do processamento do módulo de planejamento de caminho será um conjunto ordenado de pontos no plano que descrevem a trajetória a ser executada. A partir dessa trajetória, são gerados um objetivo e angulação instantâneos, que são pontos do caminho alcançáveis num intervalo de *frame*.

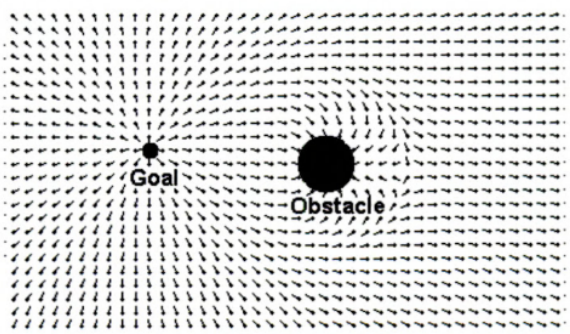

Figura 5. Univector Field [10]

D. CONTROLE DO MOVIMENTO

Dado o objetivo e angulação instantâneos gerados pelo planejamento de caminhos, é necessário mapear tais valores para velocidades de rodas que produzam o resultado desejado. Para tal, é utilizado um controle PD como mostrado nas equações 1, 2, 3 e 4, que mapeiam as velocidades necessárias para que o robô saia do ponto R para o ponto O, onde θ_{ref} é o ângulo desejado, θ é o ângulo atual do robô, ϵ_θ é o erro angular, D é a distância euclidiana entre o robô e o ponto objetivo e os valores K_p^θ, K_d^θ, K_p^v, K_d^v são constantes que dependem de fatores físicos do robô.

$$\epsilon_\theta[t] = \theta[t] - \theta_{ref}[t] \quad (1)$$

$$\omega = K_p^\theta{}_\theta[t] + K_d^\theta(\epsilon_\theta[t] - \epsilon_\theta[t-1]) \quad (2)$$

$$D[t] = \sqrt{(R_y[t] - O_y[t])^2 + (R_x[t] - O_x[t])^2} \quad (3)$$

$$v = K_p^v[t]\epsilon_v + K_d^v(D[t] - D[t-1]) \quad (4)$$

Para compensar o *delay* da câmera na tomada de decisão, implementamos um sistema de previsão que funciona usando o método de Runge-Kutta. Tal método se baseia em equações que levam em consideração o ângulo que o robô está se movimentando e sua velocidade para predizer onde o robô estará. A equação 5 calcula o ângulo futuro que o robô vai estar com base no seu ângulo atual e na sua velocidade angular. As equações 6 e 7 usam a média entre o ângulo atual e o futuro, a posição atual e a velocidade na coordenada para prever a posição futura dos objetos em campo.

$$\theta(t + \Delta t) = \theta t + \Delta t.wt \quad (5)$$

$$x(t + \Delta t) = x(t) + \Delta t.\cos((\theta.t + \theta(t + \Delta t))/2).V_x(t) \quad (6)$$

$$y(t + \Delta t) = y(t) + \Delta t.\cos((\theta.t + \theta(t + \Delta t))/2).V_y(t) \quad (7)$$

IV. MECÂNICA

O robô projetado esse ano foi planejado com o propósito de ser uma peça única e sólida, proporcionando ao robô maior estabilidade. Toda a estrutura para construção do robô foi realizada através de impressão 3D (Figura 6). A estrutura é composta dos seguintes módulos: chassi, tampa do motor, tampa das baterias, tampa chassi e capa. O módulo chassi tem como função dar suporte aos motores, às baterias e à placa principal. A tampa do motor segura o motor no chassi, a tampa das baterias tem a função de segurá-las, e a capa protege a placa do robô e é um suporte para as *tags*. Para projetar todas peças, utilizou-se o CAD 3D Autodesk Inventor [6], que permite realizar a modelagem e exportação de peças para impressão 3D.

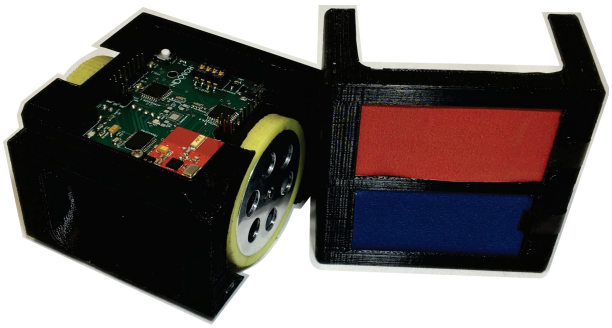

Figura 6. Robô desenvolvido pelo RobôCIn em impressão 3D.

Além da uma estrutura impressa em 3D, o robô da equipe RobôCIn possui dois micro motores com redução integrada de 50:1, e permitem 600 RPM com torque de 0,86 Kg-cm. Para completar a estrutura de locomoção do robô existem duas rodas com 50mm de diâmetro, localizadas nas laterais do robô, e 4 pivôs já inclusos na estrutura impressa em 3D.

V. ELETRÔNICA

Para controlar o robô são utilizados dois microcontroladores ATmega328, onde um atua como mestre e o outro como escravo. Eles foram escolhidos pela praticidade, tamanho, e capacidade de controlar todos os dispositivos requisitados no robô. Para controle de potência nos motores é utilizado o driver de motor TB6612FNG, que possui limite de corrente contínua de 1A por motor e 3A para corrente de pico.

O circuito do robô é composto também por um módulo *wireless*, responsável pela comunicação com o computador,

uma IMU (*Inertial Measurement Unit*), composta por um acelerômetro, magnetômetro e um giroscópio, a qual permite monitorar a orientação do robô, assim como sua velocidade de rotação, um Sensor de Mouse que retorna as posições em x e y do robô possibilitando uma descrição de trajetória do robô.

A fim de otimizar o espaço do robô, os circuitos foram impressos em placas de dupla camada. Para fazer os layouts dos circuitos impressos foi utilizado o *software* de desenvolvimento de placas de circuito impresso Eagle [7]. Nesta nova versão do robô, foram adicionados dois sensores óptico de mouse, com a função de rastrear o deslocamento do robô no campo, e auxiliar o controle de baixo nível. Os sensores de mouse necessitam estar perto do chão e por isso, desenvolvemos uma placa separada para comportar o sensor de mouse que se conecta a placa principal através de um conector, pois a placa principal fica na parte superior do robô. A placa principal foi desenvolvida conforme Figura 7 para compor a parte superior do robô e a placa do mouse foi desenvolvida conforme Figura 8.

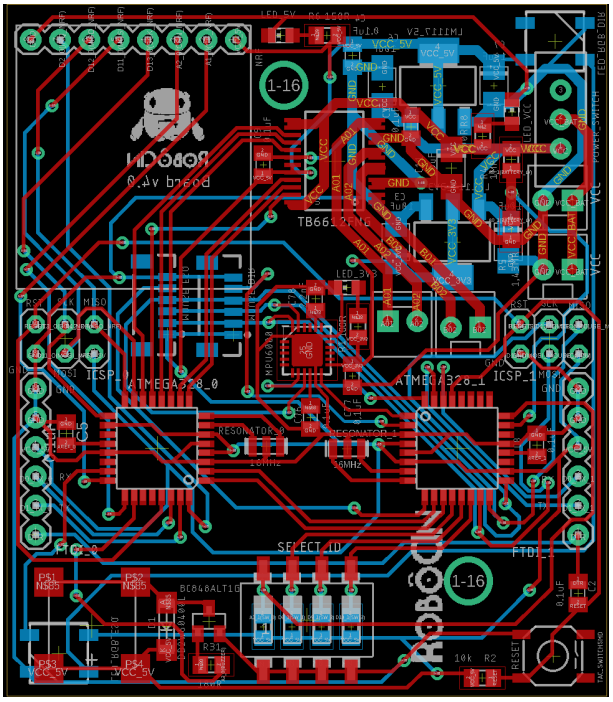

Figura 7. Projeto do circuito principal feito através do Eagle Circuit CAD.

VI. COMUNICAÇÃO

Para realizar a comunicação entre os robôs e o computador, na competição do *Very Small Size* em 2017 o circuito utilizado permitia o uso tanto do xBee Series 2 quanto o do módulo nRF24L01+. Depois de testar o desempenho do módulo nRF24L01+ nos jogos, julgou-se melhor optar pelo uso único dele e anular o uso do xBee Series 2 devido aos problemas dispostos pelo mesmo na competição *Very Small Size* de 2016. Assim, a partir da competição de

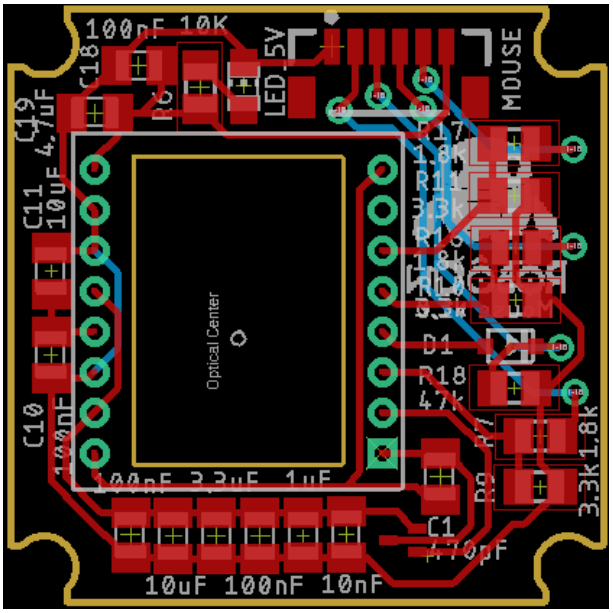

Figura 8. Projeto do circuito do mouse feito através do Eagle Circuit CAD.

2018, a equipe utiliza apenas o módulo nRF24L01+ para comunicação.

Ao decidir usar a comunicação do tipo radiofrequência (RF), a equipe implementou um protocolo através da classe *NRF24_Communication*, onde foram utilizadas as funções nativas do *RF24*. A comunicação foi desenvolvida de modo que incluísse não somente o envio de mensagens do computador para o robô, mas também a resposta do robô para o computador.

O módulo utilizado, nRF24L01+, se comunica com o microcontrolador seguindo o protocolo *Serial Peripheral Interface* (SPI). Assim, é utilizado um microprocessador ARM e dois módulos RF para a comunicação por parte do computador, onde um módulo atua como receptor, e o outro como transmissor. Contudo, no robô é utilizado um único RF que recebe os dados que o computador enviou e ao ser solicitado envia mensagens de resposta para o computador.

Para realizar a comunicação, é seguido um fluxo como está mostrado na Figura 9. Inicialmente, o computador processa a informação recebida através de imagens e converte nos movimentos que devem ser realizados pelos robôs. Para acessar o módulo nRF24l01+, o computador é conectado a uma "base station" através de uma interface serial ou Ethernet. Esta "base station" consiste de um sistema embarcado composto por uma placa ARM conectada por interface SPI a dois módulos nRF e que permite a sua comunicação com o computador. Em cada um dos robôs, um módulo nRF é utilizado para enviar ao robô as informações do computador sobre o movimento que deve realizar. Caso haja necessidade de resposta, o robô as envia para o computador também pelo seu módulo RF.

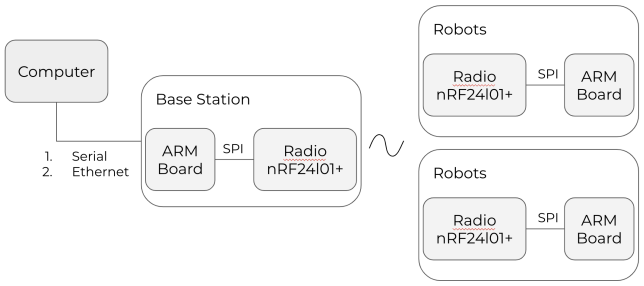

Figura 9. Arquiteura da comunicação entre computador e robôs.

VII. SIMULAÇÂO

Para desenvolver a estratégia da equipe e aprimorar o desempenho, é necessário de uso de simulação. O simulador optado pela equipe é o FIRASim [11], que foi desenvolvido pela equipe ParsianLabs e é disponibilizado com código aberto. O FIRASim foi baseado no GrSim, simulador que é utilizado pela categoria SSL (Small Size League). Ele utiliza a biblioteca ODE (Open Dynamics Engine) [13] para a simulação física e a biblioteca do OpenGL (Open Graphics Library) [12], para propósitos gráficos.

A equipe adaptou seu código para a comunicação com o simulador. Este usa o Google Protobuf como protocolo de comunicação. O software cria um socket UDP e troca mensagens de Protobuf com a simulação, recebendo delas as informações referentes ao ambiente e ao jogo e enviando para o simulador os comandos referentes às ações dos robôs. Para receber mais informações, como velocidade da bola e dos robôs e o número de gols, modificamos o protocolo de comunicação original.

Para aprimorar o desenvolvimento das estratégias e analisar melhor as situações de jogo, foi criado um árbitro interno no simulador. Assim, foi possível o reconhecimento dos eventos de gol, faltas, tiro de meta e pênalti e o reposicionamento dos robôs adequado para cada situação. Também melhorou-se a simulação física do ambiente com a alteração de algumas funções, de modo a oferecer uma opção com maior velocidade à simulação. Além disso, houve a necessidade da sincronização com o software de estratégia e do ajuste dos parâmetros de execução para de garantir uma simulação acelerada, atingindo uma velocidade de 15 vezes a velocidade original.

VIII. CONCLUSÃO

No presente *Team Description Paper* (TDP), é mostrado como foi projetado e desenvolvido o sistema necessário para participar na categoria *Very Small Soccer League*. Para isso, foi descrito um sistema de localização e detecção de objetos com objetivo de localizar a bola e os robôs em campo, um planejador de caminhos e decisor de comportamentos a fim de permitir a elaboração de estratégias de jogo, o projeto mecânico e eletrônico dos robôs e a comunicação entre eles e o sistema de controle. Também foi descrito o simulador utilizado para testes e desenvolvimento, e as mudanças realizadas no mesmo para comunicação com o *software* desenvolvido.

AGRADECIMENTOS

A equipe gostaria de agradecer o Centro de Informática da UFPE pelo apoio financeiro e de recursos durante todo o processo do projeto. Também gostaríamos de agradecer à todo apoio dado pelos professores Edna Barros e Hansenclever Bassani.

Referências

[1] OpenCV. Disponível em: http://opencv.org/. Acessado por último em 20 de junho de 2016.
[2] Sommerville, Ian. Engenharia de Software, 9ª edição. Pearson Education.
[3] Wen, Z., Shi, J., He, B., Chen, J., Ramamohanarao, K., & Li, Q. (2019). Exploiting GPUs for Efficient Gradient Boosting Decision Tree Training. IEEE Transactions on Parallel and Distributed Systems.
[4] Discrete YUV Look-Up Tables for Fast Colour Segmentation for Robotic Applications
[5] Hartigan, J., & Wong, M. (1979). Algorithm AS 136: A K-Means Clustering Algorithm. Journal of the Royal Statistical Society. Series C (Applied Statistics), 28(1), 100-108. doi:1. Retrieved from http://www.jstor.org/stable/2346830
[6] CAD 3D Autodesk Inventor. Disponível em: https://www.autodesk.com.br/products/inventor/overview. Acessado por último em 16 de junho de 2019.
[7] CAD Eagle. Disponível em: http://www.cadsoftusa.com/. Acessado por último em 20 de junho de 2016.
[8] Seesink, R. A. (2003). Artificial Intelligence in multi-agent robot soccer domain (Doctoral dissertation, Masters Thesis, University of Twente).
[9] Kim, Jong-Hwan, et al. Soccer robotics. Vol. 11. Springer Science & Business Media, 2004.
[10] Jong-Hwan Kim, Dong-Han Kim, Yong-Jae, 2004.
[11] Simulador FIRASim. Disponível em: https://github.com/fira-simurosot/FIRASim. Acessado por último em 22 de setembro de 2020.
[12] Woo, Mason and Neider, Jackie and Davis, Tom and Shreiner, Dave. OpenGL programming guide: the official guide to learning OpenGL, version 1.2. Addison-Wesley Longman Publishing Co., Inc., 1999.
[13] Open Dynamics Engine (ODE). Disponível em: https://www.ode.org/. Acessado por último em 22 de setembro de 2020.