Rafael Fernandes Gonçalves da Silva

Sistema de Identificação da Postura de Robôs na Categoria Very Small Size Utilizando Visão Computacional

Viçosa - MG, Brasil

2017

Rafael Fernandes Gonçalves da Silva

Sistema de Identificação da Postura de Robôs na Categoria Very Small Size Utilizando Visão Computacional

Universidade Federal de Viçosa

Departamento de Engenharia Elétrica

Engenharia Elétrica

Alexandre Santos Brandão

Viçosa - MG, Brasil

2017

Rafael Fernandes Gonçalves da Silva

Sistema de Identificação da Postura de Robôs na Categoria Very Small Size Utilizando Visão Computacional

Trabalho aprovado. Viçosa - MG, Brasil, 14 de julho de 2017:

Prof. Dr. Alexandre Santos Brandão
Orientador
Universidade Federal de Viçosa

Prof. Dr. Marcos Henrique Fonseca Ribeiro
Universidade Federal de Viçosa

Prof. M.Sc. Igor Henrique Beloti Pizetta
Instituto Federal do Espírito Santo

Viçosa - MG, Brasil
2017

Este trabalho é dedicado aos meus pais Rodrigo e Vilma.
Sem eles não teria chegado onde hoje estou...

Agradecimentos

Gostaria de agradecer primeiramente aos meus pais Rodrigo e Vilma e à minha irmã Débora, que estiveram sempre presentes em minha vida me oferecendo todo tipo de apoio.

Agradeço também à Deus pela sabedoria e pelas oportunidades concedidas a mim para alcançar esse objetivo.

Ao meu orientador Alexandre pela iniciação científica, pelos projetos e por todo conhecimento que obtive com ele.

À toda Equipe BDP, que foi a motivação principal na criação deste trabalho.

Por fim, agradeço aos colegas e amigos, que, mesmo de forma indireta, me ajudaram na realização deste trabalho.

"Aqueles que se sentem satisfeitos
sentam-se e nada fazem.
Os insatisfeitos são os
únicos benfeitores do mundo."
(Walter Savage Landor)

Resumo

O objetivo desse trabalho é descrever um programa de computador criado para identificar as posturas de robôs, com aplicações na categoria *Very Small Size* (VSS) de futebol de robôs. Nessa categoria, o sistema de visão é feito por uma câmera com a configuração de visão de topo (*bird view eye*) e os robôs são controlados exclusivamente por computadores. A BDP (*believe do and play*) é uma equipe da Universidade Federal de Viçosa dedicada à área de robótica e participa de campeonatos nacionais e internacionais nessa área. O setor de Processamento Digital de Imagens (PDI) da equipe é responsável por coletar dados a câmera e identificar a postura dos objetos em campo. Para isso, durante os últimos anos, foi desenvolvido um programa capaz de realizar tal identificação, utilizando ferramentas como calibração de cores, calibração da posição do campo, segmentação de cores, filtragem de imagens, dentre outras. O presente trabalho irá descrever, de uma forma detalhada, o programa criado. O resultado esperado é a identificação da posição e da orientação dos objetos em campo de uma forma contínua, com uma precisão consideravelmente alta em um intervalo de tempo relativamente pequeno, de modo a ser aplicado na categoria VSS. Ao final, os objetivos do programa são avaliados e sua aplicação na categoria é aceita.

Palavras-chaves: Visão Computacional; Segmentação de Cores; Postura de Robôs; Futebol de Robôs.

Lista de ilustrações

Lista de abreviaturas e siglas

UFV	Universidade Federal de Viçosa
BDP	*believe do'n play*
VSS	*Very Small Size*
SSL	*Small Size League*
PDI	Processamento Digital de Imagens
LARC	*Latin American Robotics Competition*
IronCup	*Inatel Robotics National Cup*
UDP	*User Datagram Protocol*

Sumário

1 Introdução

1.1 Categoria *Very Small Size* de Futebol de Robôs

A *Very Small Size* (VSS) é uma importante categoria de futebol de robôs presente em vários eventos de robótica em todo o mundo. Nesta categoria, duas equipes, cada uma com três robôs, disputam uma partida de futebol.

Os robôs devem ser menores que 75 x 75 x 75mm e devem possuir uma região visível em sua parte superior, onde serão colocadas uma ou duas cores, diferentes de preto, para identificação dos mesmos. A bola utilizada é laranja com 42,7mm de diâmetro e 46g de massa. O campo deve ser de cor preta e fosca, medindo 1,5 x 1,3m.

O sistema de visão é feito através de câmeras situadas a uma altura igual ou superior a dois metros. Através das cores observadas pela câmera, o computador deve identificar as posições dos robôs e da bola. Para diferenciar os times, são utilizadas as cores azul e amarelo, as quais são escolhidas antes do início das partidas.

Os robôs devem possuir uma etiqueta com tamanho mínimo pré definido, a fim de definir a cor dos times. Uma outra etiqueta, de cor diferente das cores citadas, pode ser usada para identificar individualmente os robôs de cada time, sendo de escolha livre. Os robôs devem ser totalmente independentes, autônomos, sendo vetada qualquer intervenção humana e controlados através de comunicação sem fio. O controle é feito somente por computadores, utilizando os dados de posição e orientação obtidos pelo processamento da imagem da câmera.

A Figura 1 demonstra como se configura uma partida.

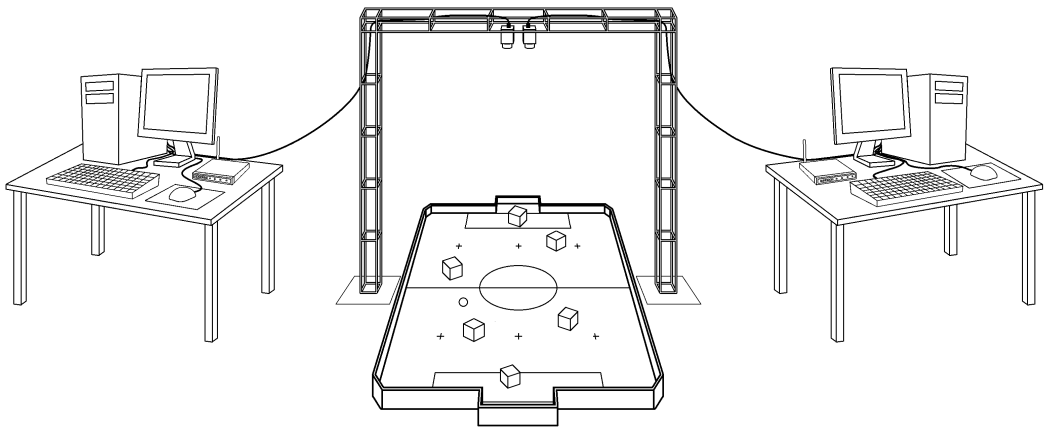

Figura 1 – Exemplo de uma partida em andamento (LARC/CBR, 2009).

1.2 Histórico da Equipe BDP

A BDP (*believe, do and play*) é uma equipe da Universidade Federal de Viçosa (UFV) e se dedica à área de futebol de robôs. A equipe foi fundada em 2005 e desde então tem se dedicado a participar de diversos eventos relacionados à robótica. Seus integrantes são graduandos dos cursos de Engenharia Elétrica, Engenharia Mecânica e Ciência da Computação. Através dos projetos da equipe, os alunos adquirem um bom conhecimento e contribuem para o desenvolvimento de pesquisas na Universidade.

De acordo com Junior, Lima e Brandao (2015), inicialmente, a BDP participava da categoria *Very Small Size* (VSS) de futebol de robôs. Em 2010, mudou de categoria, dedicando-se à *Small Size League* (SSL), também de futebol de robôs. Em 2015 seus integrantes decidiram retornar à categoria VSS e, nos anos de 2016 e 2017, se dedicou à mesma.

No ano de 2016, vários novos integrantes ingressaram na equipe, a qual acabou sendo dividida em quatro setores:

- Processamento Digital de Imagens (PDI), responsável por determinar a postura dos robôs em campo através das imagens capturadas pela câmera;
- Inteligência, responsável por decidir as ações de cada robô através dos dados recebidos do PDI;
- Eletrônica, encarregado de transformar os dados recebidos do setor da Inteligência em sinais de acionamento dos motores; e,
- Mecânica, dedicada à montagem e funcionamento das peças do robô.

Ao longo do ano, a equipe se dedicou a montar três robôs funcionais e criar um programa em MATLAB® capaz de controlá-los.

Em 2016, a equipe BDP/UFV participou da LARC (*Latin American Robotics Competition*), que ocorreu nos dias 8 a 12 de outubro na cidade de Recife-PE. A equipe alcançou as quartas de finais, obtendo o sétimo lugar dentre as 32 equipes inscritas.

No final de 2016, foram feitas algumas modificações no programa criado e nos robôs, a fim de melhorar o desempenho do time para a próxima competição.

No início de 2017, a equipe participou de mais uma competição, a IronCup (*Inatel Robotics National Cup*), que ocorreu nos dias 6, 7 e 8 de março na cidade de Santa Rita do Sapucaí-MG. Dessa vez, a equipe obteve o quinto lugar dentre as oito equipes melhores posicionadas na LARC de 2016.

Em 2017, outros novos integrantes ingressaram na equipe. Nesse ano, novas modificações nos robôs e no programa serão realizadas, sempre buscando conquistar melhores resultados nas competições futuras.

1.3 O Setor de PDI

O setor de Processamento Digital de Imagens da equipe é responsável por coletar imagens de uma câmera localizada no topo do campo e, em tempo real, identificar a postura dos objetos no mesmo. Para isso, foram utilizados procedimentos de calibração da posição do campo, calibração e segmentação de cores, filtragem de imagens e reconhecimentos de regiões em imagens binárias.

Vários foram os problemas enfrentados pelos integrantes do setor durante o desenvolvimento do programa como, a escolha do melhor sistema de cores, a realização do processo de segmentação das cores e a distorção da imagem da câmera, os quais estão explicados nas subseções a seguir.

1.3.1 Sistema de Cores

Frequentemente, uma imagem digital capturada de uma câmera está no sistema de cores RGB. Esse sistema, além das dimensões que representam a altura e a largura da imagem, possui uma terceira dimensão que contém três camadas com os valores correspondentes de cada cor primária (vermelho, verde e azul).

Um problema do sistema RGB é que este não se mostra tão eficaz na visualização da diferença entre as cores, pois as camadas de uma imagem mostram apenas a quantidade de cada cor primária presente em cada pixel da imagem. Isso causa uma grande dificuldade na percepção das cores, já que ao alterar uma das camadas produz uma cor resultante um tanto inconsistente, por exemplo, reduzir a quantidade de azul da cor branca produz amarelo. (SOLOMON; BRECKON, 2011)

De um modo alternativo, para facilitar a visualização das cores, são utilizadas representações, como, por exemplo, sistema o HSV. Assim como o RGB, o sistema HSV é dividido em três camadas. Em sua primeira camada está representado o espectro das cores, chamado de matiz (SOLOMON; BRECKON, 2011). Os seus valores são dados em graus (entre 0° e 360°) ou de forma normalizada (entre 0 e 1). É possível observar na Figura 2 que somente a matiz já é capaz de determinar de uma forma nítida a cor trabalhada.

Além da matiz, o sistema HSV possui valores de saturação e luminosidade da imagem, formando um cone. Com isso, torna-se ainda mais fácil visualizar cores mais puras (com alta saturação) e mais claras (com alta luminosidade). (SOLOMON; BRECKON, 2011)

Figura 2 – Representação em graus da matiz.

Conforme demonstrado em RapidTables (2015), a conversão dos valores em RGB para HSV pode ser dada pelas Equações (1.1), (1.2) e (1.3):

$$H = \begin{cases} 0° & \text{se } \Delta = 0 \\ 60°.\left(\frac{G-B}{\Delta}\ mod 6\right) & \text{se } C_{max} = R \\ 60°.\left(\frac{B-R}{\Delta} + 2\right) & \text{se } C_{max} = G \\ 60°.\left(\frac{R-G}{\Delta} + 4\right) & \text{se } C_{max} = B \end{cases} \tag{1.1}$$

$$S = \begin{cases} 0 & \text{se } C_{max} = 0 \\ \frac{\Delta}{C_{max}} & \text{se } C_{max} \neq 0 \end{cases} \tag{1.2}$$

$$V = C_{max} \tag{1.3}$$

onde R, G e B são os valores da terceira camada da imagem em RGB, C_{max} e C_{min} são, respectivamente, o maior e o menor valor dentre as três camadas da imagem em RGB e $\Delta = C_{max} - C_{min}$.

1.3.2 Segmentação de Cores

O processo de segmentação de cores envolve a utilização da diferença entre as cores de um objeto e sua vizinhança de forma a identificá-lo. Conforme Gonzalez, Woods e Eddins (2009), um método muito simples de segmentação é o uso de limiares, que consiste em gerar uma imagem binária, separando os pixels com valores maiores que o limiar dos valores menores que o limiar.

Em forma de equação, pode-se obter:

$$g(x,y) = \begin{cases} 0 & \text{se } f(x,y) < T \\ 1 & \text{se } f(x,y) \geq T \end{cases} \tag{1.4}$$

onde T indica o limiar de separação.

De uma forma mais complexa, para obter apenas as cores contidas em um intervalo de uma camada, torna-se necessário utilizar dois limiares, onde a resposta binária será 1 para valores entre os limiares e 0, caso contrário.

$$g(x,y) = \begin{cases} 0 & \text{se } f(x,y) < T_1 \\ 1 & \text{se } T_1 \leq f(x,y) \leq T_2 \\ 0 & \text{se } f(x,y) > T_2 \end{cases} \tag{1.5}$$

Um dos métodos mais simples utilizado para identificar os limiares de uma segmentação é por tentativa e erro, onde a qualidade do resultado é determinada pelo observador. Uma forma de deixar a segmentação mais interativa é permitir que o usuário altere os limiares enquanto observa a resposta binária, assim ele pode chegar à resposta desejada mais rapidamente (GONZALEZ; WOODS; EDDINS, 2009).

Em uma imagem colorida com três camadas, para obter uma resposta mais precisa, são necessários seis limiares. No caso de imagens em HSV, por exemplo, dois limiares delimitariam os comprimentos de onda máximo e mínimo, representando a cor desejada, dois para as saturações máxima e mínima e os últimos dois para a luminosidade. Com isso, é possível obter uma imagem binária, onde os valores iguais a 1 representam os pixels que contém a cor desejada.

A Figura 3 apresenta um exemplo de segmentação onde foram usados quatro limiares de forma a destacar apenas o fundo azul da imagem. Para isso, foram utilizados um limiar mínimo de 180° e um máximo de 240° na matiz, um valor mínimo de 50% de saturação e um valor mínimo de 75% de luminosidade.

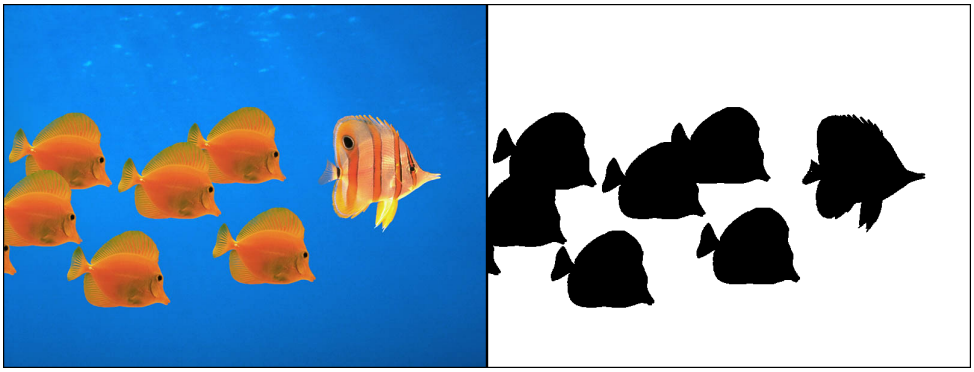

Figura 3 – Segmentação do fundo azul.

1.3.3 Distorção de Imagens

A representação de um objeto tridimensional em um ambiente bidimensional implica em perda de informação. Além de partes não visíveis do objeto, a projeção pode conter distorções, alterações na cor e outras mudanças capazes de dificultar a visualização da

imagem. Essas mudanças geométricas do objeto são causadas por seu formato (nesse caso por sua profundidade) e estão relacionadas com a postura da câmera e com a lente da mesma. (MANSUROV, 2015)

A distorção é uma deformidade na imagem, na qual as perdas de informações podem ser minimizadas. Ela pode ser calculada ou mapeada de forma com que a imagem original possa ser corrigida. Há dois tipos de distorção: tangencial e radial. A distorção tangencial depende da posição da câmera em relação ao objeto, onde objetos próximos aparentam ser maiores que os mais distantes. A distorção radial é causada principalmente pelo design de suas lentes e pode ser dividida em três: positiva, negativa e complexa. A distorção positiva representa a forma mais comum de distorção radial e ocorre quando a imagem necessita ser comprimida para caber no campo de visão da câmera, fazendo com que as linhas retas localizadas nas bordas da imagem fiquem curvadas para dentro. (MANSUROV, 2015)

A Figura 4 representa como as distorções afetam uma figura plana.

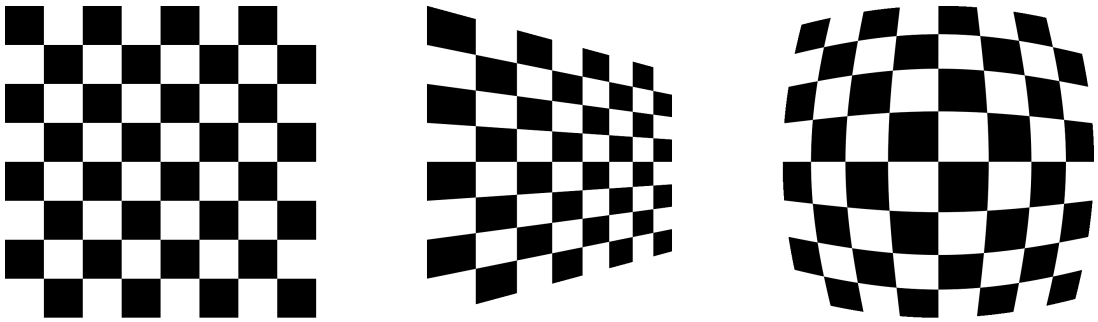

Figura 4 – Representações de uma figura (a) sem distorção, (b) com distorção tangencial e (c) com distorção radial positiva.

Para corrigir os efeitos de distorção de uma imagem, é necessária uma calibração geométrica da câmera, a qual está relacionada com os parâmetros da lente e do sensor desta. Tendo conhecimento dos parâmetros, é possível corrigir distorções e, assim, medir e localizar objetos. A calibração é altamente empregada nas áreas de robótica, sistemas de navegação e reconstrução de ambientes 3D. (MATHWORKS, 2017)

A deformação radial representa uma deformação não linear da imagem. Conforme Luhmann et al. (2014), imagens com deformações não lineares podem ser modeladas utilizando equações polinomiais. O uso de equações polinomiais é simples e bastante empregado na correção de distorções e podem ser dadas por (1.6) e (1.7).

$$\begin{aligned} x_p = {} & a_1 + a_2x + a_3y + a_4xy + a_5x^2 + a_6y^2 + a_7x^2y + a_8xy^2 + \\ & + a_9x^3 + a_{10}y^3 + a_{11}x^3y + a_{12}x^2y^2 + a_{13}xy^3 + a_{14}x^4 + a_{15}y^4 \end{aligned} \tag{1.6}$$

$$y_p = b_1 + b_2x + b_3y + b_4xy + b_5x^2 + b_6y^2 + b_7x^2y + b_8xy^2 + \\ + b_9x^3 + b_{10}y^3 + b_{11}x^3y + b_{12}x^2y^2 + b_{13}xy^3 + b_{14}x^4 + b_{15}y^4 \tag{1.7}$$

onde, x_p e y_p representam os pontos da imagem planificada, x e y são os pontos distorcidos da imagem obtida, a e b são os coeficientes da distorção da lente.

O número de termos dos coeficientes a e b varia de acordo com a precisão desejada. Geralmente, para casos em que a precisão é pouco importante, define-se apenas os seis primeiros termos. Enquanto que para casos onde seja necessária uma correção com precisão muito elevada, pode-se considerar todos. (MATHWORKS, 2017)

1.4 Objetivos

Este trabalho tem como objetivo geral descrever e analisar o programa criado para identificar posturas de robôs, com aplicações na categoria VSS de futebol de robôs. Para tal, são esperados resultados de postura de modo contínuo, preciso e com um baixo tempo de execução.

De uma forma mais detalhada, pode-se citar cinco objetivos específicos:

- Apontar e definir o programa utilizado, assim como suas funcionalidades;
- Identificar e explicar os principais assuntos relacionados ao processamento de imagens;
- Demonstrar as aplicações do programa na categoria VSS de futebol de robôs;
- Organizar e reunir dados obtidos com os testes realizados; e,
- Analisar e debater os resultados obtidos para definir as aplicações.

2 Metodologia

2.1 Padrão de Cores

O método para identificação dos robôs para esse estudo é por padrão de cores localizado em sua parte superior, logo, torna-se fundamental uma boa escolha do padrão de cores dos robôs.

De acordo com LARC/CBR (2009)

> Uma etiqueta azul ou amarela, definida pelos organizadores, irá identificar os robôs de cada time. Todos os robôs devem ter visivelmente no topo, uma região sólida da etiqueta do time, seja ela na cor azul ou amarelo. Essa região pode ser de qualquer forma, mas deve ser capaz de conter (pelo menos) um quadrado com 3,5cm de lado ou um círculo com 4cm de diâmetro.

Além disso, LARC/CBR (2009) explica que a cor de identificação individual dos robôs não pode ser laranja, branca ou cinza e que os robôs utilizados devem ter lados menores que 75mm.

O padrão escolhido consiste em dois círculos com centros que pertencem à diagonal do lado superior do robô. A Figura 5 demonstra a vista superior dos seis tipos diferentes possíveis para as cores escolhidas. O círculo maior representa a cor do time (azul ou amarelo, por padrão) e o menor identifica individualmente o robô, para o qual foram escolhidas as cores vermelho, verde e magenta, devido principalmente à diferença no espectro dessas cores.

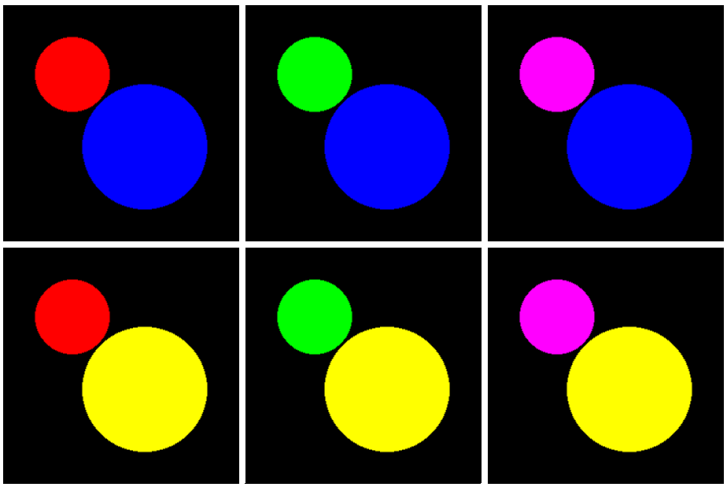

Figura 5 – Padrão de cores escolhido para os três robôs do time azul e amarelo.

A Figura 6 demonstra o padrão de cor escolhido, lembrando que as cores podem variar entre os robôs e os times. O círculo maior possui 40mm de diâmetro, com o centro deslocado à 7,5mm no eixo x e no eixo y em relação à origem do robô. O círculo menor possui 24mm de diâmetro, com o centro deslocado à 15,5mm no eixo x e no eixo y em relação à origem do robô.

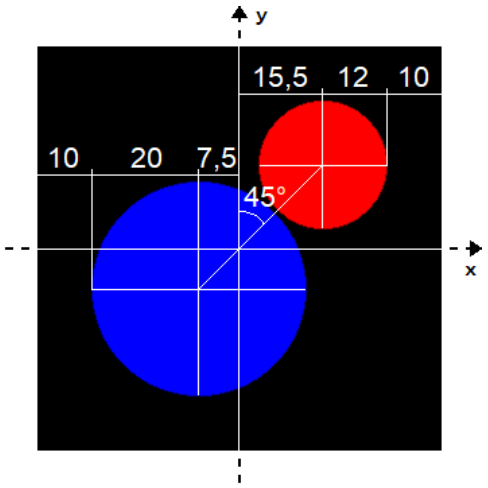

Figura 6 – Medidas relacionadas às etiquetas das cores.

Os centroides das cores serão dados obtidos pelo programa, logo, pode-se encontrar então o centroide do robô, de acordo com as Equações (2.1) e (2.2), e a sua orientação, na Equação (2.3).

$$x_{robo} = \frac{15,5x_{cor1} + 7,5x_{cor2}}{23} \tag{2.1}$$

$$y_{robo} = \frac{15,5y_{cor1} + 7,5y_{cor2}}{23} \tag{2.2}$$

$$\Psi_{robo} = tan^{-1}\left(\frac{y_{cor2} - y_{cor1}}{x_{cor2} - x_{cor1}}\right) + 45° \tag{2.3}$$

onde $cor1$ representa a cor do time e $cor2$ representa a cor individual dos robôs. Para o caso da orientação, deve-se considerar o quadrante trabalhado, a fim de saturar o valor obtido.

A escolha do padrão de cores em forma circular deveu-se a dois fatores principais. Primeiramente, por sua simplicidade e ocupação de espaço, já que círculos são de simples confecção e, na diagonal de um quadrado, ocupam pouco espaço mesmo com áreas relativamente grandes. O outro fator corresponde à filtragem de imagens binárias, já que estes eliminam, além de ruídos, quinas de objetos. Um objeto circular binário apresenta menores deformações em filtragens, quando comparados com objetos poligonais. A deformação

de um objeto dificulta a identificação de duas principais características do programa: o centroide e a área. A Figura 7 demonstra a comparação na filtragem de objetos circulares e poligonais.

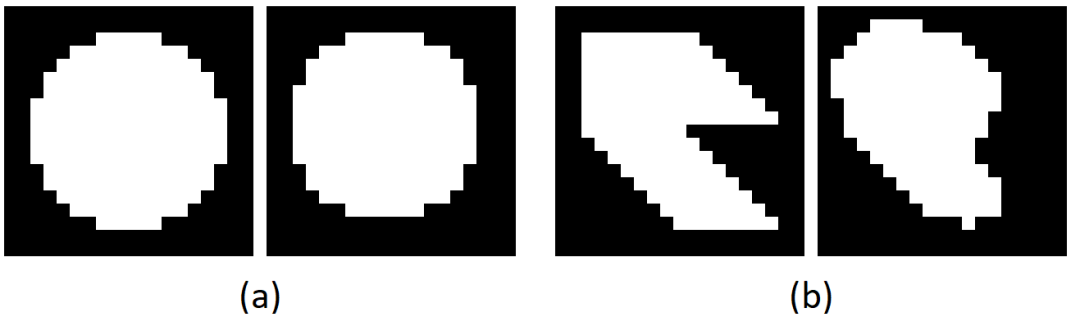

Figura 7 – Filtragens dois objetos binários: o primeiro (a) de forma circular e o segundo (b) de forma poligonal.

2.2 Filtro Utilizado

Existem diversos tipos de filtros já implementados capazes de filtrar imagens de diversas camadas, utilizando vários tipos de filtros. Porém, no caso atual, a imagem utilizada é muito simples (binária de uma única camada), e, além disso, os objetos possuem formas conhecidas.

Foi desenvolvido um filtro de convolução 3x3 específico para filtrar imagens de uma camada (binária ou inteira de 8 bits) com um tipo específico para objetos circulares. O valor de um elemento na imagem final é obtido através sa média ponderada dos pesos dos pixels vizinhos determinados na máscara. Esta foi determinada como sendo de peso 4 para o seu valor central, de peso 2 para os quatro elementos vizinhos ortogonais e de peso 1 para os elementos vizinhos da diagonal, de acordo com (2.4).

$$w(x,y) = \frac{1}{16}\begin{bmatrix} 1 & 2 & 1 \\ 2 & 4 & 2 \\ 1 & 2 & 1 \end{bmatrix} \tag{2.4}$$

O tamanho do filtro foi escolhido de acordo com a resolução e o tamanho dos objetos observados. Por sua vez, os valores da máscara foram escolhidos de acordo com o formato dos objetos, já que a máscara se aproxima de uma representação de um círculo.

2.2.1 Consideração da Altura dos Objetos

Conforme explicado na Subseção 1.3.3, os problemas relacionados às posições x e y de um objeto em uma imagem com distorção podem ser resolvidos através de equações

polinomiais. Porém, além disso, as distorções podem gerar outros dois erros na identificação de objetos.

O primeiro é referente à altura do objeto, que causa um deslocamento na posição obtida das cores. Isso ocorre nos casos em que a altura da câmera não pode ser considerada como infinita, o que resulta em um ângulo entre o objeto e a câmera e, consequentemente, um deslocamento da posição encontrada, o qual depende da altura do objeto, conforme ilustrado na Figura 8.

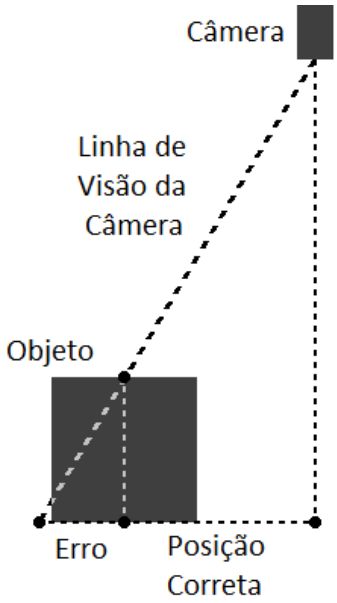

Figura 8 – Erro de posição causado pelas alturas do objeto e da câmera.

Assim, a câmera identifica a posição do objeto como sendo a posição correta somada a um erro. Para resolver esse tipo de problema, deve-se ter conhecimento das alturas do objeto e da câmera. O erro pode ser determinado através de semelhança simples de triângulos, conforme mostrado em (2.5) e subtraído da posição identificada pela câmera.

$$Erro = \frac{d_{obs}}{h_{cam}}.h_{obj} \tag{2.5}$$

onde d_{obs} é a posição identificada pela câmera, h_{cam} é a altura da câmera e h_{obj} é a altura do objeto.

O segundo erro que pode estar presente nesse caso é relacionado à área do objeto. Nesse caso, objetos próximos às bordas da imagem aparentam ser menores que os objetos próximos ao centro. Assim como o problema relacionado à altura dos objetos, isso ocorre nos casos em que a altura da câmera não pode ser considerada como infinita, fazendo com que a distância entre o objeto e a câmera dependa da posição do objeto no plano xy. Para esse caso, a diferença da área fica evidente quando há presença de distorções radiais ou tangenciais, conforme ilustrado na Figura 9.

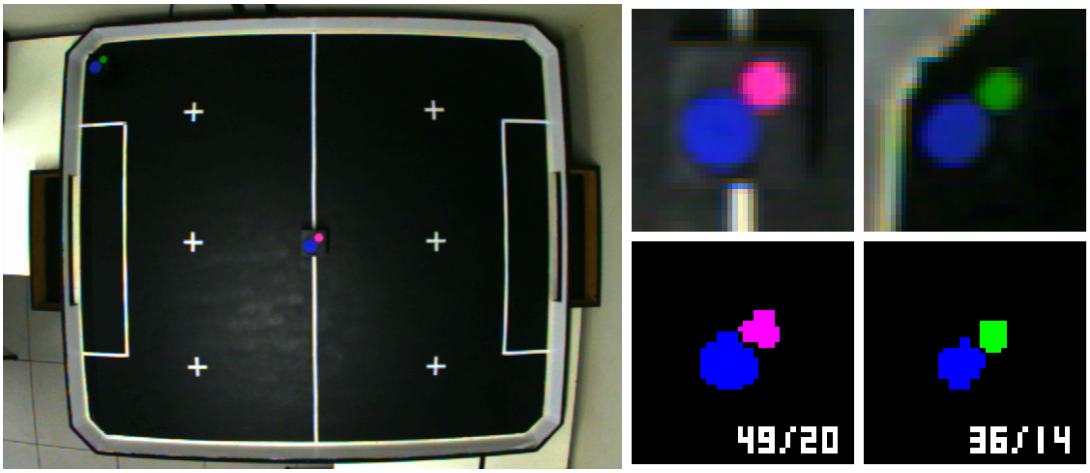

Figura 9 – Erro da área causado pela altura da câmera.

Na Figura 9, à esquerda, pode-se observar um teste realizado, onde foram postos dois robôs, um no centro do campo e outro na borda. À direita estão os resultados obtidos, onde as áreas encontradas das cores azul e magenta para o robô no centro foram 49 e 20 pixels, respectivamente. Enquanto que, para o robô na borda do campo, as áreas encontradas das cores azul e verde o centro foram 36 e 14 pixels, respectivamente, indicando uma redução de 30%.

Foi percebido que a área do objeto está relacionada à distância do objeto ao centro da imagem, logo, como as distorções na imagem não são lineares, foi inferida a utilização uma equação polinomial de segundo grau para determinar a área real do objeto, conforme mostrado em (2.6).

$$S_{real} = S_{obs} + k_1 d + k_2 d^2 \tag{2.6}$$

onde S_{real} representa a área real do objeto, S_{obs} é a área observada pela câmera, d é a distância do objeto ao centro da imagem e k_1 e K_2 são constantes que podem ser determinadas experimentalmente.

2.3 Programa Utilizado

O MATLAB® é um *software* interativo utilizado principalmente para computação numérica e gráfica. Como o próprio nome sugere, o MATLAB® é uma ferramenta especializada em cálculos com matrizes. Ademais, possui uma grande capacidade na criação gráficos e na criação de interfaces, utilizando sua própria linguagem de programação. (GOCKENBACH, 1999)

Para realizar o processamento digital de imagens no reconhecimento de robôs, o MATLAB® foi escolhido como programa de uma forma exclusiva. Diversos foram os

motivos determinantes para essa escolha, dentre eles: a facilidade na manipulação de imagens e matrizes, a simplicidade na criação de uma interface gráfica e a familiaridade dos alunos que fizeram parte desse projeto com esse programa.

De uma forma mais específica, tratando-se do jogo de futebol de robôs, o objetivo do programa criado é encontrar a postura dos robôs e a posição dos adversários e da bola através de uma imagem capturada de uma câmera localizada acima do campo. Para isso é necessária uma rotina de jogo, que se inicia na captura da imagem e finaliza na postura final dos objetos.

Primeiramente, antes do início da rotina de jogo, é necessário realizar uma sequência de procedimentos, sendo eles:

- Configuração da câmera;
- Calibração das cores;
- Calibração do campo; e,
- Definição de constantes.

A rotina de jogo pode ser definida na seguinte sequência:

- Captura, aperfeiçoamento e segmentação da imagem;
- Localização das cores; e,
- Definição dos robôs e da bola.

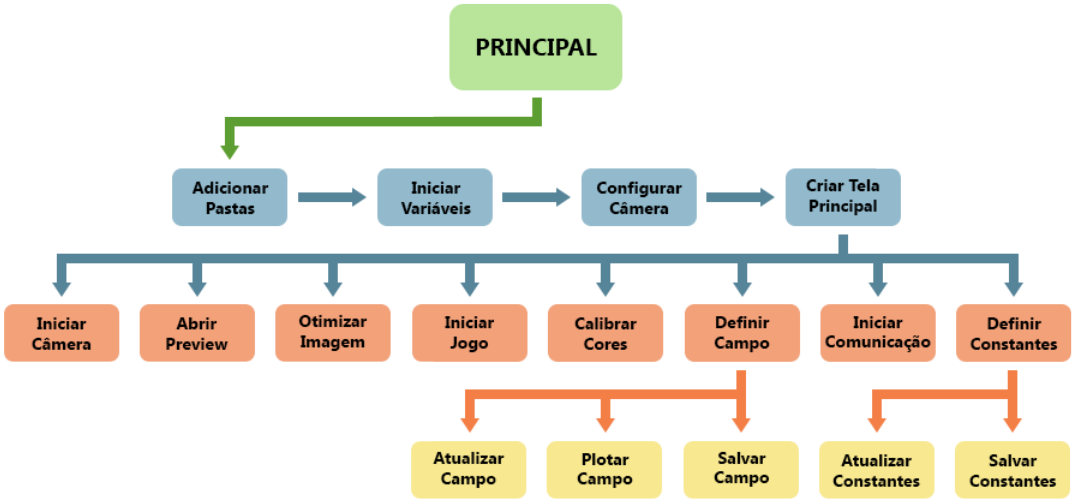

Figura 10 – Diagrama do programa principal criado.

As Figuras 10, 11 e 12 apresentam um diagrama do programa de uma maneira mais detalhada. A calibração de cores e a rotina de jogo são mais completos e, por isso, foram divididos em diagramas diferentes.

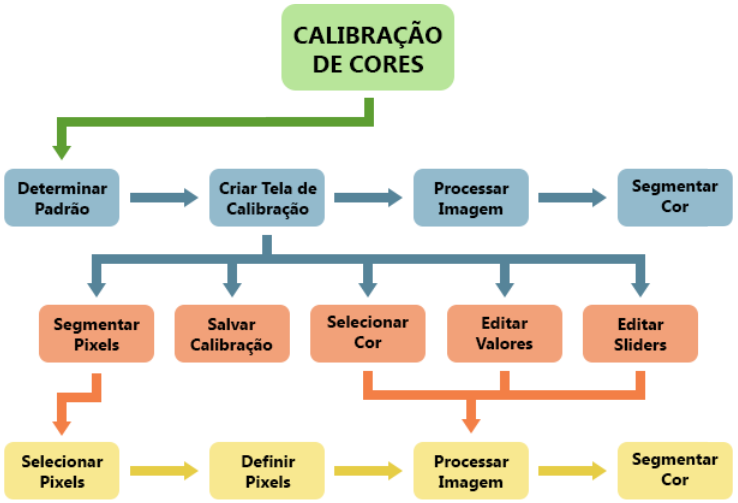

Figura 11 – Diagrama do programa de calibração de cores.

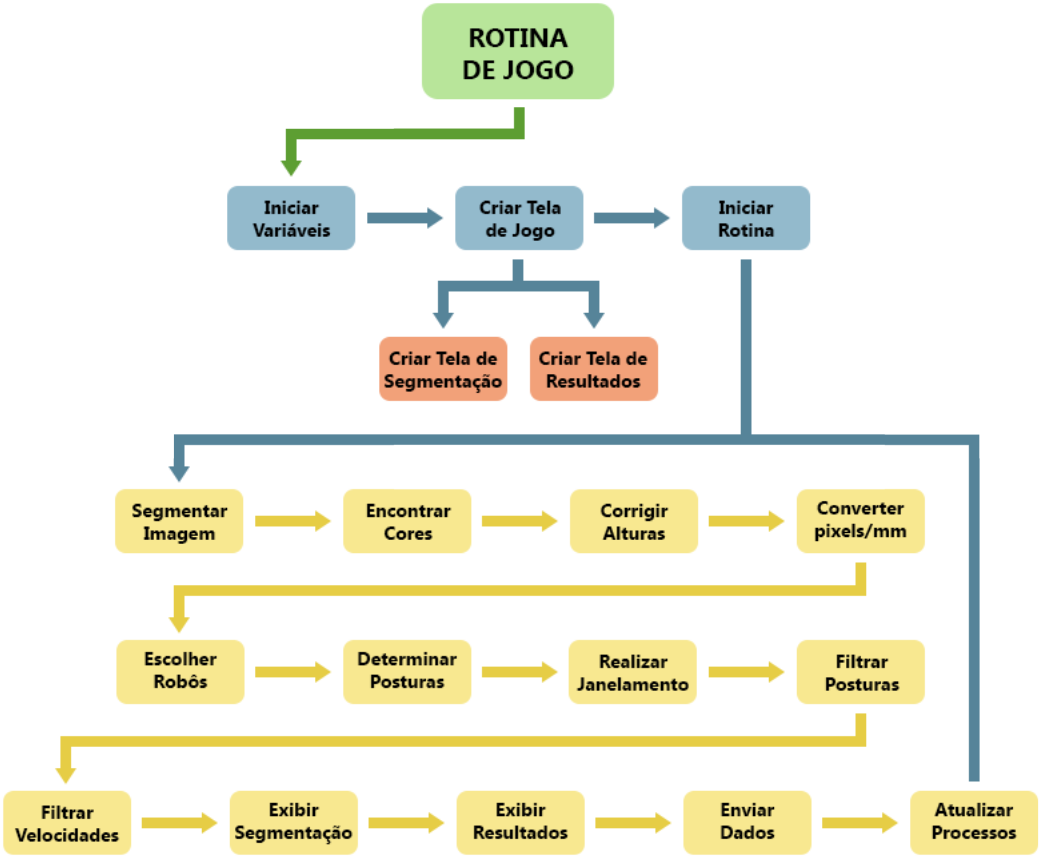

Figura 12 – Diagrama do programa de jogo.

2.4 Definição das Variáveis (Pré-Jogo)

Antes de cada partida, é de suma importância realizar uma calibração da câmera, do campo, das cores e das constantes. O objetivo dessa calibração é minimizar os erros, aumentando a precisão das posturas obtidas dos objetos. Para os casos do pré-jogo onde é necessária a captura da imagem da câmera, esta é feita seguindo uma sequência de passos. Primeiramente, o processo de captura de imagem é feito através da função *getsnapshot* (ver apêndice). Em seguida, é feito um redimensionamento da imagem para agilizar os próximos passos da rotina de jogo, pois o tempo de processamento depende em grande parte do tamanho da imagem. Para redimensionar a imagem, utiliza-se a função *imresize* (ver apêndice). Por fim, converte-se a imagem de RGB para HSV, utilizando da função *rgb2hsv* (ver apêndice).

Um *script* em MATLAB® foi criado, a fim de configurar inicialmente a câmera para uma captura RGB na resolução de 480p, utilizando a função *videoinput* (ver apêndice). Depois de iniciada as variáveis da câmera, o programa gera uma tela principal, a qual possui diversos botões relacionados ao pré jogo, conforme mostrados nas Figura 13. Os estados botões dependem das variáveis iniciadas, a Figura 14 ilustra o programa principal pronto para a rotina de jogo.

Figura 13 – Tela principal do programa sem as variáveis iniciadas.

Os três primeiros botões estão relacionados à câmera. O primeiro inicia e encerra a captura de vídeo da câmera, alterando o estado dos botões dependentes do vídeo. O segundo abre uma nova janela contendo a visualização atual da câmera, mostrada na Figura 15, a qual, no MATLAB®, é importante estar aberta para diminuir o atraso no

Figura 14 – Tela principal do programa com as variáveis iniciadas.

tempo de captura da imagem. O terceiro botão abre uma janela contendo as propriedades da câmera (ganho, saturação, nitidez, etc.), mostrada na Figura 16. Além disso, este botão determina valores para as propriedades da câmera, de modo a aumentar a frequência das imagens capturadas, já que esta depende de algumas propriedades como o tempo de exposição e o obturador da câmera.

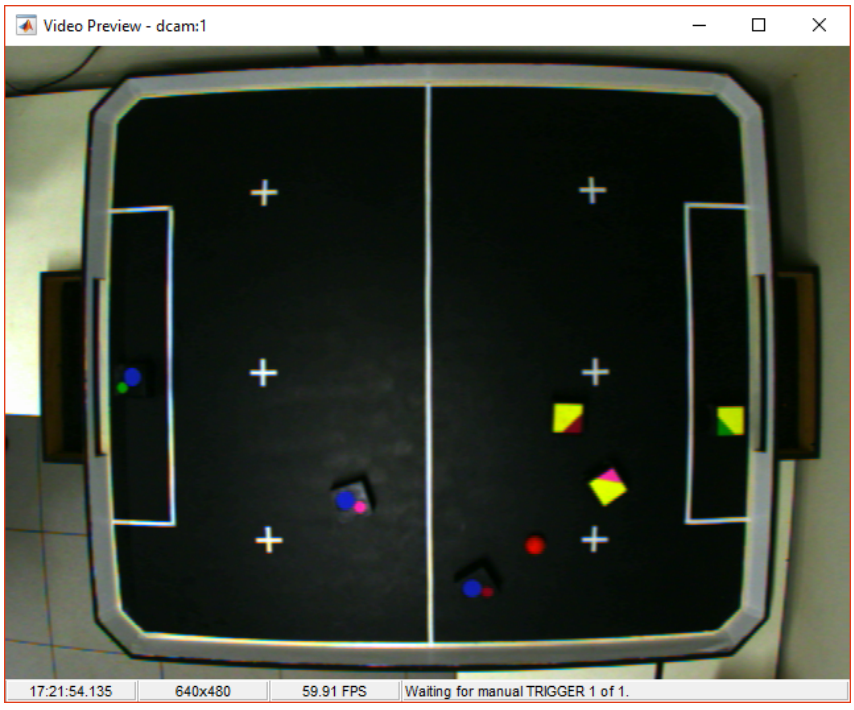

Figura 15 – *Preview* da câmera.

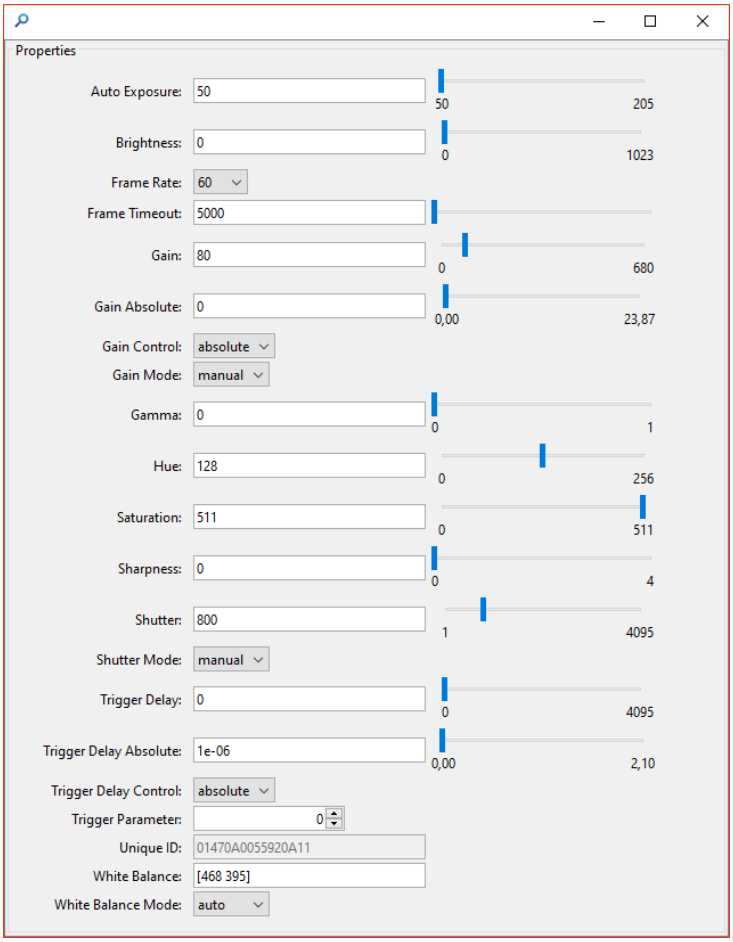

Figura 16 – Tela de propriedades da câmera editáveis pelo programa.

Dos quatro últimos botões, três estão relacionados à calibração pré-jogo e um à comunicação dos dados pela rede (que não será tratada nesse trabalho).

O botão de calibração de cores cria uma nova tela, onde é possível alterar os limiares de segmentação de cada cor. Utilizando a imagem da Figura 15 (correspondente ao *preview*), foram geradas as segmentações de cada cor e a segmentação total, utilizando os limiares salvos na calibração de cores, conforme mostradas nas Figuras 17 e 18. Conforme dito na Subseção 1.3.2, o método de determinação dos limiares por tentativa e erro é pouco preciso, porém é simples e também interativo, já que o usuário pode visualizar suas modificações em tempo real. Desse modo, sempre que algo é selecionado ou modificado nessa tela, a imagem segmentada é atualizada para a cor selecionada. Há três maneiras de modificar os limiares: através das caixas de texto (em graus [0,360] para a matiz e em porcentagem [0,100] para a saturação e a luminosidade), através dos *sliders* (aumentando e diminuindo seu valor de uma maneira mais interativa) e selecionando os pixels manualmente (através do botão segmentar pixels). Clicando nesse botão, uma nova tela é criada, onde é possível selecionar os pixels que correspondem a cor escolhida, mostrado na Figura 19. Para eliminar os pixels selecionados de forma incorreta, os limiares são definidos utilizando a média e o desvio padrão dos pixels. O último botão dessa tela salva a variável que contém os limiares em

um arquivo externo, para que possa ser carregado ao iniciar o programa novamente.

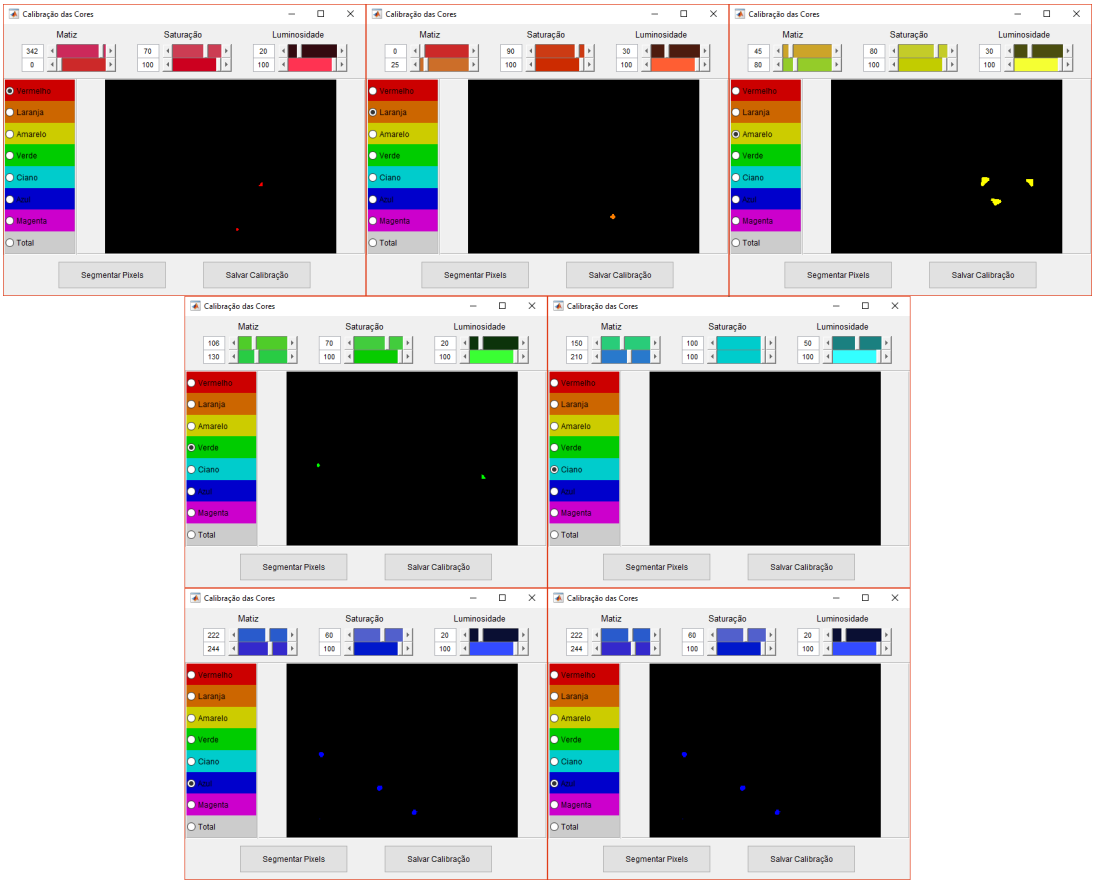

Figura 17 – Telas de calibração para cada cor.

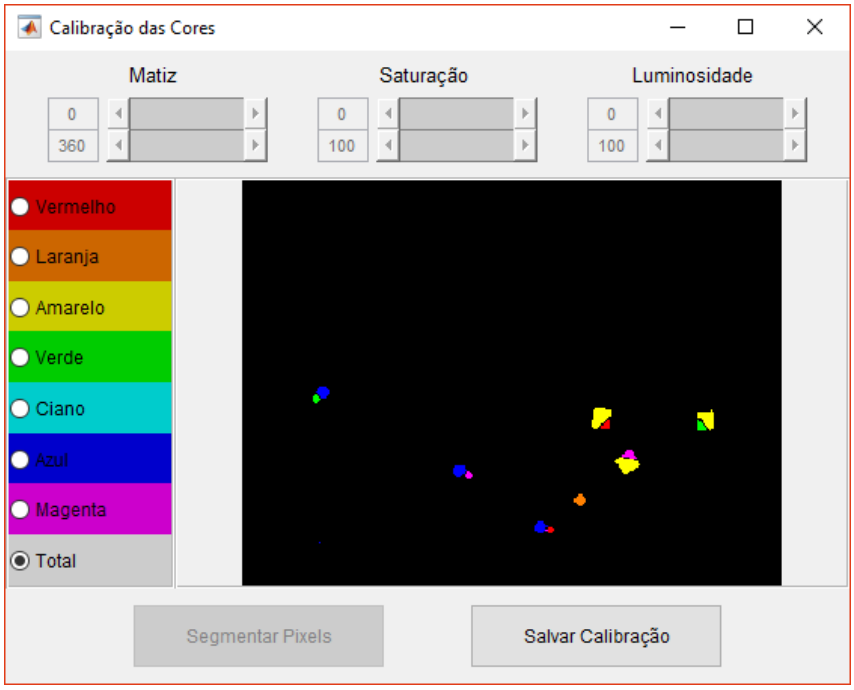

Figura 18 – Tela de calibração de todas as cores.

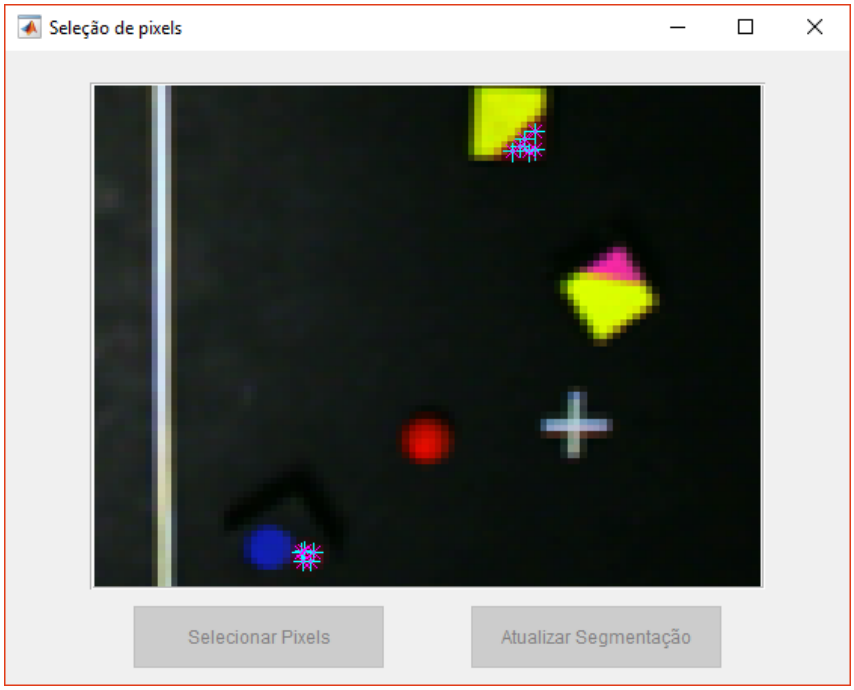

Figura 19 – Tela de seleção de pixels (exemplo para seleção de pixels vermelhos).

O botão de definição do campo também inicia uma tela, na qual é exibida uma imagem atual do campo. A principal função desse botão é a de criar as constantes necessárias para realizar a projeção geométrica do campo e, posteriormente, converter as posições dos objetos de pixels para milímetros. Foi utilizada, para corrigir a distorção da imagem, uma transformação polinomial de grau 4 (ver Subseção 1.3.3), a fim de corrigir de uma forma mais precisa tanto as distorções tangenciais quanto as radiais. A função utilizada é a *fitgeotrans*, que determina uma transformação geométrica. A Figura 20 ilustra a tela de definição do campo. Os pontos vermelhos no campo indicam a posição atual salva dos pontos chave convertidos de milímetros para pixels. O botão de atualização do campo inicia uma nova marcação dos pontos, que serão definidos pelo usuário, conforme a Figura 21. Há ainda um botão para salvar a relação entre pixels e milímetros em um arquivo externo, para que não seja necessário que o usuário defina o campo em todo início de programa.

O botão de definição de constantes cria outra tela, contendo as constantes necessárias para a realização do jogo, conforme mostrada na Figura 22. Nessa tela, é possível alterar as distâncias máxima e mínima entre as cores principal e a secundária (que formam um robô), as áreas máxima e mínima das cores, as cores dos objetos, o tempo de envio de dados (o qual não será tratado nesse trabalho) e a altura dos objetos (câmera, robôs e bola). É possível também visualizar as distâncias e áreas encontradas na última rotina de jogo, para que se possa ter uma ideia dos valores das constantes. Existe ainda um botão para salvar as constantes em um arquivo externo.

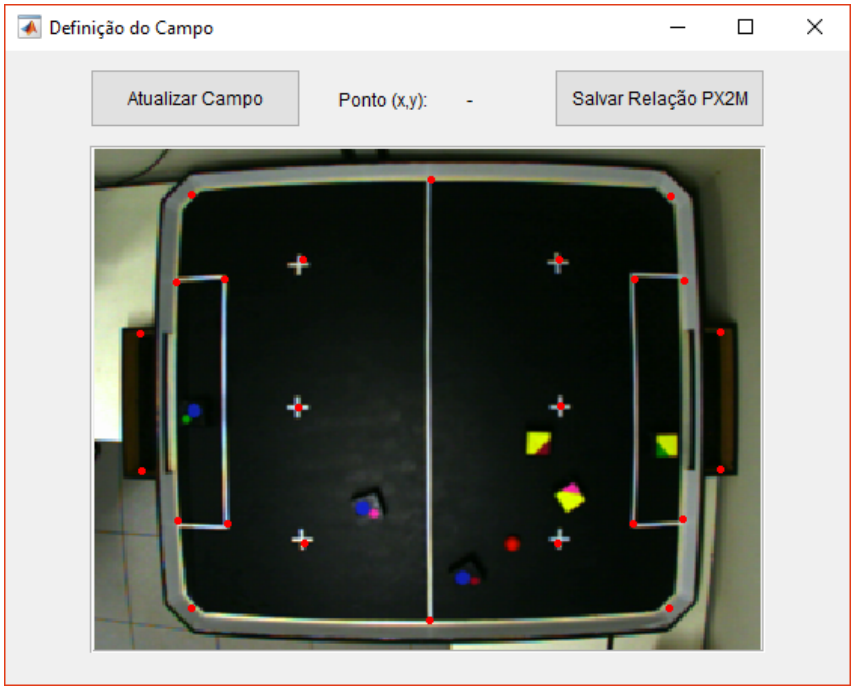

Figura 20 – Tela de definição do campo com as marcações do campo atual.

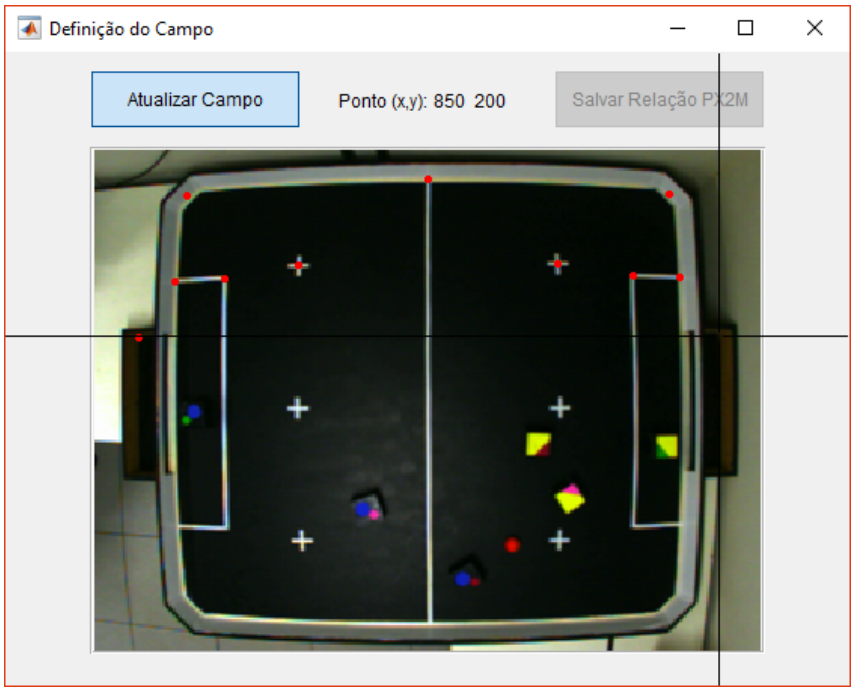

Figura 21 – Tela de definição do campo durante a seleção dos pontos chave.

Voltando à tela principal, o botão central para iniciar o jogo é habilitado somente se todas as variáveis necessárias existirem. Ao clicar nesse botão, uma nova tela é criada, as variáveis de jogo são iniciadas e o programa entra em um *loop*, representado pela rotina de jogo. Nessa tela há dois botões: o primeiro exibe a imagem segmentada encontrada e o segundo exibe os resultados encontrados (posição e postura dos objetos), ambos em tempo

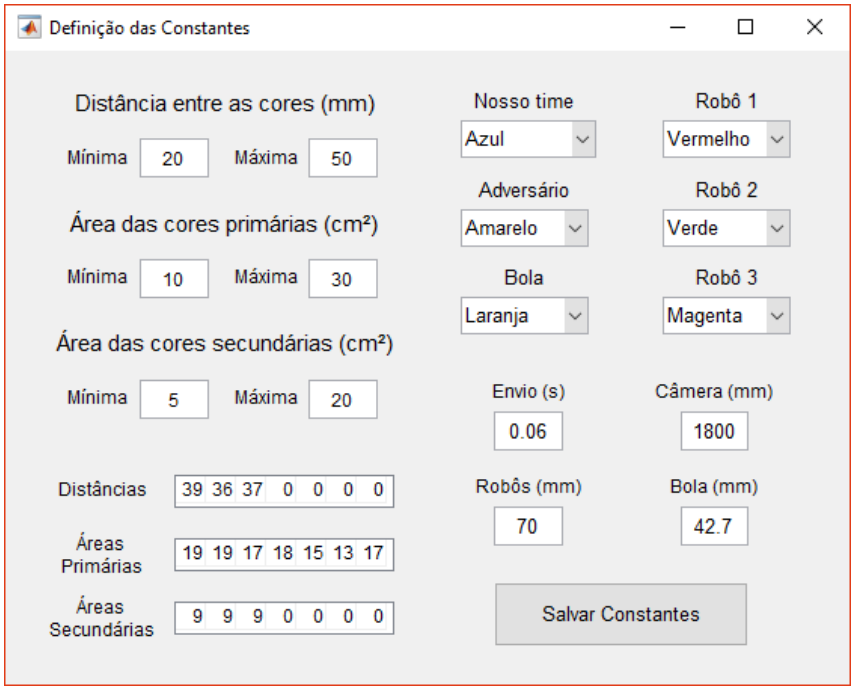

Figura 22 – Tela de constantes.

real. Por padrão, ambos não são mostrados, pois, no MATLAB®, as funções para exibir a segmentação e os resultados demandam um tempo considerável, atrasando a atualização dos dados. As Figuras 23 e 24 mostram, respectivamente, os resultados da segmentação e da postura determinados pelo programa para a imagem correspondente ao *preview*.

Figura 23 – Tela de resultados da segmentação.

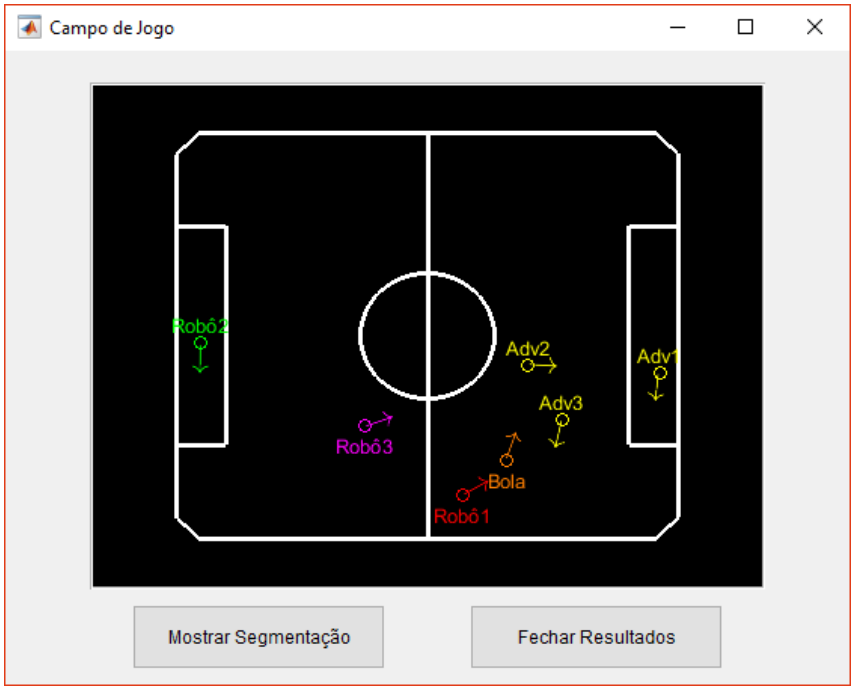

Figura 24 – Tela de resultados das posturas encontradas.

2.5 Rotina de Jogo

2.5.1 Segmentação das Cores

Após a calibração das cores, do campo e das constantes, pode-se iniciar a rotina de jogo. O primeiro passo da rotina é a segmentação da imagem. Conforme foi dito na seção 2.4, a resolução configurada da câmera é 480p e o sistema de cores da imagem capturada é RGB, logo deve-se seguir os passos para aperfeiçoar a imagem. O processo de segmentação pode ser dividido nas seguintes etapas:

- Captura da imagem;
- Redimensionamento da imagem;
- Conversão do sistema de cores de RGB para HSV;
- Segmentação da matiz, da saturação e da luminosidade;
- Interseção das três segmentações; e,
- Filtragem de cada imagem segmentada.

A segmentação da imagem é feita dentro uma sequência de passos que percorre as cores que serão usadas. São utilizados os limiares salvos na calibração de acordo com a Equação (1.5). Primeiramente faz-se a segmentação da matiz. A resposta é binária e recebe

1 para valores entre os limiares e 0, caso contrário. A matiz do sistema é representada em graus, ou seja, ela pode por exemplo ter um valor mínimo de 330° e máximo de 15°. Logo, deve-se tomar um certo cuidado com os valores de seus limiares. O mesmo é feito para a saturação e para a luminosidade.

Após a segmentação, cada cor terá três matrizes binárias, cada uma contendo a segmentação de cada camada. Para reuni-las em uma só imagem, foi utilizada a operação lógica AND, já que as segmentações realizadas devem respeitar todos os seis limiares utilizados.

Por fim, foi realizada uma filtragem nas imagens segmentadas, de forma a reduzir o nível de ruído. Para isso, foi implementada a função *FiltrarImagem*, descrita na Subseção 2.2.

2.5.2 Definição dos Centroides das Cores

Com a imagem pronta para ser trabalhada, primeiramente é necessário reconhecer as regiões de cada cor. Para isso, foi utilizada a função *regionprops* (ver apêndice), percorrendo as cores segmentada separadamente e retornando os centroides e as áreas das regiões de cada cor. Para uma maior facilidade na manipulação dos dados, foi feita uma organização das respostas encontradas pela função *regionprops*, salvando em três matrizes diferentes as posições no eixo x, no eixo y e as áreas das cores.

2.5.3 Correção dos Erros Relacionados às Alturas

Os robôs, a bola e a câmera possuem alturas fixas, logo podem haver erros de posição e nas áreas das cores encontradas. Para contornar esse problema foram utilizadas as Equações (2.5) e (2.6). As constantes utilizadas em (2.5) são fixas e são definidas antes do jogo, na tela de constantes. Já as constantes de (2.6) foram determinadas experimentalmente. Para isso, foi feita um ajuste de curva através da função *polyfit* (ver apêndice) utilizando valores medidos de distância e área em diversos pontos do campo.

2.5.4 Conversão de Pixels para Milímetros

O próximo passo a ser feito é a conversão dos centroides e das áreas de pixels para milímetros, utilizando as constantes salvas na relação de pixels para milímetros na tela de definição do campo. Esta variável, conforme dito na Subseção 1.3.3, considera as distorções do campo e foram utilizados os quinze termos das equações de conversão, mostradas em (1.6) e (1.7). Com a posição dos centroides em mm e as áreas em mm^2, pode-se filtrar novamente as cores utilizando agora as constantes com o tamanho das áreas definido na tela de constantes. Nesse ponto, as áreas maiores que o máximo permitido ou menores que mínimo são desconsideradas.

2.5.5 Definição dos Objetos

Com os centroides das cores desejadas em milímetros, pode-se agora definir quais associações de cores podem ser classificadas como robôs. Para isso, foram utilizadas estruturas de repetição, para salvar as distâncias entre todas as cores primárias e todas as cores secundárias encontradas, formando uma matriz tridimensional. De uma forma melhor explicada, se a cor do time é azul e as cores dos robôs são vermelho, verde e magenta, então essa matriz possui as distâncias de todos os vermelhos para todos os azuis na primeira camada, as distâncias dos verdes para os azuis na segunda e dos magentas para os azuis na terceira camada.

Utilizando os valores máximo e mínimo para, salvos na tela de constantes, pode-se excluir àquelas distâncias que não são candidatas a serem robôs. A matriz é então ordenada de forma crescente e seus dados são organizados em uma nova variável, para uma manipulação mais direta. Além dos robôs do próprio time, pode-se definir os candidatos à robôs adversários e à bola de uma forma mais direta, uma vez que estes necessitam apenas de uma cor para serem identificados.

2.5.6 Definição das Posturas

Antes de determinar quais candidatos realmente representam os robôs, deve-se encontrar as posições dos centroides dos robôs, e não de suas cores. Conforme dito na seção do padrão de cores, para calcular o centroide do robô, pode-se realizar uma média ponderada dos centroides das cores nos eixos x e y, de acordo com as Equações (2.1) e (2.2). Além disso, pode-se calcular também a orientação dos robôs somando 45° da linha entre as cores, conforme a Equação (2.3).

2.5.7 Janelamento das Posturas

As variáveis finais, até então, correspondem aos centroides dos candidatos a serem robôs e dos candidatos à bola, podendo haver mais de um do mesmo tipo, ou até mesmo nenhum. Para solucionar esse problema, foi utilizado a posição passada encontrada na última iteração. Caso exista apenas um valor encontrado de centroide para o objeto, esse é definido como sua posição atual. Caso existam mais de um candidato para o objeto, sua posição é definida como àquela que está mais próxima do objeto encontrado no ciclo anterior. Caso não tenha sido encontrado nenhum candidato, a posição do objeto atual é definida como sendo àquela encontrada no ciclo anterior.

Além disso, foi encontrado também o sentido de movimento dos robôs adversários e da bola. Assim, a variável final da postura de todos objetos é dada por: sua posição no eixo x em milímetros, sua posição no eixo y em milímetros e sua orientação (ou sentido de movimento) em graus.

Por fim, filtros de Kalman foram utilizados para suavizar os resultados de cálculos de postura e velocidades dos robôs.

2.5.8 Exibição da Segmentação e dos Resultados

A exibição da segmentação resultante da filtragem das cores é importante para observar as imagens com as quais o programa está trabalhando. Para exibir a imagem da segmentação, pode-se multiplicar cada cor por seu correspondente em RGB e somar as imagens. Na exibição a imagem final da segmentação foi utilizada a função *imshow* (ver apêndice) e pode ser visualizada ao clicar na opção de exibir a segmentação na tela de jogo.

Para exibir o resultado final das posturas, foi utilizada as funções *plot* e *text* (ver apêndice). O resultado final exibido é composto por um círculo que indica a posição do objeto, uma seta indicando sua orientação (ou sentido de movimento) e uma legenda que indica o nome do objeto.

O programa criado trabalha com botões e exibições de imagens de forma a interagir com o usuário em tempo real. Para isso, torna-se necessário a aplicação de uma função do MATLAB® chamada *drawnow* (ver apêndice) para atualizar e cada iteração as figuras e os processos envolvidos nos botões. Sem essa função, o programa se dedicaria apenas à rotina de jogo, reduzindo o tempo gasto, porém nada seria mostrado na tela e os botões não poderiam ser acionados.

3 Resultados e Discussões

3.1 Precisão da Postura

Utilizando um dos robôs como referência, foi realizado um teste simples para medir a precisão da resposta do programa. Para isso, o robô foi colocado nos pontos em formas de cruz do campo, os quais possuem posições definidas. A Figura 25 demostra como foi realizado esse teste.

As posturas desejadas, as posturas obtidas no experimento e os erros encontrados estão representados na Tabela 1.

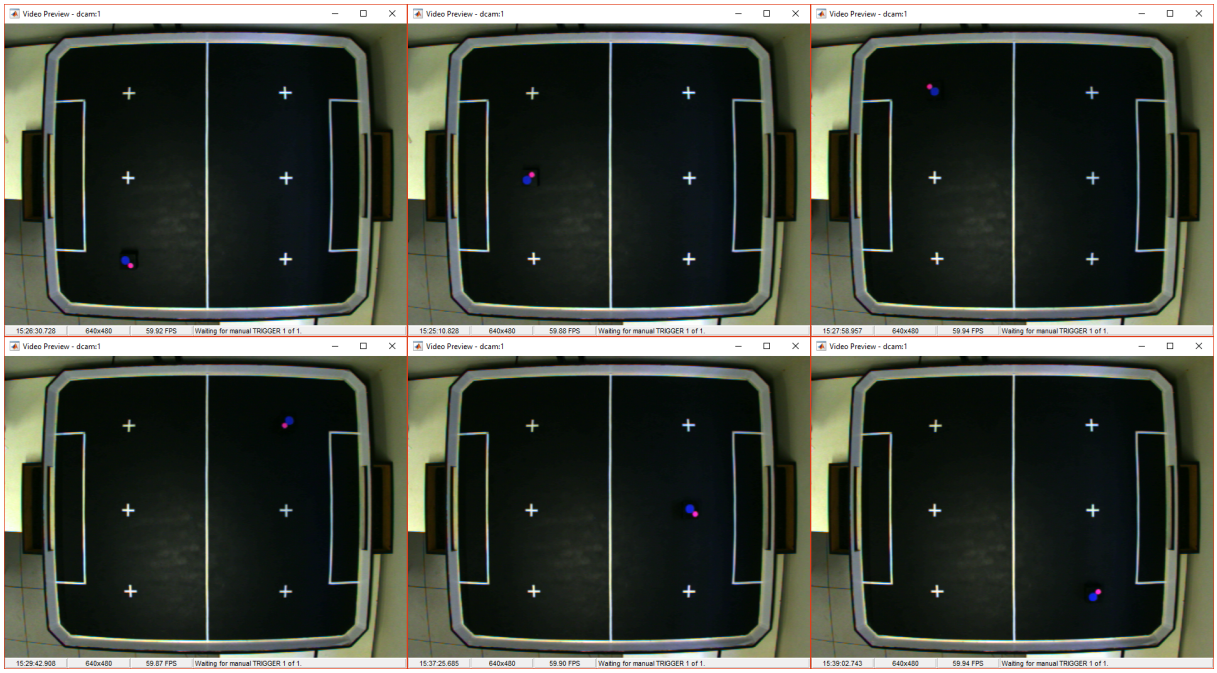

Figura 25 – Teste realizado para determinar a precisão dos resultados.

Tabela 1 – Dados desejados, dados obtidos e erros do teste de precisão da postura

Posturas desejadas			Posturas obtidas			Erros Encontrados			
x (mm)	y (mm)	Ψ (°)	x (mm)	y (mm)	Ψ (°)	x (mm)	y (mm)	d (mm)	Ψ (°)
-375	400	0	-380	387	-1	-5	-13	14	-1
-375	0	90	-370	-3	90	5	-3	6	0
-375	-400	±180	-378	-386	180	-3	14	14	0
375	400	-90	375	384	-88	0	-16	16	2
375	0	0	365	-2	4	-10	-2	10	4
375	-400	90	375	-390	91	0	10	10	1

Os erros absolutos máximo e médio encontrados na posição foram de 16mm e de

12mm, respectivamente. Já para a orientação obteve-se um erro absoluto máximo de 4° e médio de 1°.

De acordo com a Tabela 1, foi possível observar que os maiores erros ocorreram no eixo y, com um erro médio de 10mm, enquanto que no eixo x a média do erro foi de 4mm. No eixo y, os valores encontrados se aproximaram da origem desse eixo em cerca de 5%. É possível visualizar, de uma forma sutil, a ocorrência desse erro na Figura 20, onde alguns do pontos, depois da transformação geométrica, estão um pouco fora do ponto desejado. A utilização de campo oficiais, que possuem marcações estritamente precisas ajudaria a reduzir esse erro.

Os outros erros foram relativamente pequenos e foram causados principalmente por dois motivos: erro humano, já que o posicionamento dos robôs foi feito manualmente, gerando imprecisão e ruído causados pela iluminação, a qual, no local onde foram realizados os experimentos, depende da iluminação externa (variável ao longo do dia). Em ambientes fechados ou com boa iluminação interna, o erro seria reduzido.

3.2 Tempo de Execução

O tempo de execução de um programa representa algo de suma importância quando se trata aquisição de dados em tempo real. Para verificar o tempo gasto em um intervalo do programa, foi utilizada as funções *tic* e *toc* (ver apêndice), que analisam o tempo no ciclo de execução. De forma a aumentar a precisão, pode-se utilizar a média de n ciclos, calculada conforme (3.1).

$$t_o = \frac{1000\ toc(t_i) + (n-1)\ t_o}{n} \tag{3.1}$$

onde t_i recebe o valor inicial do temporizador (tic), n representa o número de ciclos e deve ser aumentado em uma unidade à cada ciclo e t_o representa a média atualizada em cada ciclo.

Foi analisada a resposta e o tempo gasto para o programa criado em uma simulação de uma partida normal, com os robôs e a bola posicionadas de forma arbitrária no campo. A Figura 15 mostra o *preview* da partida.

O teste do programa foi feito em um Desktop Dell XPS 8300, utilizando o Windows 10 Home 64 bits, com um processador Intel® Core™ i7-2600 CPU @ 3.40GHz (8 CPUs), memória RAM 8192MB 1333MHz, *driver* de vídeo AMD Radeon HD 6450 1GB DDR3 800MHz e uma câmera de vídeo Stingray F-046 C.

Depois de calibrados os limiares das cores, o campo e as constantes, foi iniciada uma rotina de jogo para a partida criada anteriormente. Na primeira parte do programa,

que inicia com a captura da imagem e finaliza com a filtragem, foram gastos 25,7ms com os seguintes tempos para cada seção:

- Captura da imagem: 1,21ms
- Redimensionamento da imagem: 3,57ms
- Conversão para HSV: 7,27ms
- Organização dos valores da imagem: 0,41ms
- Pré-alocação das variáveis: 0,11ms
- Segmentação das matizes: 1,73ms
- Segmentação das saturações: 1,66ms
- Segmentação das luminosidades: 1,44ms
- Operação lógica AND nas imagens: 1,72ms
- Filtragem da imagem segmentada: 6,56ms

A segunda parte do programa localiza os centroides das cores segmentadas e organiza a resposta encontrada. O tempo gasto foi de 9,03ms divididos em:

- Localização das áreas e dos centroides: 8,69ms
- Organização dos resultados: 0,34ms

O restante do programa é realizado de uma forma relativamente rápida, gastando 4,09ms divididos em:

- Correção dos erros relacionados às alturas dos objetos: 0,14ms
- Conversão dos pontos de pixels para milímetros: 0,92ms
- Definição dos robôs: 0,59ms
- Determinação das posturas: 0,17ms
- Realização do janelamento: 0,64ms
- Filtragem das posturas: 0,45ms
- Filtragem das velocidades: 0,14ms
- Envio dos dados: 1,04ms

Como dito na Seção 2, as partes do programa *filtragem das velocidades* e *envio dos dados*, fogem ao foco desse documento e não são tratadas. Além disso, para contagem do tempo, as duas exibições de respostas não são consideradas, já que, por padrão, nem são exibidas no programa. Estas duas tiveram os seguintes valores de tempo gasto:

- Exibição segmentação: 46,7ms
- Exibição resultados: 54,9ms

Foi medido o tempo gasto da função para atualização dos processos: *drawnow* (ver apêndice). Esta função, conforme dito na Subseção 2.5.8, é obrigatória e demanda em alto tempo de processamento com uma média de 11,8ms.

Desconsiderando a exibição dos resultados, o tempo médio gasto para realizar uma iteração completa foi de 50,6ms, provocando uma frequência de processamento de 19,76Hz.

Com os tempo gastos nas diversas partes do programa, pode-se analisar as funções que, individualmente, demandam mais tempo. Em ordem da maior para a menor, foram elas:

- *drawnow*: 11,8ms (cerca de 23% do total)
- *regionprops*: 8,69ms (cerca de 17% do total)
- *rgb2hsv*: 7,27ms (cerca de 14% do total)

Somadas, as funções demandam 27,8ms, o que corresponde a mais da metade do tempo de processamento da rotina de jogo. Para reduzir esse tempo, pode-se desenvolver um estudo sobre outras funções que possam ser mais eficientes.

4 Considerações Finais

Este trabalho apresentou e analisou o programa criado para identificar posturas de robôs, com aplicações na categoria VSS de futebol de robôs, cumprindo seu objetivo geral. Além disso, os resultados esperados com o programa foram aceitáveis, alcançando uma continuidade de resultados muito boa, com uma precisão boa e uma dinâmica razoável.

De um modo mais específico, foi possível apontar e definir as funcionalidades do programa, explicar os principais assuntos relacionados ao PDI, demonstrar as aplicações do programa na categoria VSS, reunir e analisar os dados dos testes realizados.

Acerca do programa, futuramente, pode-se buscar funções substitutas para aquelas que demandam um maior tempo de processamento. Além do mais, pode-se definir estratégias de inteligência para o movimento dos robôs e estabelecer uma comunicação entre o computador e os robôs.

Acerca do trabalho, pode-se explicar os dois passos da rotina de jogo que estavam fora do foco deste trabalho: as utilizações das velocidades dos robôs e a comunicação *UDP* utilizada entre os computadores.

Referências

GOCKENBACH, M. S. *A Practical Introdution to Matlab*. 1999. Acessado em junho de 2017. Disponível em: <http://www.math.mtu.edu/~msgocken/intro/intro.pdf>. Citado na página 21.

GONZALEZ, R. C.; WOODS, R. E.; EDDINS, S. L. *Digital Image Processing Using MATLAB*. 2. ed. Estados Unidos da América: Gatesmark, 2009. 557–561 p. Citado 2 vezes nas páginas 13 e 14.

JUNIOR, F. M. S. R. de M.; LIMA, G. G.; BRANDAO, A. S. *Equipe BDP/UFV: Desenvolvimento de Robô Autônomo da Categoria IEEE Very Small Size Soccer*. Viçosa-MG, Brasil: [s.n.], 2015. Citado na página 11.

LARC/CBR. *IEEE Very Small Size Soccer: Regras em Português*. [S.l.], 2009. Disponível em: <http://www.cbrobotica.org/wp-content/uploads/2014/03/VerySmall2009_ptbr.pdf>. Citado 3 vezes nas páginas 7, 10 e 17.

LUHMANN, T. et al. *Close-Range Photogammetry and 3D Imaging*. 2. ed. Berlim, Alemanha: De Gruyter, 2014. 34–35 p. Citado na página 15.

MANSUROV, N. *What is Distortion?* 2015. Acessado em junho de 2017. Disponível em: <https://photographylife.com/what-is-distortion>. Citado na página 15.

MATHWORKS. *What Is Camera Calibration?* 2017. Acessado em junho de 2017. Disponível em: <https://www.mathworks.com/help/vision/ug/camera-calibration.html>. Citado 2 vezes nas páginas 15 e 16.

RAPIDTABLES. *RGB to HSV Color Conversion*. 2015. Acessado em junho de 2017. Disponível em: <http://www.rapidtables.com/convert/color/rgb-to-hsv.htm>. Citado na página 13.

SOLOMON, C.; BRECKON, T. *Fundamentals of Digital Image Processing: A Practical Approach with Examples in Matlab*. Oxford, Reino Unido: Wiley Blackwell, 2011. 10–14 p. Citado na página 12.

Apêndices

APÊNDICE A – Figuras

drawnow: Atualiza as figuras e as chamadas de processos. A função é utilizada quando se deseja visualizar imediatamente uma modificação em um objeto gráfico ou em um botão.

fitgeotrans: Determina as constantes para a transformação geométrica para pares de pontos. O primeiro parâmetro é o vetor dos pares de pontos móveis, o segundo é o vetor de pontos fixos e o terceiro é o tipo da transformação.

getsnapshot: Retorna a imagem do último *frame* visto pelo objeto de vídeo. Utiliza um objeto de vídeo como parâmetro. A imagem retornada é uma matriz m x n x b, onde m é a altura da imagem, n é a largura da imagem e b é o número de bandas da imagem.

imfilter: Realiza o filtro de uma imagem. Como parâmetros, devem ser passados a imagem e o filtro, sendo que aquela pode ser de qualquer classe e dimensão.

imresize: Redimensiona de uma imagem em uma escala passada como parâmetro.

imshow: Exibe a imagem em uma figura aberta no MATLAB®.

polyfit: Determina os parâmetros de uma equações de grau n (passado como parâmetro) que representa a melhor curva para o conjunto de pontos passados.

plot: Plota um objeto (uma linha, por padrão) em um ambiente 2D utilizando o primeiro vetor (ou matriz) como o eixo das abscissas e o segundo como o eixo das ordenadas.

regionprops: Retorna uma série de propriedades dos objetos encontrados em uma imagem binária. As propriedades básicas retornadas são: Área (representa a quantidade de pixels da região), Centroide (corresponde ao centro de massa da região. Possui dois elementos, o primeiro representa a coordenada horizontal do centro de massa e o segundo representa a coordenada vertical) e Caixa delimitadora (Corresponde ao menor retângulo que contém a região. Para imagens 2D, contém quatro elementos, os dois primeiros representam o ponto de início da caixa e os dois últimos representam o tamanho da caixa).

rgb2hsv: Converte um ponto (ou imagem) do sistema de cores RGB para HSV. A terceira dimensão do HSV é normalizada e representa a matiz, a saturação e a luminosidade da imagem.

text: Cria um objeto de texto na posição indicada com parâmetro.

tic: Inicia um temporizador, salvando o tempo interno do computador no momento de sua execução.

toc: Lê o tempo passado desde o início do temporizador iniciado pelo *tic*. A saída é dada em segundos.

videoinput: Cria um objeto de vídeo. O nome do adaptador, o seu ID e o formato da imagem são os parâmetros principais dessa função, porém somente o nome do adaptador é obrigatório.