# Desenvolvimento de Visão Computacional do Robô Humanoide

**Mateus Menezes Azevedo Coelho**
Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, Brasil
Bolsista PIBIC-CNPq
mateuscoelho2009@gmail.com

**Paulo Marcelo Tasinaffo**
Instituto Tecnológico de Aeronáutica, São José dos Campos, SP, Brasil
tasinaffo@ita.br

***Resumo.*** *Nesse trabalho, aplicamos algoritmos e tecnicas de visão computacional com objetivo de identificar e extrair o máximo de informações dos objetos importantes para a Robocup. Identificamos traves brancas, bola de futebol, linhas de campo, escanteios e outros pontos importantes para tal competição.*

*Palavras chave: visão computacional, identificação de objetos, robótica, algoritmos.*

## 1. Introdução

A ITAndroids, equipe de robótica do ITA, é uma iniciativa de alunos do ITA que representa a instituição em diversas competições de robótica nacionais e internacionais. O principal objetivo dessa iniciativa é a integração dos alunos com atividades de pesquisa em engenharia, em especial em robótica e inteligência artificial.

Dentre as categorias de que a ITAndroids participa, destaca-se o futebol de robôs humanoides (*Robocup Humanoid Kid Size League*). Nessa competição, os robôs devem, autonomamente, perceber o mundo, planejar e realizar ações para marcar gols e defender seu lado do campo. Para cumprir esse objetivo, é de suma importância possuir um sistema de visão computacional eficiente e robusto. Um fator complicador é que, além de autônomo, o robô deve ser auto-contido, i.e. ele não pode usar qualquer informação ou energia de fontes externas. Assim, a qualidade dos sensores e o poder de processamento são inevitavelmente limitados, pois todo o hardware deve estar embarcado no robô. Além disso, o sistema de visão tem que funcionar no ambiente da competição, de modo que deve ser capaz de tolerar um nível de ruído considerável.

Técnicas no estado da arte para problemas gerais de visão computacional em geral envolvem alto custo computacional, o que as torna inviáveis para aplicação no futebol humanóide. Por outro lado, para permitir que os robôs consigam jogar futebol no estado atual de desenvolvimento, a organização da liga decidiu criar um ambiente codificado por cores, de modo que cada objeto no campo tem uma cor específica: a bola é laranja, as traves são amarelas, o campo é verde uniforme e as linhas são brancas. Além disso, robôs de um time devem usar "roupas" azuis, enquanto os do outro devem usar rosas. Também, os robôs devem ser preferencialmente pretos e não podem ter nenhuma cor que possa ser confundida com a de outros objetos. Estas simplificações juntamente com o pouco poder de processamento dos robôs fez a maioria dos times desenvolverem técnicas específicas para a liga, as quais se aproveitam do ambiente codificado por cor para processar as imagens com alta eficiência.
Nesse projeto, temos como objetivo a utilização de diferentes algoritmos de classificação de imagens em grupos baseados na cor, utilização de aprendizado de máquina para reconhecimento de objetos necessários em meio a partida de futebol humanoid, i.e. trave, bola e outros robôs, assim como a comparação e avaliação da eficiência de diferentes técnicas para a solução dos problemas descritos.

## 2. Descrição do Problema

Na competição de futebol de robôs humanoides (*Robocup Humanoid Kid Size League*), uma dos sensores válidos é a câmera. Através das imagens obtidas por ela, o robô humanoid deve retirar o máximo de informações para poder se localizar no campo e então marcar gols, conduzir a bola ou defender seu gol. Para isso, são usados algoritmos de visão computacional. Usamos o código base do time Austin Villa[2] , o qual foi disponibilizado publicamente.

Figura 1. Robô humanoide da equipe ITAndroids.

O desafio é identificar objetos de nosso mundo (três dimenções) dada uma imagem (duas dimensões). Além de extrair o máximo de informações de tal objeto reconhecido, como posição, distância com relação ao robô, entre outros dados.

As informações devem ser passadas ao robô somente em casos que está certa a detecção (há um problema muito maior em passar uma informação errada do que em não passar um dado correto), pois isso dificultaria bastante a localização do robô.

Os objetos a ser identificados são: traves; linhas do campo e encontros de linhas (i.e. o escanteio); bola; outros robôs.

Figura 2. Imagem de um robô em um campo de futebol humanoide. A bola na época ainda era laranja.

Traves: Um dos objetos de mais importância em meio a uma partida de futebol de robôs, é a trave. A sua detecção, passada com as informações corretas, permite uma maior orientação do humanoide além de ajudá-lo a tanto atacar quanto defender, facilitando a marcação de gols.

O algoritmo que operava no robô identificava em raros casos a trave amarela. Com a mudança das regras da competição, a trave passou a ser branca. Dessa forma, o desafio da trave se tornou maior (como se pode ver na Figura 2), pois o amarelo é uma cor pouco presente no campo, enquanto o branco da trave pode ser confundido com o das linhas e dos robôs.

Linhas do campo e encontro de linhas: A detecção de linhas do campo de forma correta permite uma melhor orientação do robô no campo, não dependendo somente da detecção de traves, que podem não estar dentro do campo de visão da câmera. Dessa forma, podemos identificar os limites do campo, assim como criar uma estratégia de *tracking* do gol adversário.

O algoritmo falhava na detecção de linhas do campo, além de errar bastante na detecção de encontros das linhas – detectando em meio a uma única linha, duas, e considerando isso um possível escanteio. O desafio das linhas e seus encontros em meio ao campo é melhorar sua detecção e evitar detecções erradas, assim como não realizar de forma alguma detecções erradas de escanteios, visto que para o algoritmo de localização, esses são pontos importantes.

Bola: O objeto mais importante no campo. A maioria das equipes brasileiras tem apenas essa detecção funcional, e ainda conseguem marcar gols e vencer a partida.

O algoritmo que operava no robô identificava muito bem a bola laranja, porém, em certos casos, fazia uma detecção errada. Além disso, com a mudança de regras, a bola se tornou branca com detalhes de outras cores. Dessa forma, o desafio da bola é criar um outro algoritmo para sua detecção.

Outros robôs: O reconhecimento de outros robôs tem sua importância para a evolução do futebol de robô. Ao identificar humanoids amigos e adversários pode-se ter um jogo mais avançado, com dribles e toques de bola, envolvendo uma estratégia mais inteligente. Porém, atualmente, poucos são os times que o fazem, principalmente devido ao foco principal que é a detecção da bola.

O algoritmo de detecção de robôs é pouco desenvolvido e não está dentre os focos dessa iniciação científica.

### 2.1. Redes neurais e *Backpropagation*

O algoritmo de redes neurais será explicado devido ao seu uso no trabalho.

Baseado no funcionamento de nossas células neurais – o que dá o próprio nome ao algoritmo – as redes neurais tem como objetivo "aprender", seja de forma supervisionada (caso haja um professor) ou não. Dada uma saída, a rede deve responder (saída) de forma adequada. As unidades básicas de formação de uma rede neural são os perceptrons (os quais tem como base os neurônios). Há uma camada de entrada e uma de saída, podendo haver camadas intermediárias (ou ocultas, visto que sua saída é apenas um sinal interno). Normalmente além das entradas, coloca-se um perceptron constante (um *bias*) que auxilia a convergência da rede, fora o *setup* inicial da mesma.

No nosso caso, usamos da teoria da *Backpropagation*. Nesse algoritmo, dadas as saídas da rede, é computado o erro (dada a saída desejada, logo é um algoritmo supervisionado), e usando fórmulas baseadas em cálculo diferencial (basicamente, anda-se no sentido contrário ao gradiente da função de erro, buscando, assim, diminuí-lo) podemos ajustar as constantes da rede de forma a adequar a saída desta a desejada.

$E = \frac{1}{2}\sum_{k}\left(t_k - o_k\right)^2$, onde $t_k$ é o valor desejado da saida e $o_k$ a saida atual. (1)

Para tanto, basta igualar as derivadas parciais a zero e iterar em busca do erro mínimo.

Neste algoritmo, há uma constante associada a cada ligação e uma função de ativação (em função da entrada do perceptron). No caso desse trabalho, foi utilizada a função sigmóide $f(\mathrm{x}) = 1/\left(1-\exp(-x)\right)$.

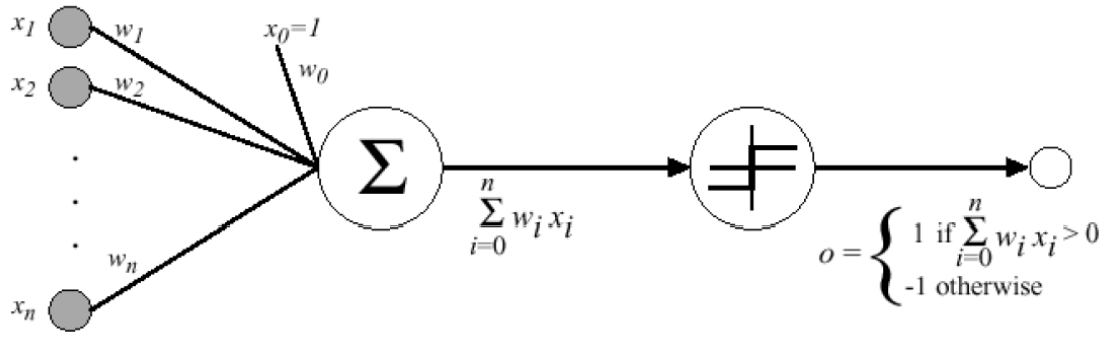


Figura 3. Neurônio discreto com função de ativação limiar.

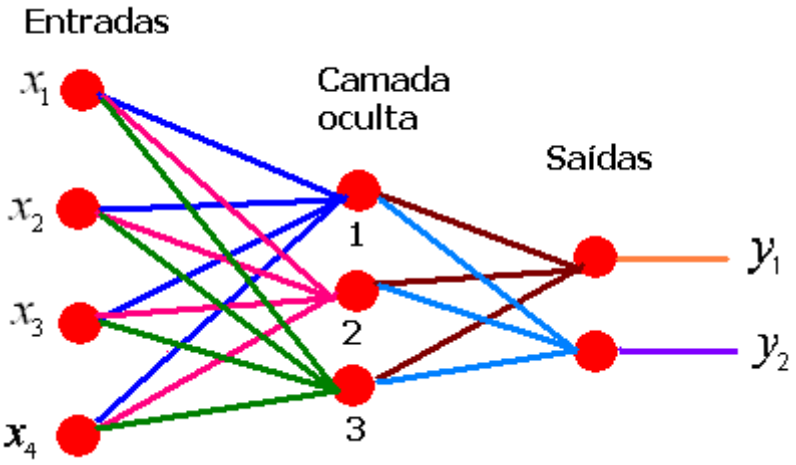


Figura 04 - Exemplo de rede neural que usa a *Backpropagation*. No caso foram utilizadas 3 camadas (uma de entrada com 4 perceptrons, uma intermediária com 3 percéptrons e uma camada de saída com 2 percéptrons).

Dessa forma, o número de camadas, o valor inicial das constantes, o número de percéptrons, são valores que afetam a convergência de tal algoritmo, devendo ser escolhidos com cautela.

### 2.2. *Speeded Up Robust Features* (algoritmo *SURF*) [3]

Esse foi o algoritmo utilizado para detectar a bola da competição.

Esse algoritmo é dividido em etapas, sendo elas a detecção de "pontos de interesse" (pontos importantes para diferenciação da imagem), localização dos pontos de interesse, descritor das proximidades, assinalamento de orientação, descritor baseado na soma de respostas da transformação Haar [4] [11] e a comparação.

Na detecção dos pontos de interesse, são utilizados filtros de forma quadrada como uma aproximação de uma suavização gaussiana [13]. SURF usa um detector de blobs baseado na matriz hessiana para achar os pontos de interesse – pontos de máximo na diferença entre a imagem e sua suavização. Dado um ponto $p = (x, y)$ em uma imagem I, a matriz hessiana $H(p,\sigma)$, onde p é o ponto e σ é a escala, é definida da seguinte forma:

$$H(p,\sigma) = \begin{pmatrix} L_{xx}(p,\sigma) & L_{xy}(p,\sigma) \\ L_{yx}(p,\sigma) & L_{yy}(p,\sigma) \end{pmatrix} \quad (2)$$

Onde, $L_{xx}(p,\sigma)$ é a derivada segunda em relação a x, assim definindo os outros.

Pode-se usar várias suavizações gaussianas sequênciais, para dessa forma descobrir blobs de diferentes tamanhos.

O objetivo do descritor é prover uma descrição única e robusta da imagem descrevendo a distribuição de intensidade dos pixels dentro da vizinhança do ponto de interesse. A dimenção do descritor tem influência direta na complexidade e na robustez da aplicação do algoritmo.

Para extrair a orientação, é utilizada a transformada Haar para cada ponto de interesse nas direções x e y dentro de um raio de 6 * σ, onde σ é a escala onde o ponto de interesse for detectado. A resposta obtida terá um peso associado a função gaussiana no ponto de interesse. A orientação dominante é, então, estimada pelo cálculo da soma de todas as respostas dentro de uma dada janela de tamanho $\pi/3$. As respostas horizontais e verticais dentro da janela são somadas resultando em um vetor de orientação local. O tamanho da janela também deve ser escolhido com caltela, para se alcançar a devida robustez e resolução angular.

Então, basta comparar os descritores e verificar se há um casamento.

## 3. Resultados Obtidos

No primeiro bimestre, foi tido como principal objetivo aprender algoritmos de inteligência artificial e de visão que pudessem ser úteis no projeto, além de verificar a bibliografia disponível sobre algoritmos já utilizados nos desafios acima.

Redes Neurais foram estudadas por várias bibliografias de diversos autores como Norwig [10], Nelson [9] e Mehra [7] e implementadas algumas de suas variedades em C/C++, baseadas no livro de Adam [1]. Também foi lido sobre Maquina de Vetores Suporte no livro de Norwig [10]. Já os algoritmos de visão foram estudados, em parte, pelo livro de Szeliski [14].

Também foram analizados artigos disponíveis na internet de outros times sobre visão computacional e algoritmos utilizados no problema.

No segundo bimestre, foi utilizada uma biblioteca de redes neurais (feita em JAVA) chamada Encog, e dessa forma foi possível criar com mais facilidade uma interface para o treinamento da rede neural.

O objetivo do uso dessa rede é a segmentação, no qual dado os valores RGB de um pixel (valores estes que vão de 0 a 255) tem-se como resposta uma cor, i.e. branco (do gol). No fim do treinamento, é feita uma tabela de cores – para cada valor RGB, é associado um valor de uma cor (os valores RGB [0; 0; 0] são associados ao preto, no caso escolhido como cor de número 7). Usamos uma rede neural supervisionada multicamadas, função de transferência sigmoide, nas quais variamos diversas vezes o número de camadas de perceptrons nelas contidas, sendo os seguintes casos testados:

1. 2 camadas internas de 10 perceptrons;
2. 3 camadas internas de 10 perceptrons;
3. 4 camadas internas de 10 perceptrons;
4. 3 camadas internas de 20 perceptrons;
5. 3 camadas internas de 50 perceptrons;
6. 4 camadas internas de 30 perceptrons;
7. 4 camadas internas de 20 perceptrons;

Todas tem primeira camada com 3 perceptrons (valores RBG) e 8 perceptrons de saida (uma para cada cor).

Para cada configuração, fizemos várias tabelas (em torno de 5 a 10), sendo um processo bem trabalhoso. Mantivemos apenas algumas poucas tabelas, que foram julgadas úteis para os trabalhos posteriores.

Podemos ver nas figuras 3 e 4 o quão duas configurações diferentes aplicadas a mesma imagem podem se parecer. Mais na frente haverá a aplicação de nossa tabela atual, feita na configuração 4, em uma imagem com condições semelhantes.

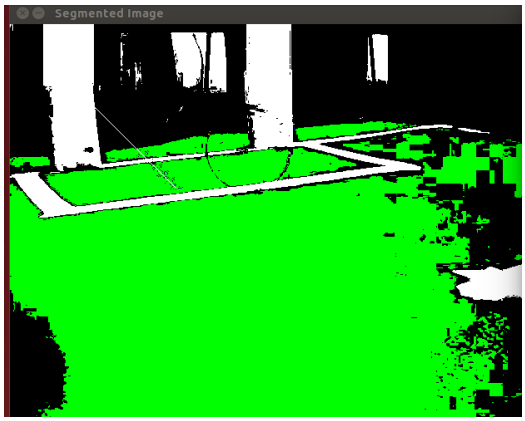


Figura 5. Melhor tabela feita na configuração 6.

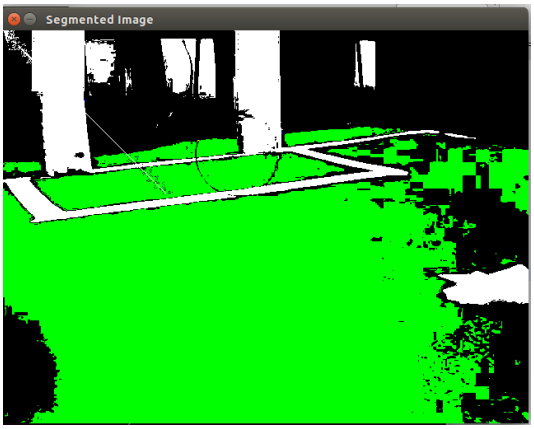


Figura 6. Melhor tabela feita na configuração 7.

Nessa interface, devemos seguir a sequência de treinamento de uma rede neural, que seria:

1. Criar exemplos para treinamento;
2. Treinar a rede;
3. Gerar tabela de cores.

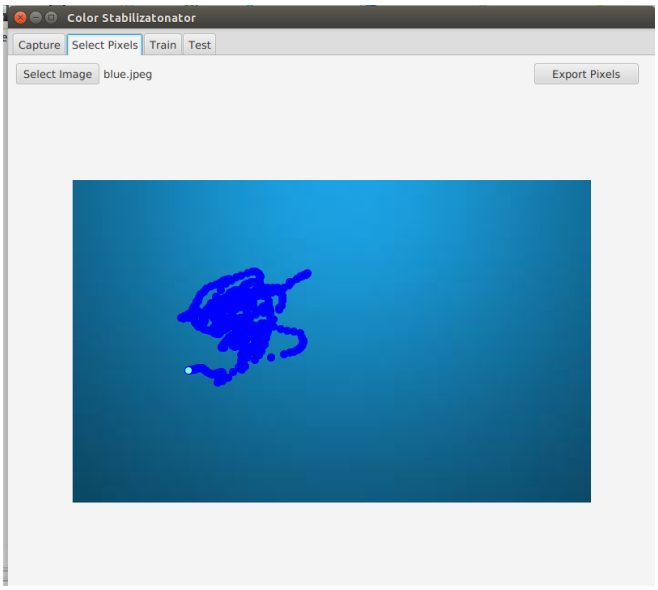


Figura 7. Interface criada (aba de desenvolvimento de exemplos).

Para gerar exemplos para o treinamento da rede neural, primeiro escolhe-se uma imagem pertinente para o treinamento e a colore com a ferramenta, com as cores disponíveis. Da forma que a imagem for colorida em cada pixel, o algoritmo tentará reproduzir. Logo, deve-se evitar colorir pixels com cor duvidosa ou colorir de forma errada. Por fim, basta exportar a imagem. Será feito um arquivo ".col" que terá os valores do pixel em RGB e o seu valor associado ao colorir. Um pixel por linha.

Caso haja algum exemplo que deva ser excluido, deve-se ir na pasta "Training Set" e apagá-lo.

As cores disponíveis para treinamento, e seus respectivos valores, são: verde (número 1); branco (número 2); laranja (número 3); rosa (número 4); azul (número 5); amarelo (número 6); preto (número 7); pixel indefinido (número 0).

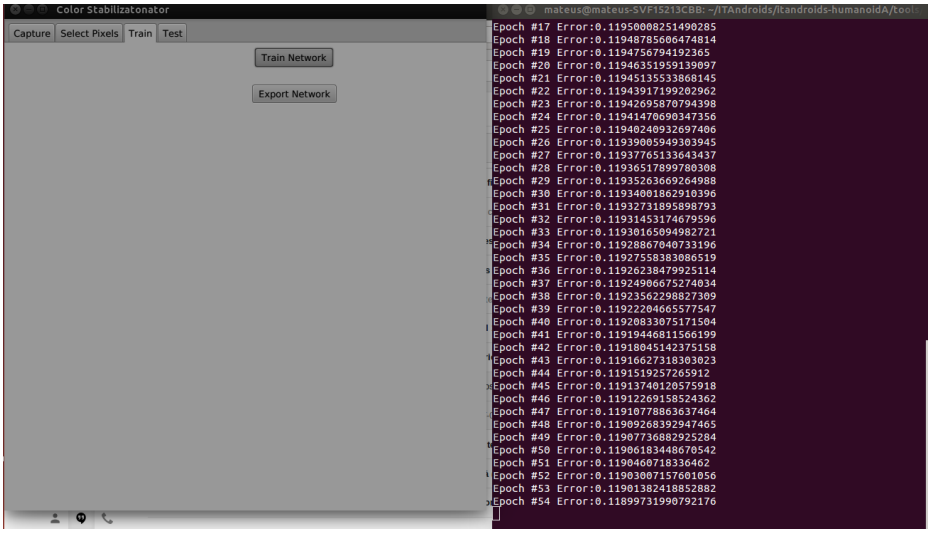


Figura 8. Do lado esquerdo, interface na aba de treinamento e de exportação de rede. Do lado direito, as iterações e seus respectivos erros.

Para treinar a rede usando os exemplos criados, basta apertar o botão treinar rede, e então serão realizadas as iterações do algoritmo. No código pode ser alterada a condição de parada (seja por número de iterações, seja por erro, ou ambas), o que não é muito prático.

Para gerar a tabela de cores, basta exportar a rede neural. Nesse ponto a rede neural terá como input cada valor RGB possível, ou seja, de 0 a 255 para cada valor resultando em $256^3$ valores de pixels diferentes. Dessa forma, teremos um acesso fácil às cores associadas a cada pixel da imagem (no momento da consulta é utilizado tabela hash).

Essa foi, sem dúvida, a tarefa que exigiu bastante tempo, visto que o número de pixels utilizados como amostra era grande, tornando as iterações demoradas. Para tabelas mais bem feitas, gastava-se, em geral, 1 hora gerando os exemplos da forma citada acima e mais 6 horas (no mínimo, dependendo do número de pixels de treino, podia chegar a mais de 10 horas, como foi o caso da melhor tabela de que dispomos).

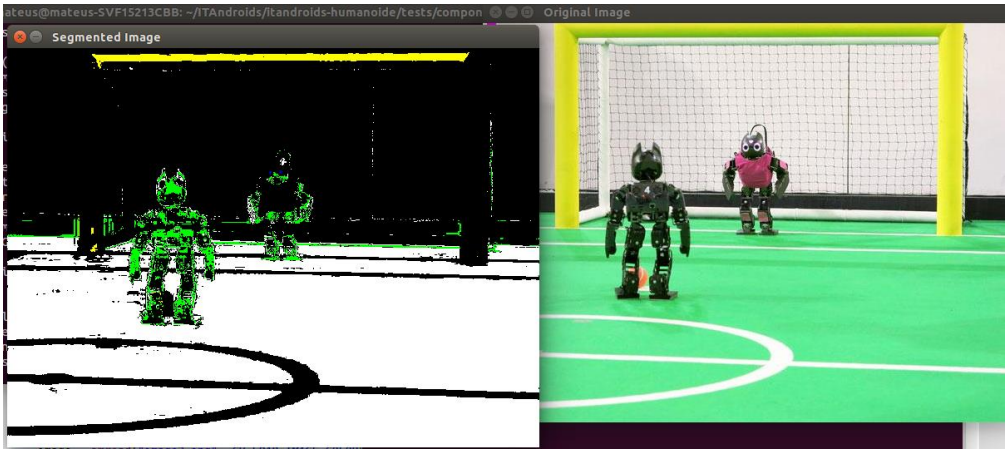


Figura 9. Tabela de cores não convergiu para o esperado.

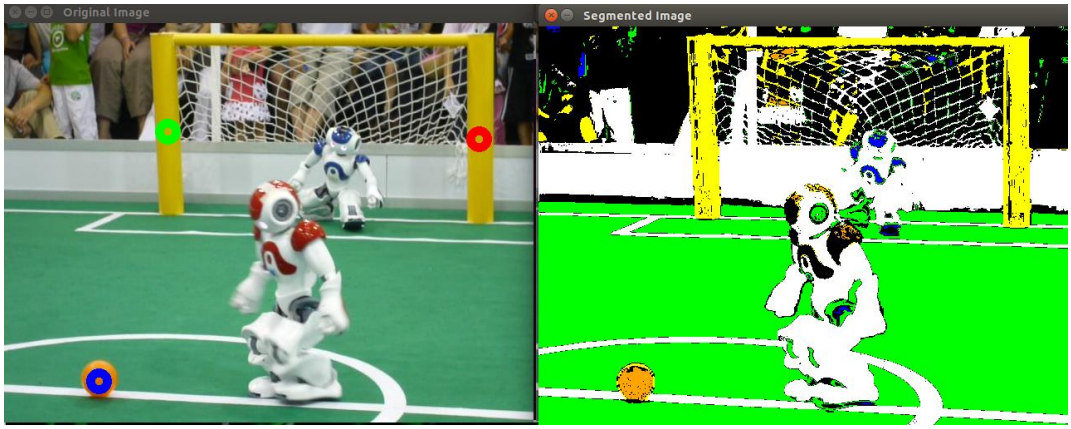


Figura 10. Tabela de cores e detecção feitos para objetivo antigo.


Figura 11. Imagem segmentada com uma das tabelas de cores feitas usando a configuração mais atual da Rede Neural.

Nesse mesmo bimestre, foi realizada uma adaptação no algoritmo de detecção de traves.

Para a detecção de traves, seguimos os seguintes passos:

1. Agrupamos os pixels de mesma cor;
2. Retiramos dados do agrupamento;
3. Aplicamos filtros nos agrupamentos.

Agrupando os pixels de mesma cor:

Para tanto, usamos uma estrutura de dados, no caso uma matriz, para agrupar pixels de mesma cor. Primeiramente, agrupamos pixels de uma mesma linha que são considerados próximos (cerca de 2 ou 4 pixels de distância) em uma estrutura chamada *RunLength* – existem *RunLengths* verticais e horizontais, porém, para o caso da trave, usamos apenas as horizontais. Em seguida, as *RunLengths* que estiverem se interceptando se as colocassem na mesma vertical, no caso das verticais, ou na horizontal, no caso das horizontais.

Exemplo: *RunLength* vertical 1 é paralela e próxima da *RunLength* 2. É verificado então se uma se interceptaria com a outra se ambas estivessem na mesma vertical.

Esses agrupamentos de *RunLengths* são chamados de *blobs*.

Retirando dados do grupamento e aplicando filtros:

Dados os *blobs*, retiramos informações importantes para uma análise correta de se aquele *blob* se trata de uma trave ou não.

Os dados que retiramos para identificação de trave branca são:

- Razão altura/largura: a trave não deve ser nem muito fina nem muito larga;

- Elevação: sabendo a altura do *blob* na imagem e o ângulo de elevação da câmera em relação ao solo, conseguimos saber qual a elevação do mesmo e podemos verificar se o possível candidato a trave não estaria "voando".
- Razão verde e branco abaixo do *blob*: abaixo do *blob*, para que este seja um candidato razoável a uma trave, deve-se ter uma maioria de pixels verdes e brancos, visto que estes estão contidos no campo.
- Razão não-verde e branco ao lado do blob: para que possamos considerar o *blob* um candidato razoável, devemos analizar se ao seu entorno não tem muito verde e branco, pois isso aumentaria o ruído na formação do blob (no caso do branco) e poderia ser confundido com uma linha de campo, em um caso que a elevação foi calculada errada (caso posto como garantia, mas que não foi verificado em testes.

Por fim, caso o *blob* passe nesses testes, ele é considerado uma trave.

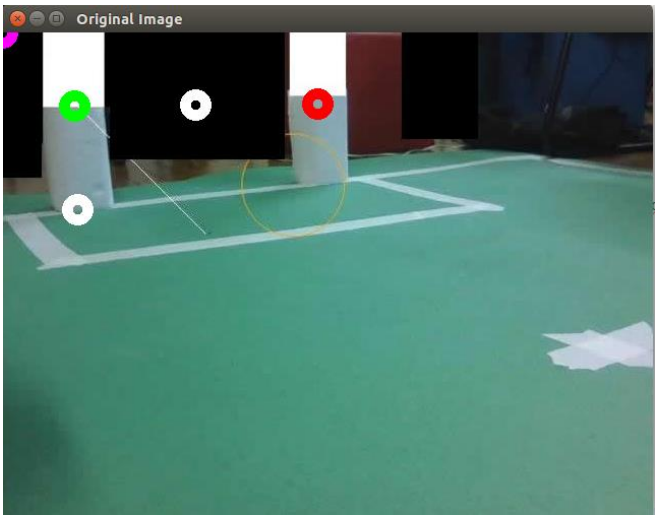


Figura 12. Detecção de trave da mesma imagem da Figura 3. Foi posto partes pretas onde havia uma parede branca. Posteriormente, esse problema foi resolvido com uma tabela mais rigorosa feita para a competição.

Ainda no segundo bimestre, foram ajustados de forma empirica (não obrigatoriamente estes parâmetros estão otimizados) os seguintes parâmetros para a detecção de linhas: distância mínima e máxima (em pixels) para unir duas *RunLengths;* passo horizontal e vertical em que a imagem é percorrida (se o pixel x[0] é lido na vertical, se o passo for de 2 pixels, o próximo a ser lido é o pixel x[2] da mesma vertical, por exemplo).

Essa fase se demonstrou difícil, embora menos trabalhosa, pois exigiu familiaridade com o código que dispunhamos. Foram testadas várias formas de se detectar as traves sem utilizar algoritmos genéricos de visão (algoritmos que podem ser utilizados para a detecção de qualquer tipo de objeto), visto que dessa forma se economizaria tempo de processamento.

Uma dificuldade que apareceu nessa fase também foi o fato de que as matrizes de transformações 2D-3D e, consequentemente, 3D-2D não estavam corretas para o modelo do robô. Com essa matriz podemos mudar o referencial da câmera do robô para o referêncial do campo, sendo importante também para o cálculo de elevação do gol. Tive, então, que corrigir essa parte do código, o que me tirou um tempo, visto que está parte se tratava de outro módulo do código com que não havia trabalhado.

Na competição, a detecção de traves detectava a grande maioria dos casos em que a trave se encontrava no mesmo campo que o robô – usando fotos da câmera do robô, ele teve 100% de sucesso -, porém não encontrava em quase nenhum caso que estava no campo oposto.

No segundo bimestre, foi ajustado o modelo matemático utilizado para calculo de distâncias da bola em relação ao robô em função do raio do *blob.* Para tanto, usamos uma exponencial ($ae^{br}$, onde "a" e "b" são parâmetros a se otimizar e "r" é o raio do *blob).* Tal modelo foi utilizado com base em outros times como o Austin Villa. Para tanto, foram tiradas diversas fotos do blob com suas respectivas distâncias reais e raio do *blob* e usando o software MATLAB, utilizando regressão linear (após linearização), ajustamos os parâmetros de tal modelo.

Tal tarefa não foi difícil, pois solicitou somente conceitos básicos de álgebra linear e teve o auxilio de uma ferramenta matemática muito eficaz.

No quarto bimestre foi feito um estudo mais aprofundado de transformadas Harr, com o objetivo de aplicá-la no problema de detecção de padrões, para assim identificar a bola. O algoritmo a ser usado seria o Viola-Jones[15], comumente utilizado para detecção de faces. Para tanto, alguns outros artigos de aplicações foram úteis[8] [12] [16].

Porém, houveram muitos problemas no treinamento do detector em cascata. Tais problemas foram de natureza prática, como problemas na instalação e manipulação de paths para realizar o treinamento.

Devido a isso, ocorreu uma adaptação do plano, para utilizar o algoritmo SURF, muito embora hajam artigos que comprovam que este não tem uma complexidade tão boa quanto o primeiro. Porém, é aplicável ao problema. O software utilizado para implementação foi o MATLAB.

Esse trabalho foi feito no quinto bimestre. A bola escolhida para o trabalho, como basta ser uma própria de futebol profissional, foi a bola oficial da Eurocopa de 2016.

Primeiramente, foram detectados os pontos de interesse de ambas as imagens (uma da bola e outra de uma imagem com a bola). As imagens, para melhor tratamento, foram convertidas para escala Gray.

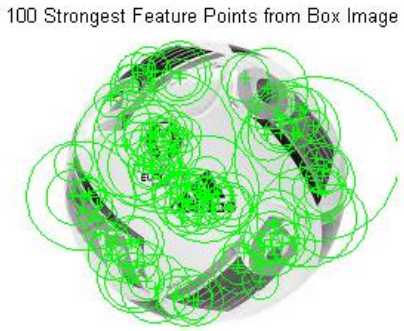


Figura 13. Pontos de interesse da bola.

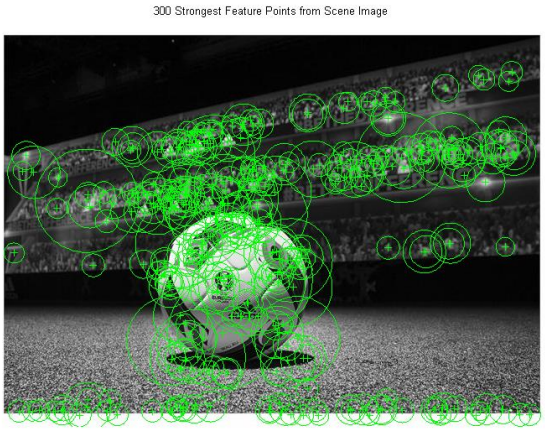


Figura 14. Detecção dos pontos de interesse da imagem.

Então, houve um casamento dos pontos de interesse, detectando, assim a bola na imagem.

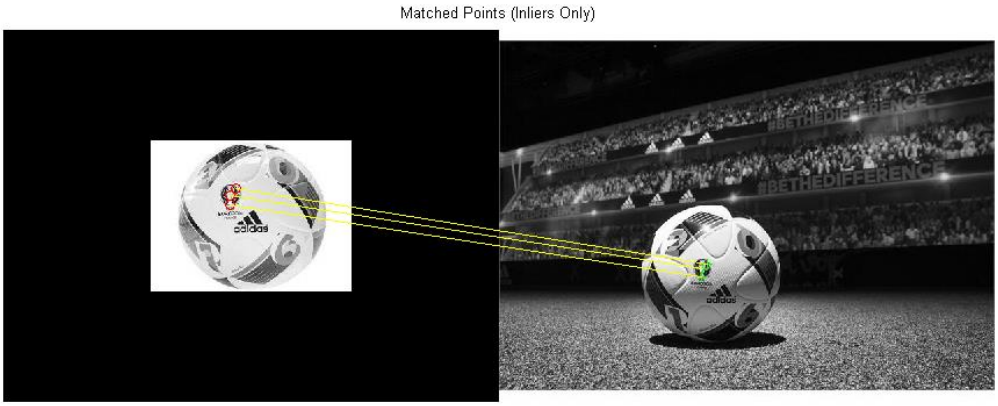


Figura 15. Casamento de pontos de interesse da imagem.

Então, bastou analizar a orientação da imagem.

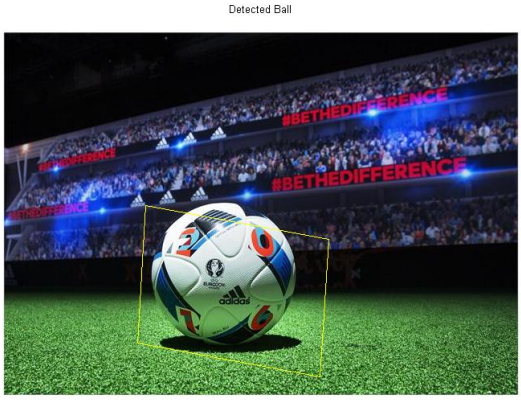


Figura 16. Detecção da bola.

Tal algoritmo, o qual é uma forma mais rápida e aproximada do algoritmo SIFT [6], para um desempenho mais completo, necessita de quatro imagens diferentes da bola, de diferentes angulos. Ao aplicar o algoritmo para tais imagens, tomava-se o resultado mais plausível de posição e orientação, tornando fácil a posterior filtragem.
Isso ocorre pois a bola pode ter diferentes pontos de interesse dependendo da angulação. Foi notado, por exemplo, que se a bola estivesse rolada de 180 graus, a detecção poderia ser falha, o que não ocorreria se houvesse outra imagem da mesma bola e seus pontos de interesse de outra angulação.

## 4. Conclusões

Nesse trabalho, foi analizada a eficácia de algoritmos não generalizados na área de visão computacional aplicada à robótica, principalmente com relação ao problema da detecção da trave branca. Quando temos recursos limitados, em um sistema embarcado, todo o processamento a menos, mantendo uma qualidade aceitável de detecção, é importante. Isso foi feito com sucesso no caso da trave, muito embora ainda haja muito a melhorar tanto na eficiência do código quanto em sua velocidade.

Foi verificada a validade de certos teoremas da área de rede neural [5], quando a aplicamos ao problema de segmentação, problema base para um processamento rápido de visão – muito embora afete a qualidade da detecção.

## 5. Agradecimentos

Agradeço ao professor doutor Paulo Marcelo Tasinaffo, meu orientador, e ao professor mestre Marcos Ricardo de Omena Albuquerque Máximo, co-orientador, por toda a orientação e apoio nessa primeira parte de meu projeto.

Agradeço também ao Instituto Tecnológico de Aeronáutica por parte da formação básica necessária para o desenvolvimento dessa Iniciação Ciêntífica.

Agradeço ao CNPQ, pelo apoio financeiro, o qual incentiva cada vez mais alunos a adentrarem nas mais diversas áreas de pesquisa.

**6. Referências**

[1] Adam, B., 1992, "Neural Networks in C++: an object-oriented Framework for Building Connectionist Systems", John Wiley & Sons, Inc. New York, NY, USA.

[2] Barrett, S., Genter, K., He, Y., Hester, T., Khandelwal, P., Menashe, J. and Stone, P., The 2012 UT Austin Villa Code Release.

[3] Bay, H., 2006, "Speeded Up Robust Features. Computer Vision", ECCV 2006: 9th European Conference on Computer Vision, Graz, Austria, Proceedings, Part I. Springer Berlin Heidelberg.

[4] Charles, K. C., 1992, "An Introduction to Wavelets", Academic Press, San Diego.

[5] Chauvin, Y. and Rumelhard, D. E., 1995, "Backpropagation: theory, architectures, and applications", L. Erlbaum Associates Inc., Hillsdale, NJ.

[6] Lowe, D. G., 1999, "Object recognition from local scale-invariant features", Proceedings of the International Conference on Computer Vision. pp. 1150–1157.

[7] Mehra, P., Wah, B.W., 1992, "Artificial neural networks: concepts and theory", New York, NY: IEEE Computer Society, 667 p.

[8] Mikolajczyk, K. and Schmid, C., 2004, "Scale and affine invariant interest point detectors", *IJCV*, 60(1):63.86.

[9] Nelson, M. M. and Illingworth, W.T., 1991, "A practical guide to neural nets", Reading, MA: Addison-Wesley, 344p.

[10] Norwig, P. and Russell, S., 2013, "Artificial Intelligence", 3rd ed. Elsevier.

[11] Ruch, D. K. and Van Fleet, P. J., 2009, "Wavelet Theory: An Elementary Approach with Applications", John Wiley & Sons.

[12] Schneiderman, H. and Kanade, T., 2004, "Object detection using the statistics of parts", IJCV, 56(3):151.177.

[13] Shapiro, L. G. and Stockman, G. C., 2001, "Computer Vision", Prentice Hall, pp. 137-150.

[14] Szeliski, R., 2010, "Computer Vision: Algorithms and Applications", Springer.

[15] Viola, P. and Jones, M., 2001, "Rapid object detection using a boosted cascade of simple features", International Journal of Computer Vision.

[16] Viola, P., Jones, M. J. and Snow, D., 2003, "Detecting pedestrians using patterns of motion and appearance", The 9th ICCV, Nice, France, vol. 1, pp. 734-741.