CENTRO UNIVERSITÁRIO FEI

VINICIUS NICASSIO FERREIRA

UMA ARQUITETURA COM PERSISTÊNCIA VISUAL PARA O RASTREAMENTO DE OBJETOS NO DOMÍNIO DOS ROBÔS MÓVEIS AUTÔNOMOS

São Bernardo do Campo
São Paulo
2018

VINICIUS NICASSIO FERREIRA

UMA ARQUITETURA COM PERSISTÊNCIA VISUAL PARA O RASTREAMENTO DE OBJETOS NO DOMÍNIO DOS ROBÔS MÓVEIS AUTÔNOMOS

Dissertação de Mestrado apresentada ao Centro Universitário FEI para obtenção do título de Mestre em Engenharia Elétrica. Orientado pelo Prof. Dr. Reinaldo Augusto da Costa Bianchi.

São Bernardo do Campo
São Paulo
2018

Nicassio Ferreira, Vinicius.
Uma arquitetura com persistência visual para o rastreamento de objetos no domínio dos robôs móveis autônomos / Vinicius Nicassio Ferreira. São Bernardo do Campo, 2018.
95 p. : il.

Dissertação - Centro Universitário FEI.
Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi.

1. localização egocêntrica. 2. visão computacional. 3. detecções múltiplas. 4. rastreamento de objetos. 5. filtro de kalman. I. Augusto da Costa Bianchi, Reinaldo, orient. II. Título.

Elaborada pelo sistema de geração automática de ficha catalográfica da FEI com os dados fornecidos pelo(a) autor(a).

centro universitário FEI

APRESENTAÇÃO DE DISSERTAÇÃO
ATA DA BANCA EXAMINADORA

Programa de Pós-Graduação Stricto Sensu em Engenharia Elétrica

Mestrado

PGE-10

Aluno: Vinicius Nicassio Ferreira **Matrícula:** 116126-4

Título do Trabalho: Uma arquitetura com persistência visual para o rastreamento de objetos no domínio dos robôs móveis autônomos.

Área de Concentração: Inteligência Artificial Aplicada à Automação e Robótica

Orientador: Prof. Dr. Reinaldo Augusto da Costa Bianchi

Data da realização da defesa: 26/03/2018

ORIGINAL ASSINADA

Avaliação da Banca Examinadora:

São Bernardo do Campo, 26 / 03 / 2018.

MEMBROS DA BANCA EXAMINADORA	
Prof. Dr. Reinaldo Augusto da Costa Bianchi	Ass.: _____
Prof. Dr. Flavio Tonidandel	Ass.: _____
Prof.ª Dr.ª Anna Helena Reali Costa	Ass.: _____

A Banca Julgadora acima-assinada atribuiu ao aluno o seguinte resultado:

APROVADO ☒ REPROVADO ☐

VERSÃO FINAL DA DISSERTAÇÃO

APROVO A VERSÃO FINAL DA DISSERTAÇÃO EM QUE FORAM INCLUÍDAS AS RECOMENDAÇÕES DA BANCA EXAMINADORA

_____

Aprovação do Coordenador do Programa de Pós-graduação

_____

Prof. Dr. Carlos Eduardo Thomaz

À família e amigos.

AGRADECIMENTOS

A Deus.

“Pensar é o trabalho mais difícil que existe. Talvez por isso tão poucos se dediquem a ele.”
Henry Ford

RESUMO

Na atualidade, existe uma grande variedade de robôs autônomos utilizados em diversas áreas, seja para cumprir procedimentos de alto risco ou trazer entretenimento. Os robôs interagem com os mais diversos tipos de objetos, e para que essa seja uma boa interação, existe a necessidade de localizá-los no ambiente. Um exemplo é a liga *humanoid* da *RoboCup*, uma competição com robôs autônomos que devem jogar futebol. Para realizar tal tarefa, eles precisam ser capazes de localizar diversos objetos, como bola, robôs e *landmarks*, através de uma câmera. Mesmo realizando seus movimentos, o robô ainda deve ser capaz de manter a localização destes objetos.

O objetivo deste trabalho é definir e implementar um sistema de rastreamento de objetos para o robô humanoide da Fundação Educacional Inaciana (FEI), considerando os requisitos da competição que a equipe participa. Para isso, foi remodelada a estrutura do sistema de visão, pois o sistema anterior estava defasado e não realizava a detecção de robôs, somente da bola. Utilizando uma arquitetura de *threads*, a visão passa a detectar simultaneamente múltiplos objetos e transmitir essa informação para o sistema de rastreamento. Este sistema gera um mapa egocêntrico e, utilizando o filtro de Kalman, realiza o rastreamento do objeto neste mapa, sempre levando em consideração os movimentos do robô e do objeto para fazer a predição da posição, velocidade e aceleração deste objeto. Com a implementação deste novo sistema, espera-se ter uma melhora de desempenho quanto à tomada de decisão, pois as informações de todos os objetos sempre estarão disponíveis, e mesmo quando possivelmente ocorrer uma falha na detecção, ainda haverá a predição.

Palavras-chave: Localização egocêntrica, visão computacional, detecções múltiplas, rastreamento de objetos, filtro de Kalman

ABSTRACT

At present, there is a wide variety of autonomous robots used in many areas, either to comply with high risk procedures or to bring entertainment. Robots interact with the most diverse types of objects, and for this to be a good interaction, there is a need to localize them in the environment. An example is the RoboCup's humanoid league, a competition with autonomous robots that must play football. To accomplish such a task, they need to be able to locate various objects, such as the ball, robots and landmarks, through a camera. Even performing their movements, the robot should still be able to maintain the location of these objects.

The objective of this project is to define and implement an object tracking system for the humanoid robot of the Educational Foundation of the Inacian (FEI), considering the requirements of the competition that the team participates. Consequently, the structure of the vision system was redesigned, since the previous system was outdated and did not perform the detection of robots, only of the ball. Using a thread architecture, the view simultaneously detects multiple objects and transmits this information to the tracking system. This system generates an egocentric map and, using the Kalman filter, it traces the object on this map, always taking into account the movements of the robot and the object to predict the position, speed and acceleration of this object. With the implementation of this new system, it is expected to have an improvement in decision-making and performance because the information from all objects will always be available, and even when a detection failure may occur, there will still be a prediction.

Keywords: egocentric location; computational vision; multiple detection; object tracking; Kalman filter

LISTA DE ILUSTRAÇÕES

LISTA DE TABELAS

LISTA DE ALGORITMOS

LISTA DE ABREVIATURAS

AGV	Veículo Guiado Autonomamente, do inglês *Automated Guided Vehicle*.
FEI	Fundação Educacional Inaciana.
FIFA	Federação Internacional de Associação de Futebol, do francês *Fédération Internationale de Football Association*.
FPS	Quadros por segundo, do inglês *frames per second*.
HOG	Histograma de Gradientes Orientados.
HSI	Matiz, saturação e intensidade, do inglês *Hue*, *Saturation* e *Intensity*.
MSL	Liga de tamanho médio, do inglês *middle size league*.
NUC	Próxima Unidade de Computação, do inglês *Next Unit of Computing*.
RGB	Vermelho, verde e azul, do inglês *Red*, *Green* e *Blue*.
RNA	Rede Neural Artificial.
RNP	Rede Neural Profunda.
SSL	Liga de tamanho pequeno, do inglês *small size league*.

LISTA DE SÍMBOLOS

α	Ângulo de rotação em X.
β	Ângulo de rotação em Y.
C	Matriz de translação.
c_h	Matriz que representa um ponto no mundo.
G	Matriz de translação.
γ	Ângulo de rotação em Z.
H	Representação do canal *Hue* em uma imagem..
I	Representação do canal *Intensity* em uma imagem.
λ	Distância focal.
P	Matriz de perspectiva.
p	Valor da intensidade do pixel.
pos_x	Posição X do objeto no mundo.
pos_y	Posição Y do objeto no mundo.
R	Matriz de rotação.
S	Representação do canal *Saturation* em uma imagem.
θ	Representa o angulo do sistema de cor Matiz, saturação e intensidade, do inglês *Hue*, *Saturation* e *Intensity* (HSI).
w_h	Matriz que representa um ponto na imagem.
x	Matriz de representação de estado.

SUMÁRIO

1 INTRODUÇÃO

A cada dia, o número de sistemas automatizados ou robôs autônomos vem crescendo, não apenas em indústrias, mas também em residências. A tendência é que, devido ao avanço da tecnologia e às novas técnicas que estão surgindo, isso aumente. Uma técnica que se tornou possível graças ao avanço que obtivemos no processamento de placas de vídeo é a Rede Neural Profunda (RNP). Nos dias de hoje, um sistema de segurança simples, com algumas câmeras e acesso à rede mundial de computadores, pode reconhecer pessoas suspeitas e segui-las por um determinado ambiente, analisando as interações que o indivíduo faz com o local. As mesmas técnicas podem ser aplicadas em um Veículo Guiado Autonomamente, do inglês Veículo Guiado Autonomamente, do inglês *Automated Guided Vehicle* (AGV), para prever as interações que esse robô pode ter, tanto com os objetos quanto com as pessoas que o cercam. Quanto maior o número de objetos que estes sistemas precisam detectar e localizar em um mapa global, maior o custo computacional. A necessidade de rastrear múltiplos objetos simultaneamente e predizer as possíveis interações entre eles é um dos maiores problemas dos robôs móveis.

A *RoboCup* é uma competição de robótica que tem várias categorias, sendo que a FEI participa de duas modalidades: a *RoboCupSoccer* e a *RoboCup@Home* (GERNDT et al., 2015). A *RoboCupSoccer* é dividida em cinco subcategorias: Humanoide, *Standard Platform*, Liga de tamanho médio, do inglês *middle size league* (MSL), Liga de tamanho pequeno, do inglês *small size league* (SSL), e simulação. O objetivo da *RoboCup* é desenvolver robôs autônomos que joguem partidas de futebol seguindo tanto as regras da Federação Internacional de Associação de Futebol, do francês *Fédération Internationale de Football Association* (FIFA) (2017) quanto as regras da própria *RoboCup*. Atualmente, a competição está em seu vigésimo ano e possui cerca de 22 equipes participantes apenas na categoria humanoide, sendo a FEI uma destas equipes.

Na *RoboCup* humanoide, o principal meio utilizado para identificar o mundo é a visão do robô. Os robôs humanoides mostrados na figura 1 são atualmente usados pela equipe RoboFEI na categoria humanoide. Eles têm processamento autônomo, ou seja, cada robô utiliza um computador, que é responsável por processar desde a imagem vinda da câmera Vermelho, verde e azul, do inglês *Red*, *Green* e *Blue* (RGB) até o controle dos motores.

Um dos problemas enfrentados na construção de uma equipe de robôs autônomos para atuar na *RoboCup* Humanoide é a detecção de objetos, devido ao tamanho do campo e às limitações de alcance da visão do robô. Nesta liga é comum os robôs executarem várias buscas pelo objeto. Para isso, eles ficam mexendo a câmera através dos servomotores, a fim de localizar o objeto desejado. Porém, em um jogo com quatro jogadores por time mais a bola, chegamos à possibilidade de nove objetos dinâmicos interagindo dentro do campo. Estes objetos precisam ser localizados. A busca cega se torna ineficiente neste ambiente, que é muito grande e dinâmico, pois o robô passa muito tempo realizando buscas para localizar objetos já detectados anteriormente. A solução apresentada neste trabalho é a realização do rastreamento dos objetos, pois assim o robô executa a busca para detectar os objetos apenas uma vez. Quando o sistema

Figura 1 – Robô humanoide desenvolvido e construído na FEI com base no robô Darwin-OP.

Fonte: Do autor.

de visão deixar de detectá-lo, seja por uma oclusão ou porque o robô mudou seu foco de atenção para procurar um novo objeto, o rastreamento é usado para manter atualizada a informação que diz respeito ao primeiro objeto, e quando for necessário saber a posição de um objeto que já tenha sido detectado, a informação dada pelo rastreamento pode ser usada como heurística para uma nova detecção.

Para que se possa realizar o rastreamento de objetos, a obtenção de informações será realizada através da câmera. Utilizando classificadores para fazer a análise das imagens, é possível detectar todos os objetos e inferir as posições destes objetos no mundo. Com todas essas informações, é possível iniciar o rastreamento dos objetos detectados e também estimar informações intrínsecas, como a velocidade de movimento ou a aceleração de cada um deles. Às informações intrínsecas será necessário aliar a movimentação do próprio agente, pois assim conseguiremos utilizá-las para atualizar a posição dos objetos com relação aos movimentos executados pelo robô. Para isso, o modelo que descreve o movimento dos objetos contempla tanto sua movimentação quanto o movimento executado pelo próprio robô. A informação gerada pelo rastreamento é usada para a predição, ou seja, para prever a posição atual do objeto, tendo como base as informações anteriores. Porém, isso gera um erro acumulativo com o passar do tempo, e assim, gradativamente essa informação perderá seu valor, até que uma nova detecção possa ser realizada e sua confiabilidade volte a aumentar ou que o objeto venha a ser dado como perdido.

1.1 OBJETIVO

O objetivo deste trabalho é definir e implementar um sistema de rastreamento de objetos para o robô humanoide da FEI. Esse sistema será testado nas competições em que a equipe participa, como a *RoboCup*.

Para que esse objetivo seja atingido, é necessário reestruturar o sistema de visão existente nos robôs, pois atualmente esse sistema realiza apenas uma detecção por vez: quando se tem a necessidade de detectar outros objetos, uma nova varredura é realizada e a informação do objeto anterior é descartada. Com a reestruturação do sistema de visão, os robôs serão capazes de detectar múltiplos objetos simultaneamente.

1.2 METODOLOGIA

Com o sistema implementado, é possível usar a informação do rastreamento para ter uma diminuição no tempo de busca quanto à detecção dos objetos, pois o sistema estará sempre monitorando o objeto e quando perdê-lo, poderá iniciar a busca em locais mais prováveis. Utilizando esse mesmo conceito, este trabalho pode ser expandido para outros robôs móveis que necessitam localizar objetos dinâmicos ou estáticos, sem a necessidade de mapear completamente o domínio ou se localizar e assim, ter um custo de processamento menor, com uma confiabilidade maior do que usando o posicionamento global.

Para descrever esse trabalho em mais detalhes, uma breve apresentação da teoria que foi usada para solucionar o problema em questão é apresentada, além de alguns trabalhos relacionados ao proposto, seguidos de uma explicação mais detalhada da proposta, finalizando com experimentos e conclusão.

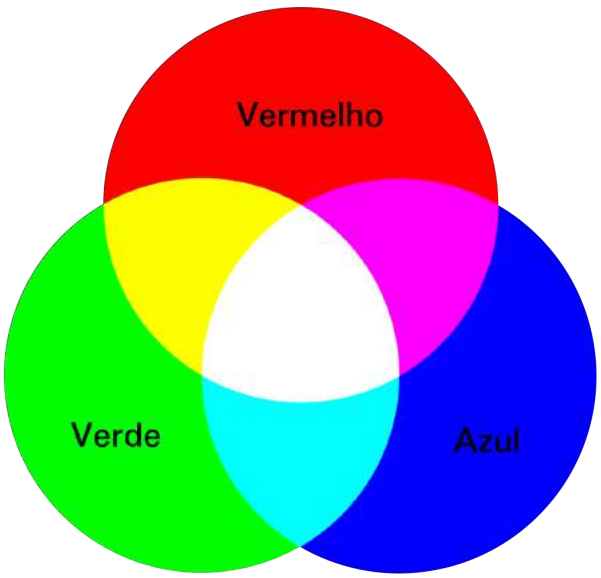
2 TEORIA

Neste capítulo serão apresentadas algumas técnicas a serem utilizadas para a detecção de objetos em uma imagem e para a realização do rastreamento destes objetos.

2.1 FUNDAMENTOS CORES

De acordo com o experimento de Sir. Isaac Newton, o espectro de cores é dividido nas cores violeta, azul, verde, amarelo, laranja e vermelho. Definir a cor de um objeto é uma das formas mais simples e rápidas para sua classificação. Para a detecção da luz, a visão humana tem cerca de 6,5 milhões de cones, dos quais 65% são sensíveis a luz vermelha, 33% são sensíveis a luz verde e apenas 2% são sensíveis a luz azul (porém, os cones azuis são mais sensíveis se comparados aos outros). Assim, nosso cérebro processa todas as cores como uma combinação destas cores primárias (GONZALEZ; WOODS, s.d.). A figura 2 mostra esse sistema de cores e suas combinações.

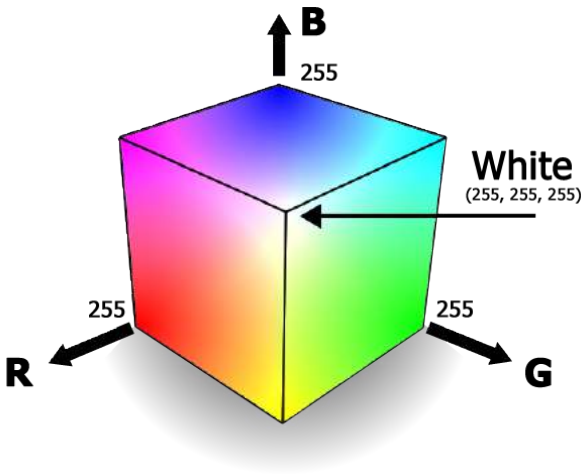
Figura 2 – Cores primárias e suas respectivas combinações.



Fonte: Do autor.

O sistema de cores RGB se baseia no sistema cartesiano de coordenadas, sendo que cada eixo representa as cores vermelho, azul e verde. Esse sistema é mostrado na figura 3. Neste caso, foi considerado que o valor máximo de intensidade da cor é 255. Esse valor vem da representação usual da intensidade de uma cor em um pixel, que é um conjunto de oito bits ou um byte, que se consegue representar $2^8 = 256$ valores. Cada eixo do plano representa uma das cores do sistema, e quando se realiza a composição dessas cores, é possível representar grande parte do espectro de cores (GONZALEZ; WOODS, s.d.).

Figura 3 – Espaço de cores RGB.



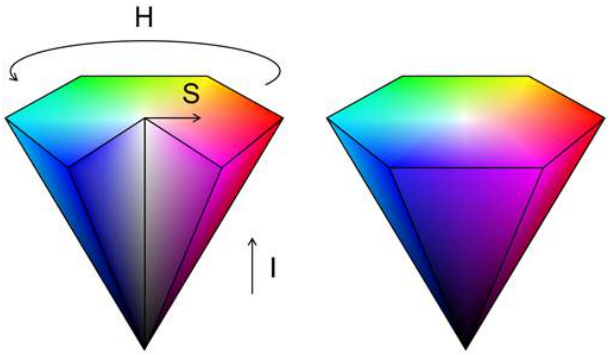
Fonte: Cunha, 2016.[1]

2.2 MODELO HSI

A técnica de segmentação de cores pode ser muito útil quando usada para detecção de objetos que possuem cores características, como o azul no céu ou o verde de uma planície. A principal dificuldade que se tem em relação ao sistema RGB é a sensibilidade à iluminação em ambientes em que esta se apresenta de forma não homogênea ou que possa variar durante o tempo. No modelo HSI, a técnica de segmentação pode ficar mais robusta com relação à variação de iluminação.

O espaço de cores HSI tem uma representação cônica, diferente da representação cúbica do RGB, e é mostrado na figura 4.

Figura 4 – Representação do sistema de cores HSI.



Fonte: Alex, 2016.[2]

[1]CUNHA, André Luiz. **Introdução à Visão Computacional**. Fev. 2016. Disponível em: <http://www.stt.eesc.usp.br/andre/palestras/CEFET-MG/01_Intro.html>.

[2]ALEX, Gorkov. **Sobre espaços de cor**. Fev. 2016. Disponível em: <https://habrahabr.ru/post/181580/>.

Para a conversão das cores de um modelo RGB para HSI se pode usar as seguintes equações. Para o cálculo de intensidade da matiz, a equação (1) deve ser aplicada:

$$H = \begin{cases} \theta & B \leq G \\ 360 - \theta & B > G \end{cases}, \tag{1}$$

sendo o cálculo de θ mostrado na equação (2):

$$\theta = \cos^{-1}\left\{\frac{\frac{1}{2}\cdot[(R-G)+(R-B)]}{\sqrt{(R-G)^2+(R-B)\cdot(G-B)}}\right\}. \tag{2}$$

Para o cálculo da saturação a equação (3) deve ser aplicada:

$$S = 1 - \frac{3}{R+G+B}\cdot mín(R,G,B). \tag{3}$$

E para o cálculo de intensidade a equação (4) deve ser aplicada:

$$I = \frac{1}{3}\cdot(R+G+B). \tag{4}$$

2.3 SEGMENTAÇÃO DE CORES

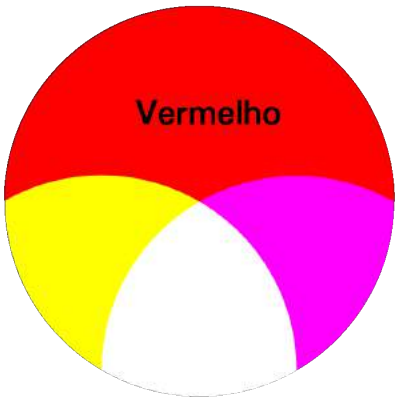
O processo de segmentação consiste em particionar uma imagem em diversas regiões para que a região de interesse venha a ser ressaltada enquanto as demais são descartadas. No caso da segmentação por cor, consiste na ressalva da parte na imagem que contenha o objeto da cor desejada. Por exemplo, considerando que na figura 2 se quer segmentar a cor vermelha, pode-se fazer de duas formas diferentes: a primeira delas é selecionar regiões que contenham qualquer intensidade de vermelho. Isso incluiria cores que são derivações do vermelho, como o amarelo, magenta e o branco. Outra forma é segmentar regiões que contenham exclusivamente a cor vermelha em qualquer intensidade. Essas duas formas de segmentação são mostradas na figura 5 (GONZALEZ; WOODS, s.d.).

A segmentação realizada neste exemplo foi feita no domínio RGB. Entretanto, para se fazer a descrição de uma cor, dificilmente esse domínio é utilizado, pois ele sofre muito com a iluminação dos ambientes. Quando as pessoas observam uma cor, usualmente a descrevem com base na matiz, saturação e brilho dessa cor. Por este motivo, o modelo HSI tem sido a ferramenta ideal para o processamento de cores em uma imagem (GONZALEZ; WOODS, s.d.).

Todas as informações obtidas com estas técnicas são relacionadas à imagem. Quando é necessário realizar o rastreamento do objeto no próprio ambiente da imagem, esta técnica é suficiente. Porém, quando se quer obter a informação do objeto no mundo, é necessário converter a informação obtida de coordenadas em *pixels* da câmera para a posição cartesiana no mundo. Para a realização dessa tarefa, utiliza-se a modelagem geométrica de câmera.

Figura 5 – Na figura 5a foi feita uma segmentação da cor vermelha e suas derivações, e na 5b exclusivamente para a cor vermelha.

(a)



Fonte: Do autor.

(b)

Fonte: Do autor.

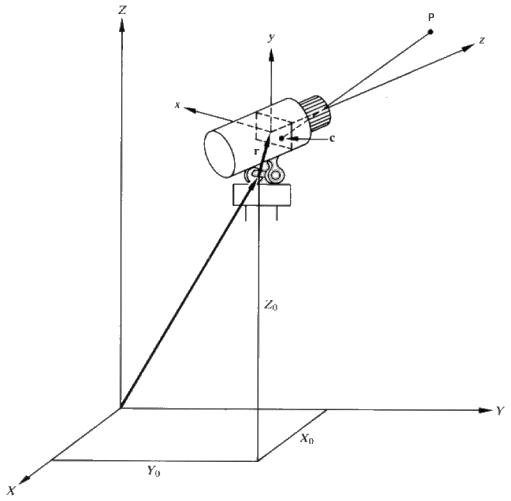
2.4 MODELO GEOMÉTRICO CÂMERA

A superfície de geração de imagem de uma câmera é normalmente retangular, diferentemente da visão humana, que tem sua imagem gerada em uma superfície esférica. Contudo, tanto as câmeras quanto a visão humana geram uma imagem espacialmente discretizada. Para modelar isso, parâmetros intrínsecos e extrínsecos são usados na calibração geométrica das câmeras (RODRIGUES, 2016).

Imagine uma câmera posicionada em um local qualquer no mundo, direcionada para um ponto P com coordenadas (X,Y,Z). Esse ponto tem um correspondente na imagem gerada pela câmera (x,y). Para se fazer essa correspondência de maneira mais fácil, um novo sistema de coordenadas é gerado na posição do orifício da câmera, o novo sistema (X_C,Y_C,Z_C). Na literatura é usual representar os pontos no ambiente com letras maiúsculas e os pontos de projeções na imagem com letras minúsculas (FORSYTH; PONCE, 2002). Na figura 6 está a descrição do ambiente proposto.

[3]RODRIGUES, Paulo Sérgio Silva. Calibração de câmeras. Apresentação de Slides. Centro Universitário FEI, fev. 2016. Disponível em: <http://www.fei.edu.br/~psergio/VisaoComputacional/Aula%206%20Calibra%C3%A7%C3%A3o%20de%20C%C3%A2meras.ppt>.

Figura 6 – Sistema de coordenadas usado na câmera.



Fonte: Adaptado de Rodrigues, 2016.[3]

Considerando que a projeção do ponto P na imagem pode ser representada pela matriz (5a), o ponto no mundo pela matriz (5b) e que k é um valor constante diferente de zero:

$$w_h = \begin{bmatrix} kx \\ ky \\ kz \\ k \end{bmatrix}, \qquad \text{(5a)} \qquad c_h = \begin{bmatrix} kX \\ kY \\ kZ \\ k \end{bmatrix}. \qquad \text{(5b)}$$

Para fazer o mapeamento da projeção do ponto na imagem para o ambiente onde se está, é possível usar a equação (6):

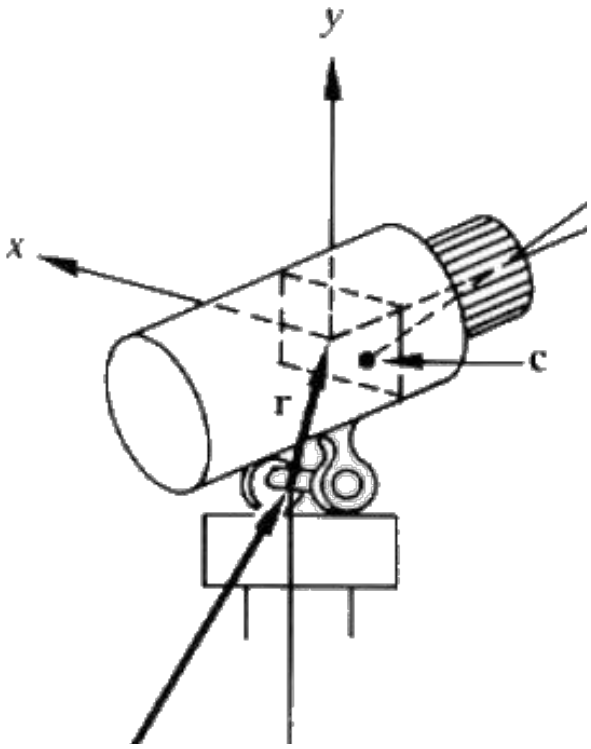
$$c_h = G \cdot C \cdot R \cdot P \cdot w_h. \qquad (6)$$

Sendo (X_0, Y_0, Z_0) a posição da câmera no ambiente, é possível escrever a matriz de translação G da seguinte forma:

$$G = \begin{bmatrix} 1 & 0 & 0 & -X_0 \\ 0 & 1 & 0 & -Y_0 \\ 0 & 0 & 1 & -Z_0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \qquad (7)$$

Uma nova translação deve ser necessária, pois as coordenadas adquiridas na imagem têm como base o retângulo onde a imagem é formada na câmera e não sua base de fixação, como mostrado na figura 7. Assim sendo, considere um vetor r como a matriz de translação do vetor da câmera, que é descrito da seguinte forma:

Figura 7 – Vetor de translação r.



Fonte: Adaptado de Rodrigues, 2016.[4]

$$C = \begin{bmatrix} 1 & 0 & 0 & -r_x \\ 0 & 1 & 0 & -r_y \\ 0 & 0 & 1 & -r_z \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{8}$$

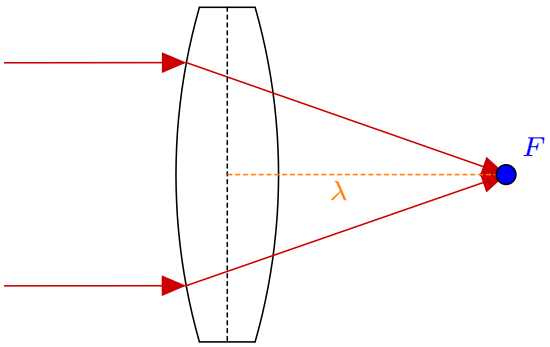
Da mesma forma que é possível transladar a câmera para a posição desejada, será necessário rotacionar a câmera. Porém, para cada eixo do plano tem-se uma matriz de rotação. Essas matrizes são mostradas a seguir. Considere que α, β e γ são os graus rotacionados respectivamente dos eixos x, y e z:

$$R_x = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos\,\alpha & \sin\,\alpha & 0 \\ 0 & -\sin\,\alpha & \cos\,\alpha & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_y = \begin{bmatrix} \cos\,\beta & 0 & -\sin\,\beta & 0 \\ 0 & 1 & 0 & 0 \\ \sin\,\beta & 0 & \cos\,\beta & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} R_z = \begin{bmatrix} \cos\,\gamma & \sin\,\gamma & 0 & 0 \\ -\sin\,\gamma & \cos\,\gamma & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}. \tag{9}$$

[4]RODRIGUES, Paulo Sérgio Silva. Calibração de câmeras. Apresentação de Slides. Centro Universitário FEI, fev. 2016. Disponível em: <http://www.fei.edu.br/~psergio/VisaoComputacional/Aula%206%20Calibra%C3%A7%C3%A3o%20de%20C%C3%A2meras.ppt>.

A matriz R, usada na equação (6), é o resultado do produto matricial $R_x \cdot R_y \cdot R_z$. A matriz de perspectiva se utiliza da distância focal da lente da câmera (λ). Na figura 8 é possível notar que essa distância representa a distância do centro da lente até o ponto focal.

Figura 8 – Lente convergindo a luz direcionada.



Fonte: Do autor.

Assim sendo, pode-se descrever a matriz de perspectiva da seguinte maneira:

$$P = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & -\frac{1}{\lambda} & 1 \end{bmatrix}. \tag{10}$$

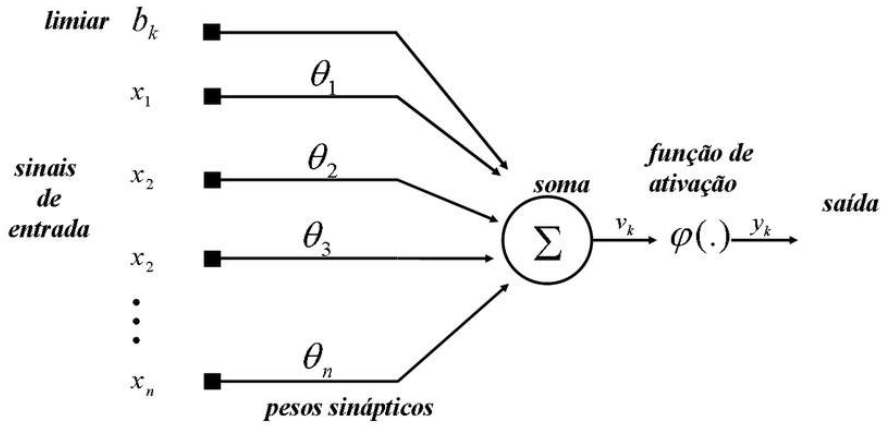
Contudo, estes valores padrões são difíceis de ser encontrados ou, em algumas situações, não podem ser encontrados. Por isso, para a realização desse cálculo utilizou-se aprendizado de máquina, e a técnica escolhida para isso foi a rede neural artificial.

2.5 REDES NEURAIS ARTIFICIAIS

Inspiradas por redes neurais biológicas, a Rede Neural Artificial (RNA) é uma técnica de aprendizado de máquina proposta por McCulloch e Pitts (1943) e aperfeiçoada por Rumelhart, Hinton e Williams (1986). Estes sistemas aprendem utilizando exemplos como dados. Um caso desta natureza é a utilização de valores de entrada de uma função $f(x)$ na entrada de uma rede neural. A resposta na saída deve ser, se não a mesma, muito próxima do valor que se teria ao se colocar o valor de entrada na função $f(x)$. Para se fazer isso, utiliza-se o neurônio artificial mostrado na figura 9.

[5]FURTADO, Helaine; CAMPOS VELHO, Haroldo; MACAU, Elbert. **ASSIMILAÇÃO DE DADOS COM REDES NEURAIS ARTIFICIAIS EM EQUAÇÕES DIFERENCIAIS**. [s.l.: s.n.], set. 2011.

Figura 9 – Representação de um neurônio artificial.



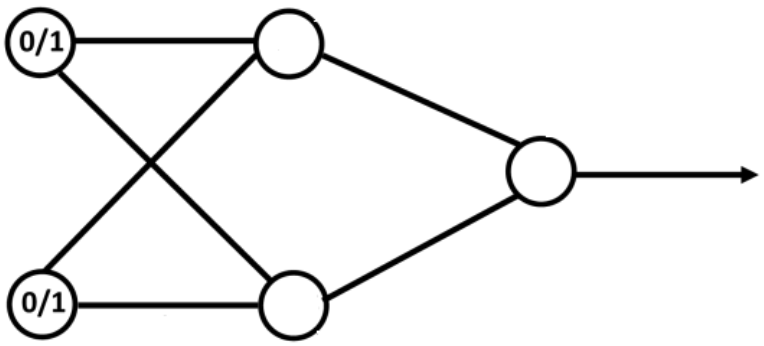
Fonte: Furtado; Campos Velho; Macau, 2011.[5]

O neurônio artificial contém uma entrada chamada de limiar ou bias. As demais entradas são x do sistema que podem ser inseridas. Pode-se ter n variáveis de entrada. Essas entradas podem ser valores constantes de uma função ou valores de pixels em uma imagem e etc. Como exemplo, serão usados os valores de uma porta lógica XOR.

Cada variável que entra no sistema é multiplicada por um peso sináptico. Os resultados destes cálculos são somados e o valor unitário aplicado a uma função de ativação $\phi(.)$. Existem diversas que podem ser usadas, como a discreta, linear, sigmoidal, logarítmica, tangente e etc.

Em computação, o processamento deste dado normalmente é realizado utilizando-se matriz. Na figura 10 é mostrada uma RNA que pode implementar a porta lógica XOR.

Figura 10 – Topologia de uma rede neural para implementação de uma porta lógica XOR.



Fonte: Do autor.

Neste caso, é possível ver que se tem variáveis de entradas e não se tem o bias. Assim sendo, a matriz que representaria a entrada é

$$x = \begin{bmatrix} x_1 & x_2 \end{bmatrix}, \tag{11}$$

por ser uma matriz de entrada, ela sempre deve ter uma quantidade de linhas igual a 1 e a quantidade de colunas igual a de entradas mais o bias, se necessário. A matriz de entrada y deve ser multiplicada pela matriz de pesos. Essa matriz é representada como

$$p = \begin{bmatrix} \theta_{11} & \theta_{12} \\ \theta_{21} & \theta_{22} \end{bmatrix}, \tag{12}$$

na qual a quantidade de entradas mais o bias (se necessário) para na camada. A quantidade de colunas deve ser igual à quantidade de neurônios desejados na camada. Se esta regra for seguida, então se terá os pesos do primeiro neurônio representados pelos valores θ_{1x} na primeira coluna e os pesos do segundo na segunda coluna, e assim consecutivamente.

Ao realizar o produto matricial de x e p, teremos como resultado

$$\nu_k = \begin{bmatrix} x_1 \times \theta_{11} + x_2 \times \theta_{12} & x_1 \times \theta_{21} + x_2 \times \theta_{22} \end{bmatrix} = \begin{bmatrix} \nu_{11} & \nu_{12} \end{bmatrix}, \tag{13}$$

isso representa o primeiro passo da somatória do produto de pesos e entradas. O resultado ν_k será aplicado à função de ativação $\phi(.)$, e como consequência se terá uma nova matriz y_k que será usada como entrada para a próxima camada ou como resposta, se for a última camada.

Para realizar um exemplo se utilizará a topologia mostrada na figura 10. Os valores de entrada serão:

$$x = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{14}$$

os pesos da primeira camada são

$$p_1 = \begin{bmatrix} 2 & -1 \\ -1 & 2 \end{bmatrix}, \tag{15}$$

com isso, teremos um

$$\nu_1 = \begin{bmatrix} 2 - 1 \end{bmatrix}, \tag{16}$$

para a função de ativação será usada uma função degrau, na qual quando o valor for menor que dois, ele será zero. Caso contrário, esse valor será um. Aplicando a função de ativação sobre ν_1 temos:

$$y_1 = \begin{bmatrix} 1 & 0 \end{bmatrix}, \tag{17}$$

a matriz y_1 será a entrada para a próxima camada. Os pesos desta segunda camada são:

$$p_2 = \begin{bmatrix} 2 \\ 2 \end{bmatrix}, \tag{18}$$

sendo assim, tem-se como resultado da somatória:

$$\nu_2 = \begin{bmatrix} 2 \end{bmatrix}, \tag{19}$$

aplicando a função de ativação sobre ν_2, tem-se:

$$y_2 = \left[1\right], \tag{20}$$

e este é o resultado retornado pela RNA da figura 10. Quando se tem uma entrada unitária e outra zero, obtém-se como resultado um valor unitário, igual ao vindo de uma porta lógica.

Porém, os valores de p_1 e p_2 foram calculados anteriormente, e na grande maioria dos casos, esses valores necessitam ser aprendidos. Para que se possa aprender, é utilizado o trabalho de Rumelhart, Hinton e Williams (1986), o *backpropagation*. Utilizando exemplos de treinamento e iniciando com pesos aleatórios, pode-se utilizar a descida de gradiente e a derivada da função de ativação para calcular correções nos pesos. A cada passo um exemplo é colocado na entrada e calculada a saída. Ao calcular a saída, um erro é calculado retropropagado pela rede atualizando os pesos. Isso é chamado de aprendizado supervisionado.

Como pode ser observado, esta técnica é capaz de realizar uma regressão linear ou não linear. Neste trabalho foi utilizada uma rede neural para aprender o modelo geométrico do sistema de visão de cada robô.

Mesmo as câmeras e lentes sendo do mesmo modelo, possuem características intrínsecas diferentes, o que faria ser necessário o modelamento de cada par de câmeras e lentes. Entretanto, ao se utilizar uma RNA, pode-se trazer um treinamento de uma rede na qual os parâmetros de entrada são os mesmos utilizados em um modelo geométrico, e como saída, tem-se a distância do objeto desejado.

Entretanto, esta informação faz a relação de um ponto específico na imagem para o mundo, mas em alguns casos é necessário detectar o objeto e classificá-lo, pois essa técnica necessita de algumas informações do objeto. Para fazer esse mapeamento, se faz necessário utilizar detectores para reconhecer o objeto em questão e utilizar as informações características deste objeto.

2.6 DETECTORES DE OBJETOS

Uma pessoa detecta um objeto por meio da análise de informações características deste objeto. Quando se tem um conjunto de características que correspondem a uma determinada classe, então é possível realizar uma detecção.

Para realizar essa detecção em um sistema computacional, utiliza-se este mesmo princípio. Após a detecção da região de interesse, pode-se usar características externas ou internas para realizar a classificação do objeto. Um exemplo de característica externa é o contorno de um objeto, ou a sua fronteira. Características internas são todas as informações que podem ser obtidas realizando a análise dos pixels contidos nesta fronteira. Independente da técnica usada, todas seguem o mesmo princípio, que é usar uma ou mais características de um objeto para a realização da detecção.

Os detectores, em sua grande maioria, funcionam com um pré-treinamento, no qual imagens do objeto desejado são selecionadas e devidamente marcadas. Chama-se esse conjunto

de imagens de "imagens positivas". Em seguida, um novo conjunto de imagens é selecionado. Neste caso, não contendo o objeto desejado, mas o ambiente ou o contexto onde a detecção será realizada. Esse é o conjunto de imagens negativas.

Com esses dois conjuntos de imagens um treinamento é realizado, onde se tenta extrair o máximo de características do objeto, sem que essas características sejam tão genéricas a ponto de detectar o objeto, onde não se existe. Esta etapa é chamada offline. Quando o treinamento é finalizado, um arquivo com as características do objeto é criado. Este arquivo é usado para a execução online do algoritmo, ou seja, para a detecção dos objetos em imagens que não foram treinadas, imagens essas que podem vir de um novo conjunto de dados ou de uma câmera (GONZALEZ; WOODS, s.d.).

2.6.1 Descritor Histograma de Gradientes Orientados (HOG)

A detecção de borda é utilizada como uma das características externas que pode-se usar para classificar um objeto. Na visão computacional, o vetor gradiente é uma das principais técnicas para a detecção de borda: quanto maior for a diferença de valores dos pixels, maior será a magnitude deste gradiente, e assim, maior a probabilidade de naquele lugar existir uma borda. A orientação é dada pelo ângulo deste vetor. Para o cálculo da magnitude do vetor em uma dada posição p_x, p_y de uma imagem, pode-se aplicar a equação (21):

$$magnitude = \sqrt{(p_{x+1} - p_{x-1})^2 + (p_{y+1} - p_{y-1})^2}. \tag{21}$$

A orientação do gradiente é dada pela equação (22):

$$\hat{a}ngulo = \tan^{-1}\left(\frac{(p_{x+1} - p_{x-1})^2}{(p_{y+1} - p_{y-1})^2}\right), \tag{22}$$

na qual p representa a intensidade do pixel em sua respectiva coordenada. Uma característica que traz robustez a esta técnica é que diferentes valores de pixels podem trazer o mesmo resultado quanto à magnitude e ângulo do vetor, tornando a técnica mais robusta em relação à variação de iluminação (FORSYTH; PONCE, 2002).

O Histograma de Gradientes Orientados (HOG) é um vetor das quantidades de gradientes com as mesmas características. Quando se obtém o histograma de gradiente de um determinado objeto, tem-se um descritor que pode ser usado para a sua detecção em novas imagens. Para classificar uma imagem, primeiro calcula-se o valor do HOG em diversos pontos de interesse na imagem, depois utiliza-se um classificador para encontrar a classe à qual aquele HOG pertence.

2.6.2 Máquina de vetores de suporte (SVM)

A SVM foi criada por Cortes e Vapnik (1995), referindo-se a um conjunto de métodos utilizados para aprendizado supervisionado. Esse algoritmo realiza o reconhecimento de pa-

drões entre dados. Ele é utilizado para classificação e regressão linear (OLIVEIRA VILÃO, 2015).

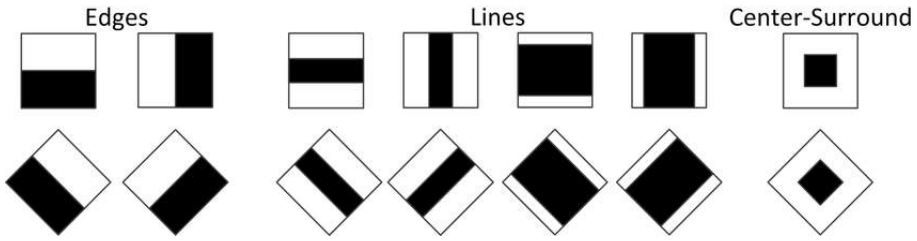
Para se realizar esta classificação, um conjunto de hiperplanos é criado em um espaço de altas dimensões, a fim de realizar a classificação dos dados. Uma boa separação é realizada. Um hiperplano atinge a maior distância entre os dados de treinamento e qualquer outra classe. Quando o hiperplano que faz a separação ótima é encontrado, finaliza-se o treinamento e os dados são descartados. Para realizar novas classificações é usado apenas o hiperplano encontrado. Realizam-se os cálculos necessários para saber em qual dos lados do hiperplano este novo dado se posiciona e, dependendo desta posição, a classificação é realizada.

A combinação do HOG-SVM foi usada no trabalho de Oliveira Vilão (2015), para realizar a detecção de robô. Para detectar a bola foi utilizado o HAAR-*AdaBoost*.

2.6.3 Descritores Haar

Proposta por Haar (1910), a análise Haar é similar a análise de Fourier, na qual uma função objetivo pode ser representada por um intervalo de função com base ortogonal, utilizando as ondaletas de Haar. As ondaletas são funções contínuas e não diferenciáveis, o que em alguns casos pode ser considerado um problema: elas podem determinar funções de transições súbitas, especialmente em figuras. A ideia de se usar ondaletas de Haar foi melhorada por Viola e Jones (2001), que criou as características Haar. Estas são características de imagens digitais, que podem ser usadas no reconhecimento de objetos (OLIVEIRA VILÃO, 2015). Na figura 11 são mostradas as características Haar usualmente utilizadas.

Figura 11 – Descritores Haar.



Fonte: Bianco et al., 2014.

2.6.4 Classificador Boosting

Essa técnica consiste em usar classificadores mais fracos e mais imprecisos para que juntos possam ter um resultado mais certeiro e com maior robustez. Para isso, uma lista de características é criada, como em uma árvore de decisão. O classificador fraco que melhor predisser o objeto será a raiz da árvore, e assim sucessivamente até que um limiar de classificação

seja atingido e uma árvore seja criada, na qual diversos tipos de classificações fracas sejam testadas, a fim de classificar o objeto (OLIVEIRA VILÃO, 2015).

Unindo as técnicas de Haar e *Boosting* é possível realizar a detecção dos objetos e usar essa informação para a localização do objeto no mundo e para o rastreamento deste objeto.

Estas técnicas foram estudadas anteriormente no trabalho de Oliveira Vilão (2015), no qual foi verificado o desempenho destas técnicas no ambiente da RoboCup humanoide.

2.6.5 Deep neural network

Sua primeira aparição foi proposta por Heck et al. (2000) no reconhecimento de falantes, entretanto não houve um grande avanço para provar sua eficiência no reconhecimento de fala. Um tempo depois, Hinton et al. (2012) se utiliza deste trabalho para aperfeiçoar o trabalho com redes neurais profundas.

Outra área que se beneficiou deste grande avanço científico foi a visão computacional. O projeto ImageNet tem mais de 14 milhões de imagens que foram classificadas por pessoas. Nessas imagens existem mais de 20 mil categorias das mais diversas. Em 2010, uma competição para a detecção de objetos foi criada, a ImageNet Large Scale Visual Recognition Challenge (ILSVRC).

Neste desafio, deve-se criar um algoritmo capaz de realizar uma detecção. Até 2011 o erro estimado destes algoritmos era de 25%. Contudo, em 2012, Krizhevsky, Sutskever e Hinton (2012) participou desta competição com um algoritmo de *deep convolutional neural networks*, conseguindo erros de 16%, um grande avanço para os algoritmos da época.

Na atualidade, estas redes têm erros extremamente baixos e, em alguns casos, chegam a superar o homem. Com isso, houve algumas mudanças quanto ao desafio. Hoje, além da detecção de objetos em imagens, tem-se os desafios de fazer detecção em vídeos, criação de legendas para imagens, dentre outros (RUSSAKOVSKY et al., 2015).

Atualmente, a RNP tem se mostrado extremamente eficaz para a detecção de objetos, entretanto sua eficiência de execução está muito atrelada ao processamento em paralelo oferecido pelas placas de vídeo. Hoje umas das áreas mais pesquisadas é a melhoria no desempenho dos algoritmos, visto a dependência de hardware, pois mesmo que de fato esse avanço tenha possibilitado o crescimento e o desenvolvimento desta tecnologia, muito do que se utiliza ainda não tem esse recurso tão desenvolvido, ou não se tem a placa de vídeo.

Um dos trabalhos que tem como objetivo o desempenho desta rede foi realizado por Howard et al. (2017). A MobileNets é uma das arquiteturas de RNP desenvolvidas para serem executadas em dispositivos móveis. Seu propósito é que ela possa manter a alta taxa de acurácia de uma RNP, porém ainda permitir a execução em hardwares não tão potentes, como para o caso proposto um dispositivo móvel (celulares, tablets, smartphones, drones, etc).

A rede proposta foi comparada com algumas das redes mais conhecidas para a detecção de objetos. Na tabela 1 foram realizadas algumas comparações entre arquiteturas de redes.

Figura 12 – A MobileNets foi projetada para execução em dispositivos móveis.

Fonte: Howard et al., 2017.

Dentre os diversos fatores analisados se retratará apenas a acurácia. Neste ponto, é possível ver que mesmo com as alterações na topologia, ainda se tem uma acurácia no mesmo patamar que as demais redes.

Tabela 1 – Comparação realizada entre algumas arquiteturas de RNP utilizadas

Modelo	***ImageNet Accuracy***
1.0 MobileNet-224	70.6%
GoogleNet	69.8%
VGG 16	71.5%
0.50 MobileNet-160	60.2%
Squeezenet	57.5%
AlexNet	57.2%

Fonte: Howard et al., 2017.

Como no desafio, estas redes são treinadas para a detecção de mil categorias de objetos nos mais diversos ambientes. Entretanto, a necessidade que se tem neste trabalho é a detecção de poucas categorias em um ambiente estruturalmente igual.

2.7 RASTREAMENTO DE OBJETOS

O problema do rastreamento consiste em inferir o movimento realizado por um objeto a partir de informações passadas. No âmbito de visão computacional, esse dado vem de uma sequência de imagens, cada qual posicionada em um instante no tempo. Essas informações podem ser medidas de diversas maneiras, desde um ponto na imagem até um conjunto de pontos. Todavia, essa informação pode não ter uma alta confiabilidade, podendo em algum momento

Figura 13 – Detecção de objetos utilizando a MobileNet SSD.

Fonte: Howard et al., 2017.

ter medidas vindas de ruídos ou incompletas. O propósito de se realizar o rastreamento é, a partir das medidas realizadas e do modelo dinâmico, rastrear o objeto e definir o seu estado no mundo para um instante atual ou futuro. Existem diversas circunstâncias para se querer fazer o rastreamento de um objeto. Algumas delas são listadas a seguir:

a) **Captura de movimento**: Imagine um sistema de segurança de uma rodovia no qual por meio de uma câmera é possível mapear os movimentos de um veículo e estudar suas características. Quando for detectada alguma anomalia no movimento do objeto, como uma frenagem abrupta ou a parada de um carro em uma das pistas, algumas ações podem ser tomadas para que, além de evitar acidentes, haja uma diminuição no fluxo de veículos naquela pista até que tudo seja normalizado.

b) **Reconhecimento de movimentos**: Dados os movimentos dos objetos, é possível determinar a qual classe pertencem ou, para objetos já classificados, reconhecer quais são as tarefas que estão sendo realizadas. Esses movimentos podem ser expressivos, como um guarda de trânsito coordenando um cruzamento, ou discretos, como a linguagem de sinais.

c) **Alvo**: Sabendo qual o movimento que está sendo executado é possível tomar decisões com base em acontecimentos futuros que podem ser previstos. Um exemplo são os carros autônomos predizendo uma batida ou um atropelamento, tomando uma decisão antecipada de frenagem ou desvio para evitar esse acontecimento.

Levando em consideração uma situação na qual existe a necessidade de rastrear um objeto no mundo, para a realização desta tarefa se levará em consideração a posição em que o objeto está, a velocidade, e a aceleração: essas informações definem o estado.

Para que seja possível atualizar o estado, é necessário obter informações com base em algum sistema de medição, que podem ser sensores de distância, temperatura, câmeras ou qualquer outro recurso que traga uma informação do estado. Além de conter, na sua grande maioria, ruído, essa informação também pode não representar todo o estado. Em um sistema de visão que faz a detecção de objetos, usualmente é possível estimar apenas a posição do mesmo em uma das variáveis que representa o estado, portanto as demais informações devem ser encontradas através do modelo dinâmico do sistema, que traz a relação entre as variáveis no sistema e seu comportamento no tempo.

Além de inferir informações que não se poderá medir diretamente, o modelo dinâmico do sistema pode trazer informações de estados futuros, tendo como base as informações dos estados passados. Assim, é possível predizer estados sabendo o modelo de movimento que rege esse sistema, o que obriga a fundição de informações dinâmicas e previsões, algo realizável quando se modela o problema com uma estrutura probabilística.

2.7.1 Filtro de Kalman

O filtro de Kalman é uma das formas de se modelar probabilisticamente um problema de rastreamento. Ele tem como fundamento duas premissas: a primeira é que as medições realizadas dependem exclusivamente do estado no qual o objeto se encontra, e a segunda é que o estado no qual nos encontramos depende apenas do estado anterior (Hipótese de Markov) (FORSYTH; PONCE, 2002).

O filtro de Kalman assume estados contínuos e representa suas certezas com uma média (μ_t) e a covariância (Σ_t) representadas no tempo (t). Para representar a probabilidade da transição do estado deve-se usar uma função linear, sempre levando em consideração um ruído de transição. A equação tem a seguinte característica:

$$x_t = A_t \cdot x_{t-1} + B_t \cdot u_t + e_t, \tag{23}$$

onde x é a representação dos estados, u_t são as ações tomadas neste estado e e_t um erro gaussiano com média zero e covariância R_t. As matrizes A_t e B_t serão explicadas mais adiante. Com isso, é possível escrever um pseudocódigo do funcionamento do algoritmo:

As linhas 3 e 4 do algoritmo são voltadas para a primeira etapa do filtro, o que diz respeito à predição, entretanto os valores encontrados nesta etapa são intermediários, por isso são representados como $\overline{x}_t$ e $\overline{\Sigma}_t$. Na etapa de atualização, que é mostrada nas linhas 5 a 7, a medida realizada pelo sensor é considerada, e os valores estimados são comparados. Essa etapa pode ser executada mais de uma vez, caso haja mais de um sensor.

Algoritmo 1 – Pseudocódigo representando o funcionamento do filtro de Kalman.

Entrada: x_{t-1}, Σ_{t-1}, u_t, z_t
Saída: x_t, Σ_t
$\overline{x}_t = A_t \cdot x_{t-1} + B_t \cdot u_t$
$\overline{\Sigma}_t = A_t \cdot \Sigma_{t-1} \cdot A_t^T + R_t$
$K_t = \overline{\Sigma}_t \cdot C_t^T \cdot (C_t \cdot \overline{\Sigma}_t \cdot C_t^T + Q_t)^{-1}$
$x_t = \overline{x}_t + K_t \cdot (z_t - C_t \cdot \overline{x}_t)$
$\Sigma_t = (I - K_t \cdot C_t) \cdot \overline{\Sigma}_t$
retorna x_t, Σ_t

Fonte: Thrun; Burgard; Fox, 2005.

Levando em consideração que se está seguindo um objeto em um plano cartesiano, toda informação de posição, velocidade e aceleração deve levar em consideração ambos os eixos, pois assim se consegue uma melhor representabilidade do estado. A matriz de estado x conterá 6 elementos, e é mostrada na equação 24.

$$x = \begin{bmatrix} pos_x \\ pos_y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} \tag{24}$$

A matriz de A_t mostra a correlação entre as variáveis do estado e também é chamada de matriz de transição de estado, pois através dela pode-se predizer novos estados. Para este caso, as regras que regem o sistema são as equações do movimento uniformemente variável. Sendo assim, escreve-se as equações na forma matricial, seguindo o modelo de predição $A_t \cdot x_{t-1}$.

$$\begin{bmatrix} 1 & 0 & t & 0 & 0.5t^2 & 0 \\ 0 & 1 & 0 & t & 0 & 0.5t^2 \\ 0 & 0 & 1 & 0 & t & 0 \\ 0 & 0 & 0 & 1 & 0 & t \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \cdot \begin{bmatrix} pos_x \\ pos_y \\ v_x \\ v_y \\ a_x \\ a_y \end{bmatrix} = \begin{bmatrix} 0.5a_x t^2 + pos_x + tv_x \\ 0.5a_y t^2 + pos_y + tv_y \\ a_x t + v_x \\ a_y t + v_y \\ a_x \\ a_y \end{bmatrix} \tag{25}$$

Como é possível observar, a constante tempo é utilizada na matriz A_t. Isso é importante, pois a representação do estado se torna atemporal. Todo o movimento previsto até este momento leva em consideração exclusivamente a dinâmica do objeto. Para finalizar o processo de predição é necessário levar em consideração os movimentos realizados pelo agente neste período, neste caso, o robô, e para isso utiliza-se a matriz B_t, conhecida como matriz de controle. Levando em consideração os movimentos do robô e que o mapa aqui descrito é um mapa egocêntrico, a matriz de controle é descrita da seguinte maneira:

$$B_t = \begin{bmatrix} -tvr_x & 0 & R_p \cos\left(\omega rt + \arctan\left(\frac{pos_y}{pos_x}\right)\right) - pos_x \\ 0 & -tvr_y & R_p \sin\left(\omega rt + \arctan\left(\frac{pos_y}{pos_x}\right)\right) - pos_y \\ 0 & 0 & R_v \cos\left(\omega rt + \arctan\left(\frac{v_y}{v_x}\right)\right) - v_x \\ 0 & 0 & -R_v \sin\left(\omega rt + \arctan\left(\frac{v_y}{v_x}\right)\right) - v_y \\ 0 & 0 & R_a \cos\left(\omega rt + \arctan\left(\frac{a_y}{a_x}\right)\right) - a_x \\ 0 & 0 & -R_a \sin\left(\omega rt + \arctan\left(\frac{a_y}{a_x}\right)\right) - a_y \end{bmatrix} \tag{26}$$

Os valores de vr_x, vr_y e ωr são respectivamente a velocidade do robô para o lado, para frente e o giro. Com isso, encerra-se a fase de predição e assim, é possível estimar os estados futuros ou atuais com base em informações passadas. Entretanto, se houver apenas a predição não é possível ter a certeza de onde está o objeto, pois nesta etapa sempre há erros de movimentos que são acrescentados. Na linha 4 esse erro é representado pela matriz R_t. Para aumentar a certeza desta informação, utiliza-se sensores que possam trazer informações do estado em que se está. A etapa que cuida deste processo é a atualização. Na linha 5 encontra-se a matriz K_t, chamada de ganho Kalman. Ela é a responsável por fazer a correlação entre o que foi predito e o que foi medido. Como dito anteriormente, na grande maioria dos casos, os sensores não medem todas a variáveis que representam um estado. Assim, a matriz C_t é a responsável por fazer essa relação entre o que foi medido e o que ela representa no estado. A matriz Q_t representa o desvio padrão do sensor utilizado para fazer a medida. A matriz C_t é mostrada a seguir:

$$C_t = \begin{bmatrix} 1 & 0 & 0 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 & 0 & 0 \end{bmatrix} \tag{27}$$

As matrizes Q_t e R_t serão determinadas através de medições dos valores de sensores e movimentos do robô.

3 TRABALHOS RELACIONADOS

Quando, por fatores diversos, não se pode observar um objeto, deve-se ter uma forma de prever a posição do mesmo. Deste modo, um modelo de movimento se faz necessário, pois mesmo não detectando o objeto, precisa-se armazenar a informação da posição por um período de tempo. Este foi um dos desafios encontrados no trabalho de Deguchi, Kawanaka e Okatani (2004), no qual uma pessoa é seguida em uma imagem e quando ocorre uma falha na detecção, a predição é iniciada para que ainda se tenha informações atualizadas do objeto e uma localização, mesmo quando ele não está sendo detectado pela câmera.

Outra forma de solucionar este problema é deixar mais robusta a técnica de detecção. Mesmo os objetos sendo parcialmente ocluídos, é possível ainda realizar a classificação do mesmo e detectá-lo na imagem. Porém, técnicas deste tipo são normalmente mais custosas quando comparadas às técnicas de rastreamento.

Figura 14 – Rastreamento de pessoa utilizando filtro de partículas.

Fonte: Deguchi; Kawanaka; Okatani, 2004.

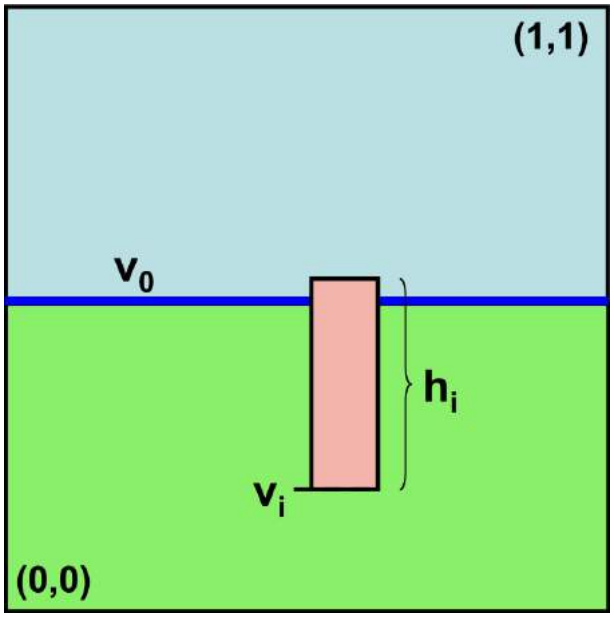
No trabalho de Coifman et al. (1998), existe a necessidade de extrair informações extrínsecas de objetos e utilizá-las para uma manipulação melhor na infraestrutura rodoviária de cidades. Entretanto, para que seja possível utilizar esse dado, um mapeamento da informação na imagem para o mundo é necessário, pois assim será possível utilizar a informação mapeada e correlacioná-la com informações do mundo.

Uma maneira de se fazer isso foi proposta no trabalho de Hoiem, Efros e Hebert (2008), no qual, levando em consideração o contexto da imagem e as características de um objeto foi possível classificá-lo com uma maior exatidão. Levando em consideração a altura do objeto detectado, pode-se localizar seu posicionamento no mundo, no qual dada a altura do objeto e partindo do pressuposto de que este objeto deve estar no chão, utiliza-se a perspectiva da câmera para estimar a sua distância.

Na figura 15 foi mostrada uma simplificação dos valores necessários para se fazer este mapeamento. Em geral, os valores necessários são a altura da câmera e do objeto detectado, o quanto esta câmera está inclinada e onde a linha do horizonte se passa na imagem.

O trabalho de Çelik (2013), também foi desenvolvido com o foco na interação do sistema de visão com a localização, tendo como foco a *RoboCup* humanoide. Com base na experiência adquirida, foi concluído que o sistema de visão tem ruído e que para utilizar a informação

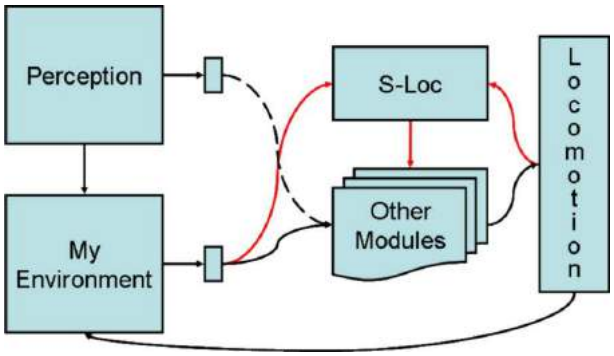
Figura 15 – Considerando informações como a linha do horizonte e altura do objeto é possível se fazer o mapeamento para o mundo.



Fonte: Hoiem; Efros; Hebert, 2008.

deste processo se fazia necessário um sistema entre o módulo de visão e os demais processos que utilizam a informação gerada. O sistema proposto é mostrado na figura 16, na qual o processo de *Perception*, neste caso, representa os sensores para detecção do objeto (visão) e a informação gerada é pré-processada pelo processo de *My Environment* e transmitida para os demais processos.

Figura 16 – Arquitetura utilizada no trabalho Çelik, 2013.



Fonte: Çelik, 2013.

Independente da técnica de detecção usada, ela terá falsos positivos e negativos que ocasionam alguns problemas quando este dado é usado diretamente para a tomada de decisão e/ou localização dos objetos. Um falso positivo pode ocasionar decisões erradas ou localizações

incorretas. Além disso, a informação é instantânea, ou seja, assim que o robô deixa de detectar a bola, seja por motivo de falso positivo ou oclusão do objeto, o robô considera imediatamente a perda do objeto, o que gera uma quebra na decisão correta que estava sendo executada. Uma informação errada, com ruídos ou com uma interpretação falha, pode acarretar em um problema maior quando se compara com a falta de informação. Além disso, outros sensores podem e devem ser usados para aumentar a confiabilidade da predição, como é o caso da odometria.

Entretanto, quando se possui diversos sensores para trabalhar e informações assíncronas, ou seja, que têm uma frequência de atualização irregular, essas informações acabam sendo inconsistentes, pois cada uma delas pertence a um instante no tempo e não podem ser comparadas entre si. A predição é a forma que se tem de estimar o valor desta informação no instante de tempo atual a partir dessas informações passadas. Quanto mais antiga a informação, maior a incerteza que se tem na predição. Um trabalho que mostra a eficiência do para trabalhar com vários sensores com tipos de medidas diferentes é o de Kriegman, Triendl e Binford (1989), que dispõe de diversos sensores e cada qual transmite a informação de uma maneira, podendo ser uma medida direta do que se deseja ou uma média indireta que deve ser tratada.

Kriegman, Triendl e Binford (1989), utilizou sensores de toque e laser aliados a um sistema de câmeras estéreo. As informações de distância na imagem são mapeadas para o mundo e aliadas com os outros sensores para melhor resolução. Utilizando características de pontos conhecidos, após fazer a detecção e o mapeamento destes pontos para o mundo, uma correlação é realizada entre os pontos detectados por todos os sensores. Ao invés de ter diversos sensores para realizar este mapeamento, neste trabalho terá apenas uma visão monocular. Entretanto, a classificação do objeto permite a obtenção de informações importantes e com isso, a realização do mesmo mapeamento para o ambiente.

Quando se tem essa informação no mundo aliada com a informação temporal, os efeitos da detecção de falsos positivos e negativos são minimizados, trazendo uma robustez maior e uma informação mais precisa, porém não se deve descartar a necessidade de levar em consideração os movimentos executados por esse agente (ÇELIK, 2013).

Outro fato que também será levado em consideração é a relação que os objetos estáticos têm entre si. No trabalho de Leonard e Durrant-Whyte (1991), são dadas as características dos pontos detectados e as suas relações em um mapa já conhecido, sendo possível estimar nossa posição no ambiente. No trabalho proposto serão empregadas essas relações para classificar os *landmarks* detectados e consequentemente localizar todos os *landmarks* estáticos. Por essa classificação dos *landmarks* ser realizada pela informação extraída das posições dos pontos no ambiente e suas relações, é possível classificar esses *landmarks* mais rápido do que em uma classificação no sistema da visão. Para o armazenamento dos dados temporais vindos à visão, utilizou-se um buffer de memória que armazena essas informações para serem usadas conforme a necessidade. Entretanto, uma das propostas deste trabalho é usar o filtro de Kalman para o armazenamento temporal destas informações, assim se tem um processamento mais rápido para inferência da nova posição e não apenas as N posições armazenadas no *buffer* para inferir.

Na *RoboCup* há algumas equipes que tratam os ruídos vindo do sistema de visão. Uma destas equipes é a Rhoban, a campeã da *RoboCup humanoid* 2016 e 2017. Para fazer esse tratamento, eles utilizam dois filtros de partículas. O primeiro deles é utilizado para se manter um mapa egocêntrico dos objetos que estão sendo detectados e o segundo é para o mapa global dos objetos no campo e o próprio robô (LY et al., 2016). Isso é feito para tratar o erro que vem do sistema de visão, como falsos positivos e a própria incerteza quanto às medições feitas pelo sistema de visão.

Contudo, essa filtragem também pode ser melhor quando se tem um sistema de visão ativo. Isso foi mostrado no trabalho de Seekircher, Laue e Röfer (2011), no qual foi proposto um sistema para o controle ativo da cabeça do robô. Esse sistema foi utilizado tendo como base a minimização da entropia. Com isso, o robô é capaz de saber qual é o melhor lugar para manter o foco da visão e assim, se manter localizado. Essa técnica é capaz de aumentar a precisão da localização do robô no mundo e de rastrear o objeto neste ambiente. Os objetos que são utilizados para a autolocalização são as linhas do campo e seus cruzamentos. A bola pode também ser usada para esse fim. A figura 17 mostra um exemplo desta detecção.

Figura 17 – Detecção das linhas de campos e suas intersecções feitas no trabalho de Seekircher; Laue; Röfer.

Fonte: Seekircher; Laue; Röfer, 2011.

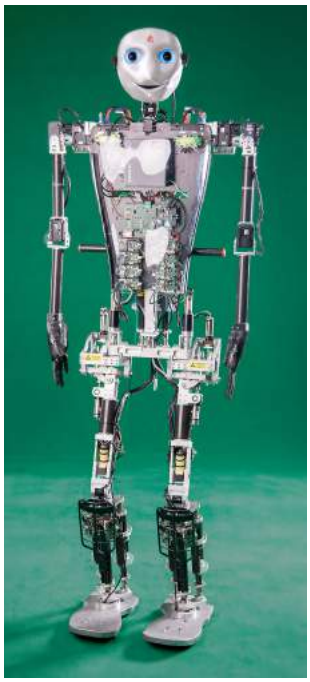
Uma forma alternativa de se realizar o controle da cabeça é a utilização do valor da informação perfeita. Assim, também é possível fazer o controle de qual objeto deve ser detectado para que se tenha um aumento na certeza relacionada à posição dos objetos. Essa informação já é calculada atualmente pelo sistema de localização do time RoboFEI.

Algo essencial para o rastreamento dos objetos é sua detecção. No trabalho de Schnekenburger et al. (2017), foram testadas algumas RNP para detecção dos objetos no domínio da RoboCup.

O robô utilizado para os testes é um robô da categoria AdultSize (figura 18). Todos os objetos da liga humanoide têm as mesmas características, dadas as devidas proporções. Isso é

importante ser ressaltado porque os testes realizados por este robô podem ser aproveitados para validar a utilização de uma RNP para outras categorias.

Figura 18 – Robô da categoria AdultSize.

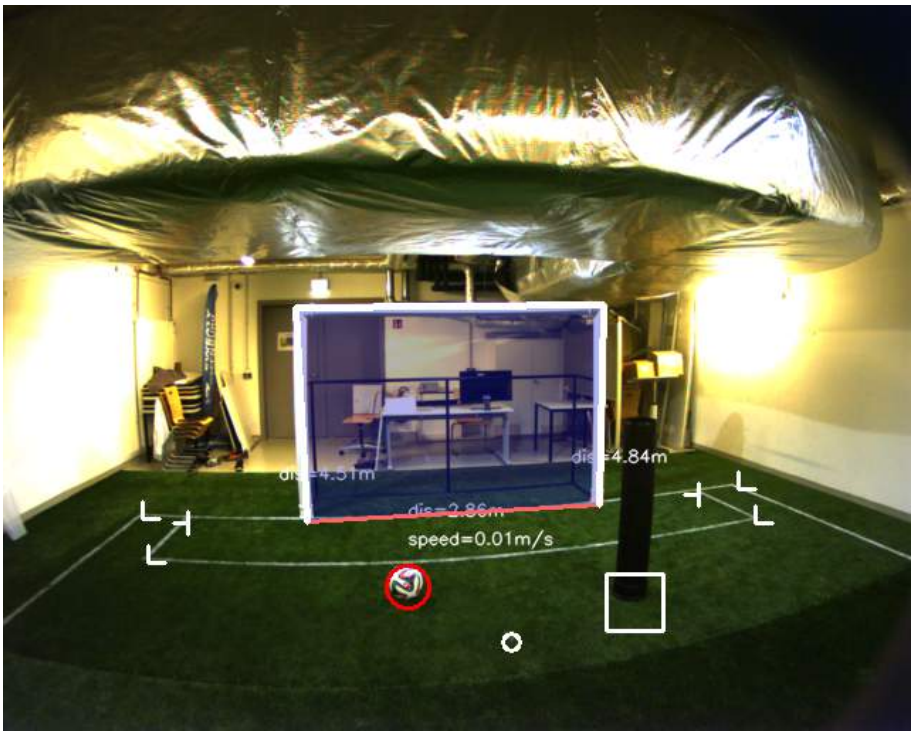


Fonte: Schnekenburger et al., 2017.

No trabalho proposto foi utilizada uma RNP para detectar a bola, landmarks e obstáculos que são característicos nesta categoria. O propósito destas detecções foi a utilização das informações para o sistema de localização e para a estratégia do robô. Na figura 19 temos um exemplo quanto à detecção destes objetos.

Os próximos capítulos mostrarão o trabalho realizado tendo como base os trabalhos aqui apresentados e a estrutura que atualmente se utiliza no robô.

Figura 19 – Detecção de objetos no domínio da RoboCup humanoide.



Fonte: Schnekenburger et al., 2017.

4 SISTEMA VISUAL E RASTREAMENTO DE OBJETOS UTILIZANDO FILTRO DE KALMAN

Na biologia, a capacidade de reter uma pequena quantidade de informações em um curto período de tempo é chamada de memória visual a curto prazo. Essa habilidade é extremamente importante para a locomoção, pois permite lembrar de objetos recentemente vistos para planejar uma trajetória, como quando se atravessa uma rua ou quando um motorista troca de faixa (COGNIFIT, 2017). Sem esta memória nos esqueceríamos de coisas simples, como o farol vermelho ao atravessar a rua ou o carro ao lado ao trocar de faixa, o que causaria acidentes. Existem diversas doenças que prejudicam este tipo de memória, como por exemplo o Alzheimer, que em alguns casos tem como sintoma a perda ou enfraquecimento desta habilidade, o que leva a pessoa a ter dificuldade para se locomover (KAWAS et al., 2003).

Ao se locomover ou se planejar, essa memória é acionada. As informações observadas são utilizadas juntamente com outras informações, que são inferidas e aplicadas para uma tomada de decisão. Para o âmbito da robótica existe algo similar. Assim, dado um conjunto de informações, deve-se ser capaz de inferir e extrair novas informações sobre o ambiente e tomar uma decisão a respeito.

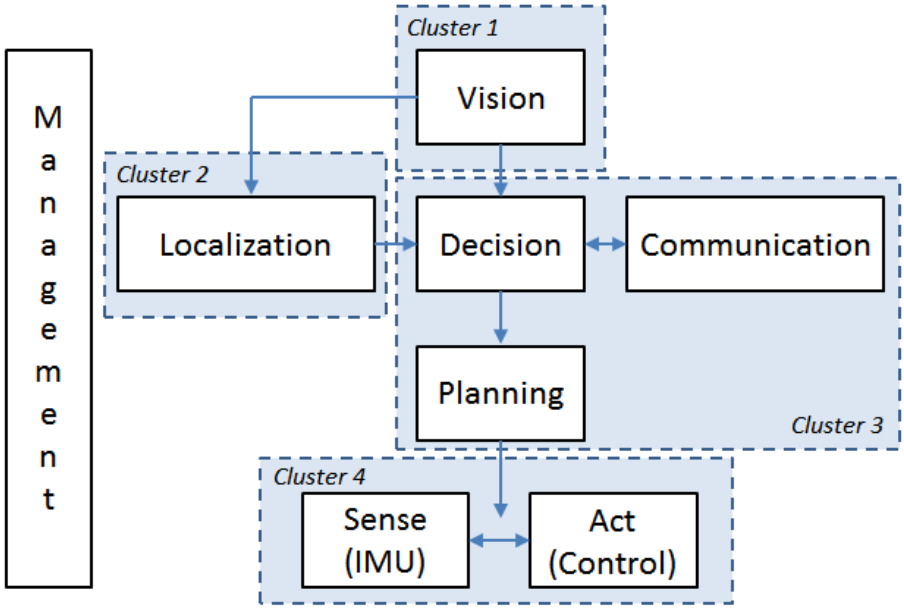
A proposta deste trabalho é a implementação de um sistema de memória visual de curto prazo para o robô humanoide da FEI. Esse sistema deve ser capaz de utilizar apenas informações do sistema visual para inferir novas informações e de armazenar informações relevantes recebidas anteriormente.

4.1 ARQUITETURA ORIGINAL

Atualmente o robô tem como arquitetura de funcionamento a *cross architecture* (figura 20). Ela é composta de diversos processos, e cada um deles contém uma tarefa fundamental para o todo. Um exemplo é o controle, um processo responsável pelos movimentos do robô, uma tarefa aparentemente simples, mas que tem seus desafios, como o gerenciamento de equilíbrio, ações reativas e gestão de desempenho dos servomotores. Para o sistema de decisão, é indiferente saber que o robô está tendendo para a direita, portanto com essa arquitetura consegue-se filtrar esse tipo de informação e transmitir apenas informações relevantes.

A troca de informações entre os processos não é gerenciada por nenhum outro processo, tendo sido simplificada, o que é denominado memória compartilhada (*blackboard*). Quando um processo deseja transmitir alguma informação para outros processos, escreve esta informação na memória física do computador (RAM), em uma variável pré-estabelecida. Recapitulando o exemplo do controle do parágrafo anterior, quando o processo de decisão deseja que ele se movimente para frente, ele escreve na variável "decision_control_a"o valor “1”. Por ser um espaço na memória compartilhado por todos os processos, todos têm acesso ao valor da variável, inclusive o controle que ao ler o valor sabe que deve se movimentar para frente.

Figura 20 – Arquitetura atualmente utilizada para o RoboFEI, as setas em azul representa o sentido da comunicação entre os processos.



Fonte: Perico et al., 2014.

Para este trabalho, o foco estará em dois processos: o de visão e o de decisão. Como pode ser visto na figura 20, o sistema de visão transmite a informação adquirida por meio da câmera diretamente para o sistema de decisão. Da mesma maneira, quando a decisão necessitar trocar o foco da visão para um novo objeto, ela tem uma comunicação direta através de uma variável no *blackboard*, a qual ao ser alterada, também altera o enfoque entre os objetos detectados. O objetivo principal do sistema de visão é detectar, se não todos, o máximo de objetos possível em uma imagem e transmitir a informação para o sistema de decisão. Contudo, a arquitetura do sistema de visão permite a detecção de apenas um objeto por vez. Sendo assim, ao se detectar um objeto a informação de posição deste objeto é escrita no *blackboard* juntamente com a informação de qual objeto corresponde. A informação de posição do objeto é transmitida de forma qualitativa, representada por perto, médio, longe e muito longe.

A decisão tem como objetivo montar a melhor estratégia possível, com base nas informações que recebe. Consequentemente, a responsabilidade de validar as informações para assegurar sua credibilidade se tornou uma de suas tarefas. Para se fazer isso, a decisão armazena a última informação vinda do sistema de visão como verdade absoluta por um período de tempo, e caso não haja uma nova detecção nesse período, se considera como um objetivo perdido.

Esse é o problema principal que foi abordado, pois quando o sistema de visão para de realizar uma detecção, seja porque o objeto saiu do campo de visão, foi ocluído ou parou de

detectar para detectar outro objeto, o sistema rapidamente “esquece” do objeto anterior, o que na maioria dos casos gera um transtorno para a decisão.

Para solucionar este problema foi desenvolvido um sistema de memória visual. Porém, o conceito de memória de curto prazo está além do sistema visual e antecede o sistema de decisão, por isso uma adaptação na arquitetura foi realizada.

4.2 PROPOSTA DE MELHORIA NA ARQUITETURA

Um novo módulo será acrescentado à arquitetura, batizado de memória visual. Ele tem como objetivo ser um intermediador do sistema de visão e dos demais sistemas. Na figura 21 é mostrada a alteração feita na arquitetura. O sistema de visão envia a informação exclusivamente para a memória visual e apenas ela controla a visão. Com isso, o sistema de visão tem uma única função: a de detectar o objeto no *frame*.

Os objetivos fundamentais dos processos de visão e decisão se mantêm. Entretanto, foi necessária uma remodelagem na estrutura do sistema de visão, para que venha a se adaptar ao sistema de memória visual. A memória visual tem como objetivo utilizar a informação vinda do sistema de visão, armazená-la e tratá-la para que se possa ter uma melhor decisão. Inicialmente, irá se detalhar o funcionamento da visão, para então explicar como a memória visual se utiliza da informação transmitida a ela e interage com a visão.

4.2.1 Sistema de Visão

O sistema de visão deve transmitir um conjunto de informações dos objetos detectados. Todas essas informações serão escritas em um espaço no *blackboard* e, quando utilizadas pela memória visual, a variável “tag” será zerada para que um novo dado possa ser escrito no lugar. São 21 espaços liberados para a detecção de robôs, um espaço para a bola e 18 espaços para landmarks. Os dados que serão transmitidos são mostrados na tabela 2.

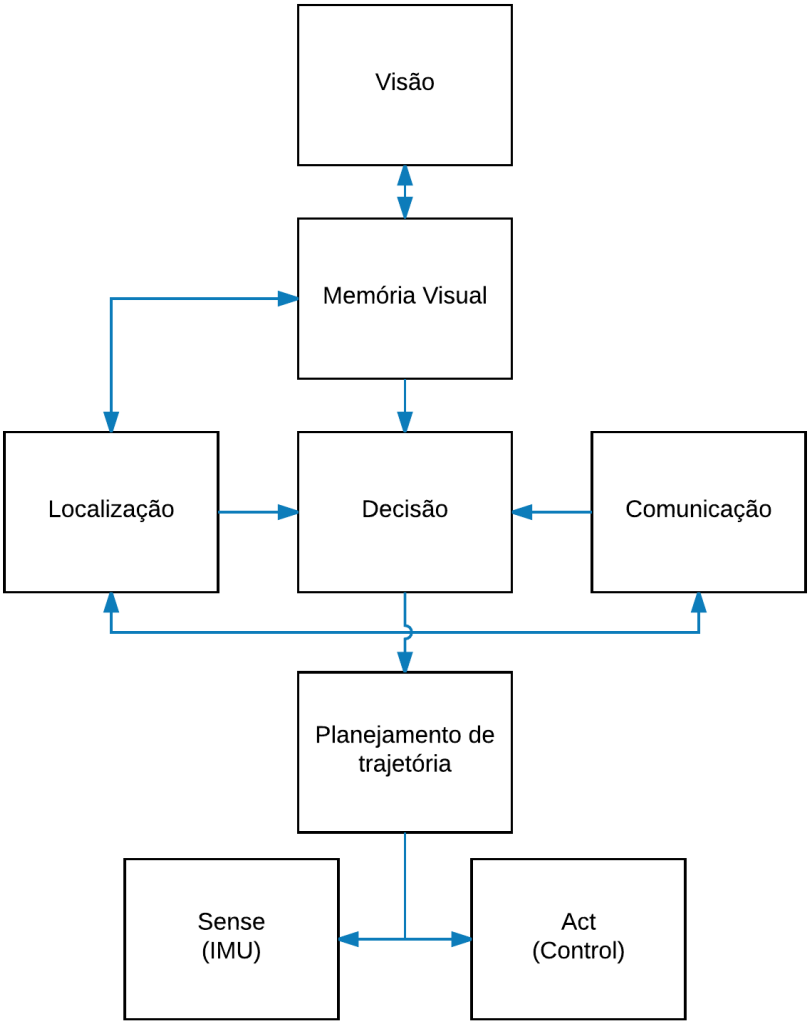
Tabela 2 – Lista de variáveis que serão usadas no *blackboard* para descrever objeto na imagem.

Nome Variável	***Tipo***	***Descrição***
tag	Int	Número que define qual objeto representa, no caso robôs se ele é adversário, aliado, ou desconhecido;
positionx	Float	Posição X em cm do centro do robô até objeto;
positiony	Float	Posição Y em cm do centro do robô até objeto;
time	Double	O instante no tempo em que a informação foi obtida.

Fonte: Do autor.

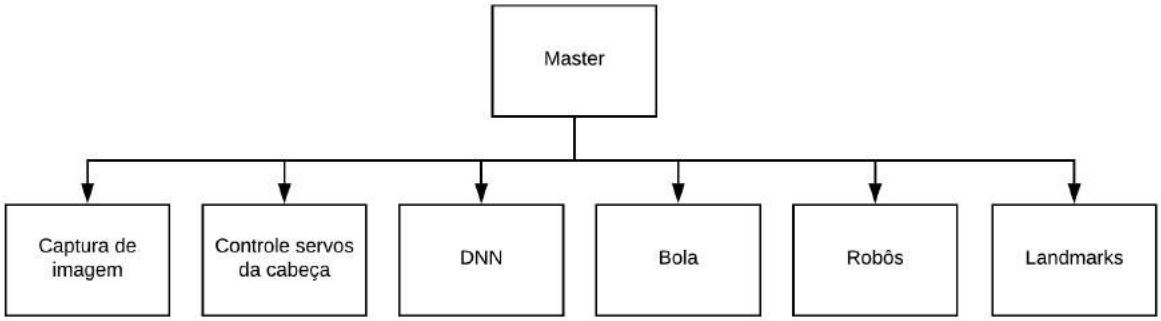
Para realizar a detecção dos objetos simultaneamente, optou-se pela arquitetura de *threads*, na qual cada *thread* é responsável por um determinado grupo de objetos. Assim foi desenvolvida uma arquitetura para a visão baseada na figura 22.

Figura 21 – Arquitetura proposta para no RoboFEI, as setas em azul representa o sentido da comunicação entre os processos.



Fonte: Do autor.

Figura 22 – Cada módulo representa uma *thread* que será executada pelo sistema de visão.



Fonte: Do autor.

As *threads* se comunicam através de variáveis internas do programa. A execução e troca de dados entre elas é organizada pela *thread master*, porém as *threads* responsáveis por objetos como bola, robô e *landmark* têm acesso ao *blackboard*, para que assim, possam escrever os valores mostrados na tabela 2.

4.2.1.1 Master

Sua função principal é manter todos os processos em operação com o menor custo possível. Caso haja alguma falha em um processo, a master irá reiniciá-lo. O algoritmo 2 exemplifica como funcionará essa manipulação de informações.

Algoritmo 2 – Funcionamento *thread master* no sistema de visão.

```
1  Inicializar todas as threads;
2  enquanto não for solicitado a finalização do programa faça
3  |  Inicializa contagem de tempo;
4  |  Realiza uma cópia da última observação da thread captura de imagem e envia
   |   para thread DNN;
5  |  Salva os elementos que foram detectados em list;
6  |  para cada thread inicializada faça
7  |  |  Envia a informação dos elementos em list que lhe corresponde;
8  |  fim
9  |  se contagem de tempo for menor que pré-estabelecida então
10 |  |  Aguarda tempo restante;
11 |  fim
12 fim
13 Finaliza todas as threads.
```

Fonte: Do autor.

Quando o processo de visão é chamado, a *thread master* é executada. Seu primeiro passo é inicializar as demais *threads*, porém a inicialização é feita uma por vez, e caso aconteça algum problema durante o processo de inicialização, ela tem a responsabilidade de reinicializar o processo ou finalizar os demais processos e informar o erro.

A solicitação de encerramento pode vir de alguma *thread* que apresente um mal funcionamento, externamente, de algum processo que solicite, ou até mesmo de um usuário. Para todos estes casos, a *master* deve finalizar cada *thread* e monitorar se ela foi devidamente finalizada, pois quando não há uma finalização adequada, criam-se processos que ocupam processamento da máquina e consequentemente recurso computacional.

Uma contagem de tempo é inicializada neste processo, pois durante os testes se constatou que não se faz necessária uma detecção a taxas tão altas. O tempo que está sendo utilizado é de um segundo. Isso quer dizer que o sistema de visão transmite uma informação a cada um segundo.

A observação mostrada no algoritmo 2 é um vetor que contém informações relevantes naquele instante de tempo. Ela será melhor explicada na seção seguinte.

Ao finalizar o processamento, a *thread* DNN gera uma lista com todos os objetos que foram detectados e retorna para a *master*, que armazena essa lista. Para cada *thread* é enviada a observação realizada, porém apenas a parte da lista que lhe diz respeito. Um exemplo é a *thread* robô, que recebe uma lista apenas dos objetos robôs, e para o caso dos *landmarks*, a lista enviada é sobre todos os *landmarks* detectados.

4.2.1.2 Captura de imagens

Esta *thread* tem como função capturar em um instante no tempo as informações vindas da câmera, juntamente com outras informações que constituem uma observação. As informações de observação são mostradas na tabela 3.

Tabela 3 – Lista de valores usados para descrever uma observação realizada pela *thread* Captura de imagens.

Nome Variável	***Tipo***	***Descrição***
frame	Mat (OpenCV)	Frame capturado da câmera
pos_tilt	Float	Posição do motor responsável pelo tilt da câmera no instante de leitura
pos_pan	Float	Posição do motor responsável pelo pan da câmera no instante de leitura
mov	Int	Movimento que está sendo realizado no instante de observação
time	Double	Qual instante no tempo foi observado

Fonte: Do autor.

A aquisição de informações acontece de forma ininterrupta a uma taxa de $0{,}0\overline{3}$ segundos. Essa é a taxa correspondente aos Quadros por segundo, do inglês *frames per second* (FPS), da câmera de 30. O algoritmo 3 mostra como é processada sua função principal, porém ela também é capaz de tratar alguns erros, como a rápida perda de conexão com a câmera ou a mudança da porta na qual está conectada.

Ao ser inicializada, ela procura a câmera entre as portas do computador, e quando se conecta a ela, armazena o endereço da porta em que foi localizada a câmera. Caso não consiga localizar a câmera, um erro é informado para a *thread master*, solicitando o encerramento do processo.

Todavia, outra forma que se tem para a utilização desta thread é por meio do argumento de entrada "vídeo". Quando este argumento é passado, deve passar também um endereço do vídeo. Assim, ao iniciar o sistema, ao invés de a câmera ser acessada para a leitura do quadro, o vídeo é acessado. Com isso, é possível realizar testes e depurações de filmagens de jogos, ou experimentos.

A contagem de tempo à qual se refere é a FPS da câmera utilizada. O processamento para a leitura da câmera deve sempre utilizar a taxa da mesma, a fim de sempre se ter os dados mais atualizados e não gerar *buffer* de imagens.

Algoritmo 3 – Funcionamento *thread* captura de imagem no sistema de visão.

```
1  Inicializar todas os periféricos;
2  enquanto não for solicitado a finalização do programa faça
3  |  Inicializa contagem de tempo;
4  |  Armazena um quadro da câmera;
5  |  Armazena os valores dos motores de pan e tilt;
6  |  Armazena o inteiro que representa qual movimento está sendo executado pelo
   |   robô;
7  |  Gera o vetor observation com os valores armazenados anteriormente;
8  |  se contagem de tempo for menor que pré-estabelecida então
9  |  |  Aguarda tempo restante;
10 |  fim
11 fim
12 Finaliza todas os periféricos.
```

Fonte: Do autor.

Para se fazer a leitura do dado da câmera, utiliza-se a biblioteca *OpenCV*, uma biblioteca *open source* de visão computacional. Após realizar a aquisição do quadro, uma leitura no *blackboard* é realizada para se obter a posição dos motores de *pan* e *tilt* no instante de leitura do quadro. Essa informação é utilizada para auxiliar na inferência de distâncias e ângulos dos objetos no ambiente.

Outro dado do qual se faz a leitura é o valor que corresponde a qual movimento está sendo executado pelo robô naquele instante. Essa informação é utilizada pela memória visual para calcular sua própria velocidade. Com os dados armazenados, um vetor é criado com estas informações. O nome dado para este vetor é "observação".

4.2.1.3 DNN

Foi criada uma *thread* para a RNP, devido à necessidade que se tem para preparação, custo no tempo de processamento e funções secundárias à detecção. A arquitetura utilizada foi proposta por Howard et al. (2017): uma rede de detecção de objetos proposta para sistemas móveis. Ela foi escolhida justamente por sua característica de detectar múltiplos objetos simultaneamente e arquitetura desenvolvida para sistemas móveis. Contudo, nenhuma alteração ou adaptação nesta arquitetura foi alterada.

Para o desenvolvimento, foi utilizada a biblioteca Tensorflow (MARTÍN ABADI et al., 2015). Esta biblioteca é exclusiva para aprendizado de máquina utilizando RNP. Com ela, foi criada a classe para detecção dos objetos e o treinamento. Para este trabalho foi utilizada uma rede apenas para a detecção dos objetos de futebol de robô humanoide. São oito objetos no total, sendo eles robôs, bola, intersecção de linhas em “L”, “T” e “X”, marca do pênalti, círculo central e base do gol.

Quando a rede finaliza a detecção, uma lista dos objetos detectados é criada. Para cada objeto detectado tem-se a classe à qual ele pertence e a posição na qual o objeto está na imagem.

Algoritmo 4 – Algoritmo da *thread* DNN no sistema de visão.

```
1  Inicializar todas os periféricos;
2  enquanto não for solicitado a finalização do programa faça
3  |  enquanto não receber um vetor de observações faça
4  |  |  Aguarda envio;
5  |  fim
6  |  Realiza a detecção do quadro enviado;
7  |  Gera uma lista com todos os elementos detectados;
8  |  Filtra os objetos detectados inferiores ao "threshold_to_train_max;"
9  |  Retorna a lista criada para a thread master;
10 fim
```

Fonte: Do autor.

Essas informações são anexadas à observação que lhe foi enviada e retornam para a *thread master*. Esses valores são mostrados na tabela 4.

Tabela 4 – Para cada objeto que se é detectado um conjunto destes dados é adicionado em uma lista.

Nome variavel	***Tipo***	***Descrição***
x	Float	Mostra qual ponto inicial no eixo x da imagem para iniciar o contorno do objeto
y	Float	Mostra qual ponto inicial no eixo y da imagem para iniciar o contorno do objeto
height	Float	Mostra qual a altura do objeto em porcentagem
width	Float	Mostra qual a largura do objeto em porcentagem
class	Int	Mostra qual o tipo do objeto detectado dentre as classes conhecidas
probability	Float	Qual a probabilidade deste objeto ser o que foi detectado

Fonte: Do autor.

Após o processamento, realiza-se uma filtragem para eliminar os dados detectados erroneamente. Utiliza-se o valor "*probability*" de cada objeto para detectar e faz-se uma comparação com o valor *threshold_to_train_max*. Caso essa probabilidade seja menor, então o objeto é descartado.

Uma funcionalidade secundária que foi implementada é a opção de *train*. Quando esse argumento de entrada é passado para a visão, esta salva os quadros que lhe foram enviados, e caso consiga fazer uma detecção, um arquivo xml é criado com o mesmo nome do quadro, informando qual posição do quadro foi detectada e qual objeto foi classificado. Porém, neste caso, o valor que se usa para fazer a filtragem de ruídos é *threshold_to_train_min*. O propósito desta implementação é facilitar na aquisição de novas imagens e quando possível, já marcar os objetos que já são conhecidos, mas nos quais se tenha uma menor confiabilidade.

4.2.1.4 Robôs

Esta *thread* tem como função tratar as informações referentes a robôs vindas do sistema da RNP (algoritmo 5). Para que a informação possa ser utilizada pelo sistema de memória, ela deve ser convertida para o ambiente egocêntrico que a memória trabalha. Isso é feito utilizando uma rede neural. Nela temos como parâmetros de entrada as posições x e y da base e centro do objeto, o tamanho do objeto na imagem e o tamanho do objeto no mundo real. Como saída se tem as distâncias x e y do objeto até a câmera.

Algoritmo 5 – Funcionamento *thread* robôs no sistema de visão.

```
1  Inicializar todas os periféricos;
2  enquanto não for solicitado a finalização do programa faça
3      para cada robô detectado faça
4          Segmenta a região do objeto;
5          Determina qual time o robô pertence ou atribui desconhecido;
6          Utiliza rede neural para estimar distância do objeto x e y do objeto;
7          Escreve no blackboard as informações detectadas;
8      fim
9      enquanto não houver um novo conjunto de dados faça
10         Aguardar;
11     fim
12 fim
13 Finaliza todas os periféricos.
```

Fonte: Do autor.

Além da estimativa de distância, uma segmentação de cores também é realizada a fim de determinar a qual time o robô pertence. Isso é feito convertendo apenas o quadro onde o objeto se encontra para HSI. Após a conversão, são realizadas segmentações de ciano e magenta. Se a cor predominante for maior que um mínimo determinado anteriormente, então o robô é classificado como adversário ou aliado. Caso contrário, ele é dado como desconhecido. Finalizadas as extrações de características, os dados extraídos são escritos no *blackboard*, como mostrado na tabela 3.

4.2.1.5 Landmarks

Esta *thread* tem como objetivo detectar os *landmarks* e transmitir essa informação para o sistema de localização. Para se fazer isso, será necessário ter a classificação do *landmark* e o ângulo em graus, sendo a relação do centro do robô, o robô e o *landmark*. O algoritmo 6 mostra como é implementada essa função.

Para realizar o cálculo do ângulo do algoritmo, foi feita a leitura da posição do *landmark* na imagem. As posições x e y do centro de *landmark* são passadas para uma função pré-determinada na qual tem-se o ângulo em graus como retorno. Essa função estima apenas o

Figura 23 – Utilizando a segmentação de cores é realizada a classificação dos times em um jogo.

(a) Segmentação de cor para magenta.

Fonte: Do autor.

(b) Segmentação de cor para ciano.

Fonte: Do autor.

Algoritmo 6 – Algoritmo da *thread* Landmarks no sistema de visão.

```
1  Inicializar todas os periféricos;
2  enquanto não for solicitado a finalização do programa faça
3  |  para cada landmark detectado faça
4  |  |  Determinar ângulo entre o robô e landmark;
5  |  |  Escreve no blackboard as informações detectadas;
6  |  fim
7  |  enquanto não houver um novo conjunto de dados faça
8  |  |  Aguardar;
9  |  fim
10 fim
11 Finaliza todas os periféricos.
```

Fonte: Do autor.

centro do ângulo da câmera para o *landmark*. A este ângulo ainda é acrescido o valor do motor *pan*.

Toda essa informação é finalmente transmitida para a localização, utilizando o *blackboard*. As informações dos *landmarks* detectados não são transmitidas para memória visual, pois ela é responsável por realizar o rastreamento de apenas um destes *landmarks*. Ele será transmitido pela localização. Essa estrutura será melhor explicada na seção 4.2.2.2.

4.2.1.6 Controle da cabeça

Esta é a *thread* é responsável por controlar os motores da cabeça. Seu objetivo é manter o foco da câmera para o objeto de maior necessidade quanto à detecção. Para fazer isso, utiliza a informação transmitida pela memória visual através do *blackboard*. No algoritmo 7 o seu funcionamento é mostrado.

A contagem de tempo na qual o controle da cabeça se fundamenta é a comunicação dos servomotores. A equipe RoboFEI utiliza um servomotor que tem uma taxa de comunicação de um mega-hertz. Dada a leitura realizada no *blackboard* e a lista de todos os objetos que foram detectados pela RNP, o processo calcula quais serão os novos valores para os motores de *pan* e *tilt*, transmite para os servomotores e escreve no *blackboard*.

4.2.2 Memória Visual

Esse processo é o responsável pelo rastreamento dos objetos detectados pelo sistema de visão. Para se realizar essa operação, uma arquitetura de *threads* também foi formada, na qual uma *thread* é criada para cada objeto, e todos esses módulos serão gerenciados pela *master*, que é responsável por ler as informações passadas pelo sistema de visão e distribuí-las para suas respectivas *threads*. Para as informações incompletas, como por exemplo a detecção de um robô sem a detecção do time ao qual ele pertencente, a *master* vai inferir a qual objeto essa informação corresponde e transmitir esse dado apenas para o módulo ao qual essa informação

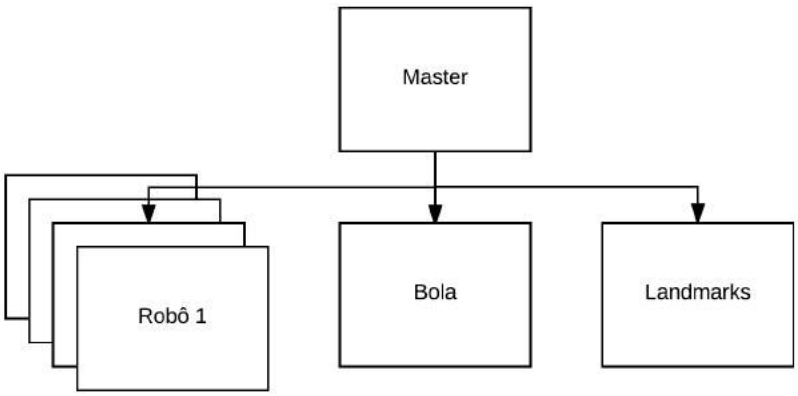
Algoritmo 7 – Algoritmo da *thread* Controle da cabeça no sistema de visão.

```
1  Inicializar todas os periféricos;
2  enquanto não for solicitado a finalização do programa faça
3  |  Inicia a contagem de tempo;
4  |  Realiza a leitura do objeto de interesse;
5  |  se objeto de interesse está na lista de detecções então
6  |  |  Determina os valores dos motores para centralizar o objeto no quadro;
7  |  |  Escreve esse valor nos motores e no blackboard;
8  |  fim
9  |  senão
10 |  |  Inicia uma varredura no local mais provável onde ele possa estar;
11 |  |  Escreve esse valor nos motores e no blackboard;
12 |  fim
13 |  se contagem de tempo for menor que pré-estabelecida então
14 |  |  Aguarda o tempo restante;
15 |  fim
16 fim
17 Finaliza todas os periféricos.
```

Fonte: Do autor.

mais se assemelha. Um diagrama da arquitetura proposta é mostrado na figura 24, na qual cada módulo representa um tipo de *thread* que será executada e cada objeto terá sua *thread*, que gerenciará a predição e a atualização deste objeto com base na informação transmitida pela *master*. Existem sete *threads* do tipo robôs. Cada uma delas será responsável por realizar o rastreamento de um robô.

Figura 24 – Arquitetura proposta para sistema de memória visual.



Fonte: Do autor.

Da mesma maneira que o sistema de visão, as *threads* também tem acesso ao *blackboard*, em que cada *thread* escreve as informações que calculou em seu respectivo espaço. A transição dos dados para os demais processos também usará uma lista na memória compartilhada. Entretanto, essa lista já representa todos os elementos possíveis e suas características. Nesse caso seriam um objeto para a bola e vinte e um objetos de robô. Cada informação será

processada por uma *thread* de uma maneira específica. Os próximos parágrafos trarão mais detalhes.

Os valores escritos pelas *threads* são mostrados na tabela 5. O dado de posição sempre leva em consideração um mapa egocêntrico do objeto. A velocidade do objeto será transmitida utilizando o módulo das velocidades calculadas em x e y. Em exemplo, pode-se supor que a memória tenha calculado uma velocidade de -3 cm/s em x e 4 cm/s em y. A velocidade que será escrita no *blackboard* é de 5 cm/s.

Tabela 5 – Dados transmitidos pela memória visual para representar os objetos que estão sendo rastreados.

Nome variavel	***Tipo***	***Descrição***
tag	Int	Informa se o objeto foi localizado/está sendo predito e valida a informação seguinte
x	Float	Distância no eixo x em centímetros
y	Float	Distância no eixo y em centímetros
velocidade	Float	Módulo da velocidade máxima já atingida pelo objeto
aceleração	Float	Módulo da aceleração atual do objeto

Fonte: Do autor

4.2.2.1 Master

Na memória visual esse algoritmo tem como principal função a distribuição das informações vindas do sistema de visão para os objetos. No algoritmo 8 é mostrada a implementação desta *thread*.

Para exemplificar imaginemos que no instante um foram detectados três robôs sendo eles um adversário, um aliado e um indeterminado, no instante de tempo dois foram detectados dois robôs adversários, e um aliado. Esses dados é passado como argumento de entrada para o algoritmo 9, ao ser iniciado o instante um de tempo é armazenado na *lasttime*, e observando as threads que já estão em funcionamento as três listas são criadas, vamos supor que houve instantes anteriores no tempo que levaram a criação de execução das três threads, cada uma delas já foi estabelecida anteriormente como robô adversário, aliado e indeterminado, assim sendo cada thread é colocada em sua respectiva lista.

Quando um dado é retirado do vetor X, supondo que este dado seja do robô adversário, então uma lista de candidatos é formada, sendo composta pelas listas de robôs adversários e robôs indeterminados. Com isso, chega-se a dois possíveis candidatos. Para esses candidatos é calculado o peso com base na fórmula mostrada no algoritmo. Levando em consideração que a *thread* com o robô indeterminado seja a que teve o maior peso, então esse dado é passado para a *thread*, o cálculo de atualização é realizado e o status de "robô indeterminado"é atualizado para "robô adversário". A *thread* que recebeu o valor é removida da lista de indeterminados.

O próximo dado que é retirado do vetor X é referente ao robô aliado. Por ser um dado referente a um robô aliado, os possíveis candidatos são uma composição da lista de aliados e

Algoritmo 8 – Algoritmo da *thread master* na memória visual.

```
Inicializar todas as threads;
enquanto não for solicitado a finalização do programa faça
    Inicia a contagem de tempo;
    Faz a aquisição de todos os dados enviados pelo sistema de visão;
    Ordena os dados recebidos por classes e tempo (dos mais antigos até os mais
     recentes);
    Envia os dados para a thread de landmark e aguarda o cálculo de velocidade;
    Envia os dados referente a bola para a thread correspondente;
    Separa os dados de robôs para sua respectiva thread;
    para todas as threads em execução faça
        se o erro predito é dez vezes maior que o erro de visão então
            Finaliza a thread;
        fim
    fim
    para todas as threads em execução faça
        Realizar a predição para o atual instante no tempo e escrever no blackboard;
    fim
    se contagem de tempo for menor que pré-estabelecida então
        Aguarda o tempo restante;
    fim
fim
Finaliza todas as threads.
```

Fonte: Do autor.

Algoritmo 9 – Algoritmo da função "DistribuindoRobos"na memória visual.

```
Entrada: Vetor X com os dados dos robôs
Gera 3 listas com as threads existentes: aliados, adversários, e indeterminados;
Armazena na variável lasttime o primeiro valor de tempo da lista X;
enquanto X não estiver vazia faça
    Retira um dado do vetor X;
    se se o dado tiver um tempo diferente do salvo em lasttime então
        Gera 3 listas com as threads existentes: aliados, adversários, e
         indeterminados;
        Atualiza o valor de lasttime;
    fim
    Utilizando o valor de time do dado gera uma lista de possíveis candidatos;
    Calcula um peso para cada candidato com base na equação:
    2^{\frac{1}{1+\sqrt{(x_{estimado}-x_{dado})^2+(y_{estimado}-y_{dado})^2}}};
    Envia o dado para a thread que tem maior peso;
    Remove o thread da lista que lhe é correspondente (aliados, adversários, e
     indeterminados);
fim
```

Fonte: Do autor.

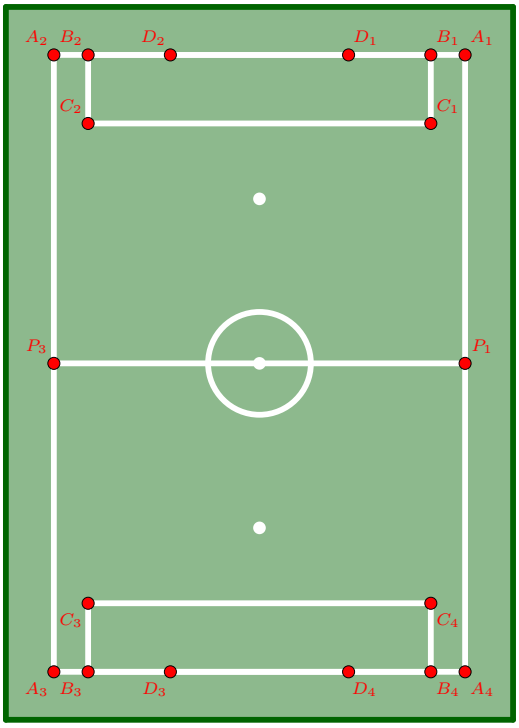
indeterminados. Para esse caso, haverá apenas um candidato, pois a lista de indeterminados agora se encontra vazia. Assim sendo, o dado lido é mandado diretamente para esse candidato e ele é removido da lista de aliados. Acontece de maneira similar com o dado do robô adversário.

Agora, quando tem uma nova retirada do vetor X, será constatado que o valor do tempo é diferente do valor armazenado em *lasttime*. Nesta situação, as listas são atualizadas. Agora, na lista de adversários existem duas *threads*, na lista de aliados apenas uma, e nenhuma na lista de indeterminados. Essa mesma lógica se repete até que se tenha distribuído todos os dados.

4.2.2.2 Landmark

Além do modelo de movimento dos objetos, também se usará em conjunto um modelo de transição com relação ao próprio robô. Ele leva em consideração como o movimento do robô influencia na posição dos objetos a sua volta. Esse modelo será utilizado em todos os objetos. Porém, para se determinar a velocidade e aceleração para esse modelo, os *landmarks* serão usados. Por serem objetos estáticos, ao estimar-se uma velocidade e aceleração, infere-se que essas informações são do próprio robô e repassa-se para os demais processos. Os *landmarks* que serão usados são mostrados na figura 25.

Figura 25 – Imagem do campo de futebol com a representação em vermelho dos landmarks que foram considerados.



Fonte: Do autor.

Como pode ser visto na figura 25, o campo tem pontos característicos (em vermelho), que são usados como *landmarks*. Porém, esses pontos são simétricos, tanto na horizontal quanto na vertical. Levando em consideração essa simetria foram definidos como *landmarks* os pontos A, B, C, D, e P para os quatro quadrantes, devidamente representados de 1 a 4. A informação dos *landmarks* é transmitida diretamente para a localização.

A localização utiliza as informações dos *landmarks* e outros sensores para estimar a posição do robô em campo, com essa informação ela transmite para a memória visual qual a distância em x e y para o *landmark* A_1, com a informação de apenas um *landmark* a predição da velocidade do robô é calculada e utilizada nos demais objetos, para cada movimento que pode ser executado no robô uma matriz de velocidade é calculada, assim a predição também se altera com relação ao movimento que está sendo executado pelo agente. O algoritmo 10 mostra o funcionamento desta *thread*.

Algoritmo 10 – Algoritmo da *thread landmarks* na memória visual.

```
1 Entrada: Vetor X com os dados do landmark
2 para cada dado faça
3 |   Checar qual movimento estava sendo executado;
4 |   Atualizar a matriz de estado com base no movimento;
5 |   Realizar a atualização do filtro de Kalman com base na dado;
6 fim
```

Fonte: Do autor.

Quando se faz a atualização da matriz de estado, as duas primeiras linhas, referentes à posição, se mantêm inalteradas. Apenas as velocidades e a aceleração são alteradas. Na matriz de correlação, as duas primeiras linhas são mantidas e as demais linhas são alteradas. Após essa alteração a simetria da matriz é restabelecida.

Apenas depois deste processo que é realizada a predição do *landmark*, utilizando o filtro de Kalman. Assim, é possível determinar quais são as velocidades e acelerações do robô.

Apenas quando o robô está parado tem-se uma matriz pré-determinada. Para essa matriz, as velocidades, acelerações e matriz de covariância são completamente zeradas. Para os novos movimentos, as velocidades e acelerações também se inicializam do zero, porém os valores da matriz de covariância têm na sua diagonal principal o valor inicial duas vezes maior que o erro de visão.

4.2.2.3 Robôs

Este módulo é responsável pelo rastreamento do robô no campo de futebol. Para predizer a posição deste objeto no tempo, utiliza-se as informações repassadas do sistema de visão e o filtro de Kalman. Essa informação é atualizada a uma taxa de $0{,}\overline{3}$ segundos no *blackboard* para que a decisão e os demais processos possam utilizar essa informação. Quando nenhuma informação é recebida do sistema de visão, inicia-se o período de predição do robô, que se

encerra se tiver um erro de predição maior que 10 vezes o erro de visão, pois neste caso, a melhor decisão será considerar o objeto como perdido e realizar uma nova detecção.

Para se realizar a predição dos robôs não foi considerada a aceleração do objeto, apenas a velocidade, pois ao se movimentarem, os robôs tendem a manter uma mesma velocidade. Outro fator importante para o funcionamento foi a determinação da precisão. Todo valor que seja menor do que ele é automaticamente levado a zero. Assim, consegue-se trazer uma melhor precisão e evita-se erros que podem acontecer devido a valores extremamente pequenos ou grandes.

4.2.2.4 Bola

Para a predição da bola, leva-se em consideração a aceleração e consequentemente, a desaceleração do objeto. Esta é uma informação útil, pois traz a informação da força do chute para o robô que se encontra próximo à bola. Esse dado é importante para uma tomada de decisão. A informação de desaceleração da bola no campo pode ser usada no simulador para se ter um ambiente simulado mais fiel ao real ou para um robô goleiro que planeja uma defesa.

Contudo, essa aceleração não é considerada de forma linear. Quando a bola está em movimento, a direção da velocidade é salva, e se em algum caso de predição ou atualização, a direção da velocidade da bola for alterada, então essa informação é descartada e uma nova predição é realizada até o instante no tempo em que a velocidade seja zero. Com o estado nesse instante, tanto a aceleração quanto a velocidade da bola são zeradas, e assim continua-se o processo que estava sendo realizado.

Com isso, tem-se todas as alterações propostas, utilizadas para desenvolver a memória visual do robô humanoide. O sistema visual agora tem a capacidade de realizar detecções de múltiplos objetos simultaneamente e de realizar um processamento individual e em paralelo para cada categoria de objetos detectados utilizando a arquitetura em thread. Isso é útil para aumentar o desempenho na detecção de todos os objetos e agora ter uma melhor fluidez quanto à informação detectada por cada objeto.

Com a detecção dos objetos, o sistema de memória visual pode utilizar esta informação para gerar um mapa egocêntrico com base nas informações detectadas, e extrair características intrínsecas do objeto detectado e suas próprias informações. Isso é importante para uma melhor estratégia, pois tem um melhor aproveitamento das informações detectadas. A detecção e a extração destas informações serão testadas a seguir.

5 EXPERIMENTOS

Para avaliar o sistema foram realizados três experimentos. Cada um tem como objetivo testar uma característica do sistema. Para fazer isso, utilizou-se o novo sistema de visão, mostrado no capítulo 4.2.1 para se fazer a detecção de robôs. Alguns valores foram pré-estabelecidos nos sistemas para funcionamento. Os valores utilizados são apresentados na tabela 6.

Tabela 6 – Dados utilizados nos experimentos.

Memória Visual		Valor
Robots	precision	0.6
	vision_error	50
Settings	weight_robot	0.6
	execution_period_ms	333

Visão		Valor
RNP	threshold_to_train_max	30%
	threshold_to_train_min	28%
Robots	percentage_time_color	10%

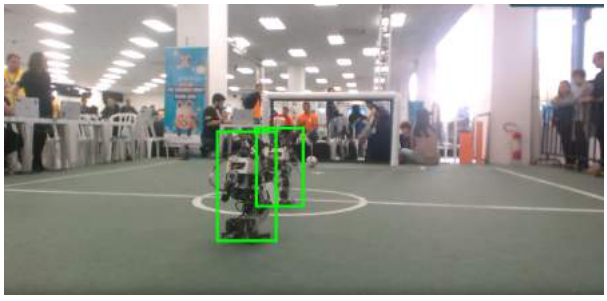
Fonte: Do autor.

5.1 AVALIANDO DESEMPENHO DA RNP COM HAAR

Tendo como base o trabalho de Schnekenburger et al. (2017), no qual foram testadas diversas RNP, foi necessário testar a detecção de objetos em comparação à técnica até então utilizada pela equipe. A proposta deste experimento é validar a utilização de uma RNP na arquitetura da equipe RoboFEI e realizar uma comparação de desempenho da rede. As imagens utilizadas para a comparação têm a resolução de 1280 x 720. Esta foi a resolução escolhida por ser usualmente utilizada pelo sistema de visão da equipe. Com isso, fez-se a detecção dos objetos desejados em cada uma das imagens. Entretanto, a técnica do HAAR-*AdaBoost* tem a necessidade de realizar um treinamento para cada objeto que deseja ser treinado. Portanto, foi treinado apenas para a detecção de robôs, que são o elemento fundamental para os demais experimentos (figura 26a).

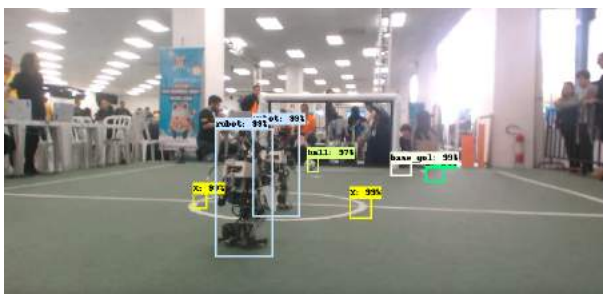
Figura 26 – Detecção das imagens utilizando detectores.

(a) Detecção realizada utilizando HAAR-*AdaBoost*.



Fonte: Do autor.

(b) Detecção realizada utilizando RNP.



Fonte: Do autor.

Na figura 26, é mostrada a detecção dos objetos utilizando cada uma das técnicas. Para realizar a detecção dos objetos, a RNP da figura 26b foi executada utilizando o computador com as configurações mostradas na tabela 7.

Tabela 7 – Descrição de hardware do computador utilizado para execução do algoritmo de RNP.

Processador	CPU Intel Core i7-5500U @ 2,40GHz 4
Memória	7,6 GiB
Sistema operacional	Ubuntu 14.04 LTS

Fonte: Do autor.

Mesmo esta configuração não sendo a ideal para a execução dos algoritmos, ela foi utilizada para se manter próxima à utilizada no trabalho realizado por Oliveira Vilão (2015). Assim, uma comparação coerente pode ser realizada. A avaliação de viabilidade será a comparação dos FPS necessários por cada uma das técnicas. Os valores medidos são mostrados na tabela 8. Os valores do HAAR- *AdaBoost* e HOG-SVM foram extraídos do trabalho de Oliveira Vilão (2015).

Tabela 8 – Resultado obtido.

Nome	***FPS***
HAAR-*AdaBoost*	17
HOG-SVM	11
RNP	5

Fonte: Do autor.

5.1.1 Resultado

Como se vê na tabela 8, o resultado da RNP é inferior em relação ao tempo de processamento quando comparado com as demais técnicas. Todavia, como é visto na figura 26b, esta técnica detectou todos os tipos de objetos em uma imagem, diferentemente das demais técnicas que detectam apenas uma classe de objetos.

Para se realizar a detecção de todos os objetos, uma cadeia de classificadores deveria ser aplicada para que todos os tipos de objetos possam ser classificados. Utilizando o HAAR-*AdaBoost*, teria-se uma detecção a cada 58,82 milissegundos. Atualmente tem-se a necessidade de detectar 8 objetos, o que levaria no total $8 \times 58{,}82 = 470{,}56$ milissegundos ou 2 FPS.

Outra maneira de se realizar a detecção é utilizar o paralelismo. Todavia, esse paralelismo é custoso, pois cada *thread* iniciada reserva para si uma quantidade de processamento.

Neste caso, oito *threads* realizando processamentos simultaneamente, podendo prejudicar o desempenho dos demais processos da arquitetura.

Com isso, optou-se por utilizar a RNP para a detecção dos objetos, pois esta requer apenas um processamento para detectar todos os objetos. Tem-se a possibilidade de se realizar esta detecção utilizando uma placa de vídeo, o que disponibiliza mais processamento para os demais processos.

5.2 EXPERIMENTO OCLUSÃO

Nesse experimento tem-se um robô andando em linha reta pelo campo de futebol com uma velocidade média de aproximadamente 10 cm/s, enquanto outro robô observa esse movimento utilizando a memória virtual para realizar o rastreamento.

Figura 27 – Estrutura montada para realização deste experimento.

Fonte: Do autor.

Na figura 27, é possível ver à esquerda o robô observador. Ele foi posicionado na marca do pênalti olhando para frente. Este é o robô que está utilizando a nova arquitetura. À direita tem-se o robô que realiza o movimento sobre a linha do meio de campo.

Dois pontos principais que foram analisados. O primeiro deles é uma aparição rápida para o observador, seguida de uma oclusão. Para essa situação, a intenção é analisar o comportamento do sistema em situações de detecção rápida, como em uma varredura. Isso foi realizado posicionando um objeto para realizar uma oclusão poucos instantes depois de o robô ser observado. A figura 28 mostra alguns instantes no tempo para exemplificação.

O segundo ponto é uma observação mais demorada da cena, na qual o sistema de memória tem uma condição melhor para realizar uma observação e aperfeiçoar a previsão. No instante final do experimento, o robô que está realizando o movimento sai do campo de visão do observador e continua seu movimento. Nesta situação específica, o objetivo é testar a predi-

Figura 28 – Instantes seguidos no tempo para mostrar a oclusão realizada.

(a) Instante inicial onde se tem a primeira aparição do robô pela visão do observador.

Fonte: Do autor.

(b) A distância percorrida no instante anterior não foi detectada ao robô ou informada.

Fonte: Do autor.

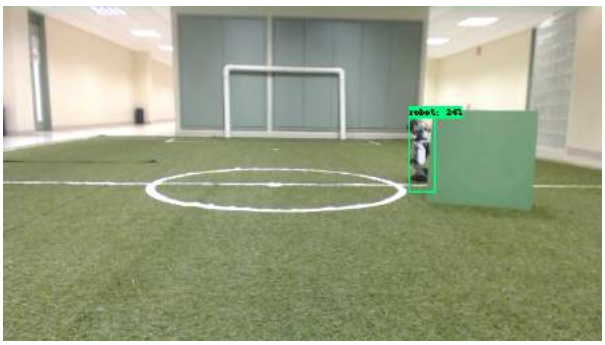
(c) Neste instante o observador não consegue detectar o robô e começa realizar uma predição.

Fonte: Do autor.

(d) O robô continua seu movimento atrás da barreira.

Fonte: Do autor.

(e) Quando o robô ressurge do outro lado a predição já se encerrou e por isso um novo período de predição se inicia.



Fonte: Do autor.

(f) Neste instante o robô reaparece sempre tentando manter a velocidade e o caminho na linha.

Fonte: Do autor.

ção do sistema depois de um longo período de detecção do objeto, no caso o robô. Essa situação é mostrada na figura 29.

Foram realizadas dez experiências com os robôs, seguindo essas mesmas características. Na figura 30 são mostrados no tempo a posição e o erro estimados pelo filtro de Kalman, e a posição estimada pela velocidade.

Figura 29 – Instantes seguidos no tempo para mostrar o período de perda e predição.

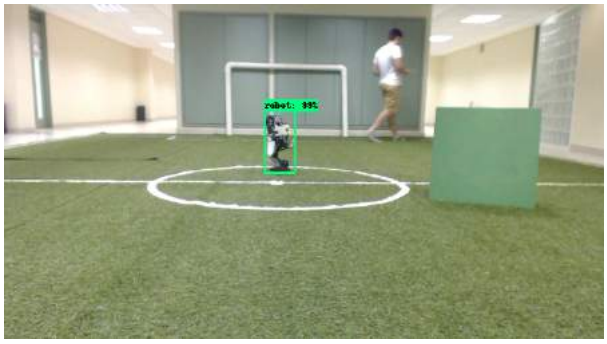
(a) Apos o instante da figura 28e se tem um período grande de observação e atualização.

Fonte: Do autor.

(b) Nesse instante o robô continua sua trajetória no decorrer da linha.

Fonte: Do autor.

(c) Neste instante já se tem uma predição boa da velocidade do robô e sobre sua posição



Fonte: Do autor.

(d) Observemos que o robô não consegue andar exatamente sobre a linha se faz necessário correções durante o percurso.

Fonte: Do autor.

(e) O observador já perdeu o robô de vista e agora se utiliza da predição para determinar onde o robô se encontra

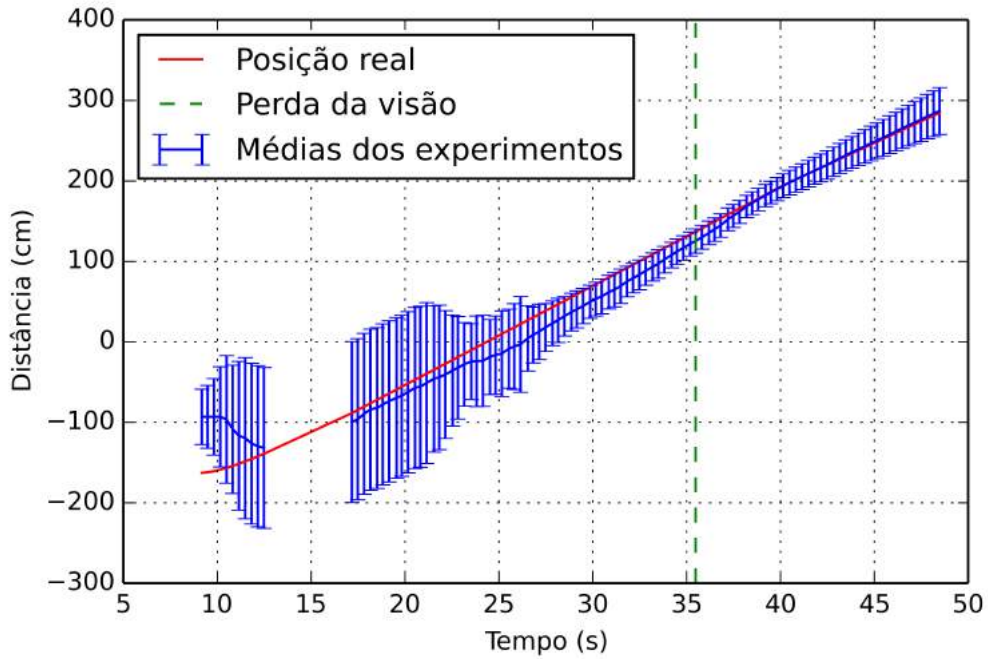
Fonte: Do autor.

(f) O robô continua com seu caminho até o final do campo mantendo sua velocidade constante.

Fonte: Do autor.

Inicialmente, não se tem uma detecção dos robôs, e consequentemente não se tem dados para a predição. Na primeira situação, com o surgimento do robô em aproximadamente 9,18 segundos, uma atualização começa a ser realizada. Entretanto, como se tem um intervalo muito curto para a detecção do robô seguida de uma oclusão, o sistema logo perde o robô e deixa de fazer a predição, pois o erro se torna três vezes maior que o erro de visão. Após essa

Figura 30 – Gráfico de rastreamento do robô no eixo Y em função do tempo.



Fonte: Do autor.

oclusão, o robô é novamente detectado e uma nova predição se inicia. Com um tempo maior para observação, o erro estimado pelo filtro começa a diminuir. O robô tende a sumir do campo de visão por volta de 35,49 segundos e nesse mesmo instante vê-se novamente um aumento no erro estimado.

O erro entre o valor estimado utilizando o tempo de vídeo e o sistema visual é mostrado na figura 31. Observe neste gráfico que o erro entre a possível posição real do objeto e a estimada é baixo quando comparado com o erro de visão. Na média, o erro se manteve em $26{,}6 \pm 12{,}87cm$.

5.2.1 Resultados

Vê-se que o sistema apresentou o desempenho esperado para o aspecto de oclusão. Quando se tem um surgimento rápido seguido de uma oclusão, não é possível localizar o objeto com precisão e por um tempo prolongado. Essa característica é explicada pelo fato de o Kalman ser um filtro probabilístico. Filtros probabilísticos necessitam de algumas observações para uma melhor predição do modelo (KALMAN, 1960). Contudo, em uma varredura já é possível realizar a detecção. Quando localizado o objetivo, o sistema de memória tem condições de direcionar a câmera para a possível região do objeto, caso ache interessante.

Quando há um novo ciclo de detecção, vê-se uma predição muito próxima da realidade, um erro na média de 26,6 cm, inclusive nas variáveis não-observáveis, como a velocidade. Isso pode ser notado na figura 30, na qual os ângulos das retas são muito próximos. Após os 35,49

Figura 31 – Gráfico de rastreamento do robô no eixo Y em função do tempo.

Fonte: Do autor.

segundos não se tem mais a detecção do robô, porém diferentemente da primeira situação, tem-se uma predição mais refinada, o que permite manter a informação do objeto por mais tempo e com maior qualidade.

Vale notar na figura 31 a queda do erro ao longo do tempo e o valor médio do erro, que é de 26,6 cm, valor menor do que os 50 cm de erro dados pelo sistema de visão. Isso é algo muito positivo, pois se tem um aumento na precisão da informação para o processo de decisão.

Para a oclusão nesse primeiro experimento, foi utilizado um objeto não detectável. Um problema usual em sistemas de rastreamento é que ao se ter uma oclusão de um objeto sobre o outro, ocorre a movimentação dos dois objetos durante o período da oclusão. Essa situação foi testada no experimento seguinte.

5.3 EXPERIMENTO OCLUSÃO ENTRE OBJETOS SEMELHANTES

Para esse experimento, tem-se dois robôs com as mesmas características. O primeiro deles se mantém parado de frente para o observador e o segundo, andando de pelo campo. Por serem objetos semelhantes, as mesmas informações de características são transmitidas para o sistema de memória, como qual o objeto detectado, no caso dos robôs, e a qual time pertence. Essa estrutura pode ser vista na figura 32.

Para que a memória possa, inicialmente, diferir um objeto do outro, os robôs são mostrados em posições diferentes (figura 33a). Nesse instante, ela deve ser capaz de diferenciar os dois objetos e extrair as características intrínsecas de cada um deles, pois com o passar do

Figura 32 – Estrutura preparada para experimento 3.

Fonte: Do autor.

tempo, o robô que está andando se aproxima e oclui o primeiro. Mas, como é possível notar na figura 33d, existe uma diferença de distância entres eles. Nesta situação, o sistema de memória deve saber qual robô está sendo observado, para que neste período de oclusão apenas o robô que está de fato se movimentando venha a continuar o movimento, o robô parado se mantenha na posição predita anteriormente, e após o período de oclusão, os robôs voltem a ser detectados e suas informações atualizadas (figura 33e).

Na figura 34 é mostrada no tempo a posição em Y e o erro estimado pelo filtro de Kalman.

Durante a detecção, é possível notar que em todos os instantes no tempo o erro em ambos os robôs tende a diminuir. Entretanto, por volta dos 27 segundos acontece a oclusão do robô fixo pelo móvel. Nesse instante, é possível notar que no robô móvel o erro tende a aumentar (figura 34b), e uma pequena correção na posição é realizada. Já no robô móvel, no qual ainda é possível realizar a detecção do erro, tende a se manter, e ao se encerrar a oclusão, os erros voltam a diminuir. No caso do robô móvel, ela volta a ter seu aumento nos 35,03 segundos, pois novamente o robô sai do campo de visão e utiliza-se apenas a predição do objeto.

Outra característica notada neste experimento é a variação de $1{,}17 \pm 0{,}06$ cm, apresentada na figura 35b. Essa variação é referente ao valor médio do estimado para o robô fixo. Para o robô móvel, a variação se manteve similar ao experimento anterior, com uma breve perturbação aos 27 segundos, referente à oclusão.

5.3.1 Resultados

Neste experimento é possível notar que, inicialmente, o sistema de memória visual foi capaz de realizar a diferenciação de objetos idênticos diferenciados apenas por suas posições. Quando se teve a oclusão do robô, houve uma perturbação, que é possível se notar na figura

35. Entretanto, as características dos robôs, como a velocidade, não sofreram alterações, e isso manteve uma boa predição neste período para ambos os robôs.

Figura 33 – Instantes seguidos no tempo para mostrar a oclusão realizada.

(a) O robô a frente do observador se mantém fixo em todo o instante.

Fonte: Do autor.

(b) O robô que realiza o movimento com a mesma velocidade dos experimentos anteriores.

Fonte: Do autor.

(c) Para o observador ocorre a oclusão dos robôs tornando possível detectar apenas o robô que realiza o movimento.

Fonte: Do autor.

(d) Neste instante é possível notar que a distancia entre os robôs são diferentes.

Fonte: Do autor.

(e) Apos a oclusão o observador volta a detectar ambos os robôs e continua sua atualização.

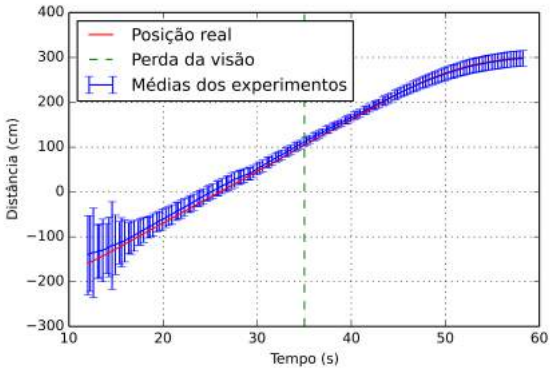
Fonte: Do autor.

(f) O robô continua realizando seu movimento sobre a linha até o final do campo.

Fonte: Do autor.

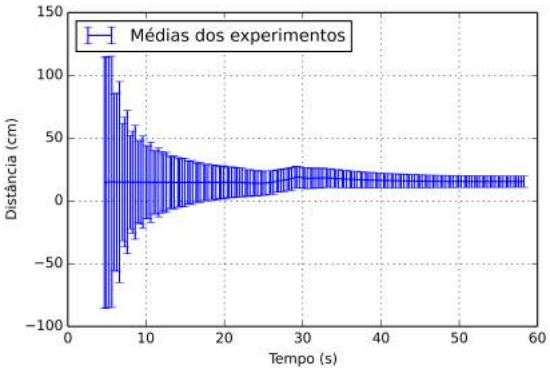
Figura 34 – Gráficos de posição para experimento 2.

(a) Posição Y do robô que realiza o movimento.



Fonte: Do autor.

(b) Posição Y do robô parado a frente do observador.



Fonte: Do autor.

Figura 35 – Gráficos dos erros para experimento 2.

(a) Erro do robô que realiza o movimento.

Fonte: Do autor.

(b) Erro do robô parado a frente do observador com relação a média estimada.

Fonte: Do autor.

6 CONCLUSÃO

Atualmente, os robôs móveis autônomos realizam diversas tarefas, para as quais na grande maioria dos casos, têm a necessidade de detectar múltiplos objetos, podendo estes serem estáticos ou móveis. Neste trabalho, essa necessidade foi detectada no futebol da *RoboCup* na categoria humanoide. Foi mostrado um modelo criado para solucionar a detecção de múltiplos objetos e realizar o rastreamento de objetos que podem ser tanto estáticos (*landmark*) quanto dinâmicos (bola e robôs).

Foi mostrado um modelo de visão capaz de detectar múltiplos objetos utilizando RNP e uma arquitetura de *threads*. Mesmo que a arquitetura apresentada utilize apenas uma câmera, este modelo pode ser usado agregando outros sensores como laser, sonar, etc. Para esse modelo, obteve-se uma alta confiabilidade na detecção, devido à utilização da RNP, e consequentemente, uma detecção simultânea rápida para diversos tipos de objetos. A arquitetura de *threads* possibilitou o tratamento dessas informações também de forma simultânea, o que trouxe uma velocidade maior na detecção dos objetos e um desempenho maior para o sistema de visão.

Como já foi dito, a detecção de um objeto é um ponto importante para a interação do robô com o ambiente, porém para que o robô possa tomar decisões mais precisas, há a necessidade de realizar o rastreamento de objeto, e no modelo proposto foi criado um processo que tem como objetivo realizar esse rastreamento. Para ele deu-se o nome de "memória visual". De forma semelhante ao sistema de visão, ela também utiliza uma arquitetura de *threads* para realizar suas operações, porém diferentemente da visão, utiliza uma *thread* para cada objeto que está sendo rastreado. Ao receber informações do sistema visual, essas informações são repassadas para as *threads* com as quais se tem maior relação, e essas *threads* têm como função manter atualizada a informação deste objeto no *blackboard*, para que a decisão ou demais processos possam se utilizar dela para montar uma estratégia.

O primeiro experimento realiza uma simples comparação do desempenho que se tem com a utilização da RNP para a detecção dos objetos com as técnicas já utilizadas. Como foi mostrado no capítulo de teoria, a RNP tem uma acurácia muito elevada com relação ao processamento. Mesmo sendo possível a utilização da técnica sem o uso de GPU, seu uso é fortemente recomendado, pois assim é possível liberar processamento na CPU e otimizar a performance de detecção.

Para validar a arquitetura proposta foram realizados três experimentos. No segundo (capítulo 5.2), em específico, foi possível testar uma situação que ocorre com frequência: um objeto ser rapidamente detectado durante uma varredura ou ter constantes falhas de detecção devido ao fato de não se ter muitos exemplos de treinamento. Nestes casos, o sistema deve manter o objeto detectado na memória por um tempo considerável, e caso não haja uma nova detecção, que venha a ser esquecida. Com isso, garante-se que o sistema tenha a capacidade de manter objetos na memória, mesmo com uma detecção com muitos falsos negativos, e para o

caso de uma movimentação rápida de cabeça, o objetivo pode ser observado, e se necessário, iniciar um rastreamento da visão.

Outro ponto muito importante que deve ser observado foi o fato de conseguir-se realizar o rastreamento de objeto por um longo período de tempo e extrair informações características que condizem com a realidade medida. Estimar a velocidade do robô observado trouxe a possibilidade de inicializar um rastreamento com maior precisão, pois com essa informação, foi possível realizar uma previsão de melhor qualidade, e com isso estimar a posição do objeto, mesmo quando não se tem uma observação do mesmo.

Para o terceiro experimento (capítulo 5.3), tem-se a situação de detectar dois objetos semelhantes, no caso, dois robôs adversários. O que se desejou validar nesta situação foi a condição de detectar dois objetos semelhantes sem confundir as informações que pertencem a cada um. Uma situação que leva a esse tipo de acontecimento em sistemas de rastreamento é a oclusão destes objetos. No experimento, criar esta condição de oclusão entre objetos semelhantes e avaliar o sistema se mostrou robusto, pois mesmo quando houve a oclusão do objeto, houve também um pequeno aumento no erro estimado. Porém, as posições tanto do objeto ainda observado quanto do objeto ocluído se mantiveram constantes. Isso é estritamente fundamental em um sistema de rastreamento e também para o sistema de memória visual, pois assim como na biologia, essa informação armazenada é importante para uma tomada de decisão, seja ela de locomoção ou estratégia.

Com esses experimentos, foi possível validar o sistema proposto e concluir que ele conseguiu solucionar o problema de falha na detecção de objeto e a perda deste objeto. O modelo proposto que neste trabalho foi aplicado ao robô humanoide da FEI pode ser adaptado para outros sistemas de robôs móveis que necessitem se localizar no ambiente e interagir, pois a memória permite melhor aproveitamento das informações vindas de outros sensores, neste caso, a câmera. Uma contribuição que se tem é a utilização de uma RNP para se detectar *landmarks*. Neste trabalho, foi mostrada no capítulo 5.1, uma rede que realiza esta detecção.

Para o RoboFEI, também foi possível aumentar a complexidade das estratégias que podem ser tomadas, pois mais informações dos objetos agora são detectadas e podem ser melhor aproveitadas. Para dar continuidade na avaliação e validação do modelo, uma interação com módulo de localização deve ser implementada, pois como mostrado no capítulo 4.2.2.2, faz-se necessário ter a localização para que seja possível predizer a velocidade do observador, e assim, avaliar o modelo quando se tem o observador e objeto observado em movimento. Também deve ser realizado um estudo mais aprofundado sobre qual deve ser a rede mais apropriada. Além da rede proposta por Howard et al. (2017), existem outras que devem ser testadas e suas topologias alteradas para que seu desempenho possa ser melhorado e se possível, eliminar a dependência da GPU.

REFERÊNCIAS

[S.l.]. **Scoring recognizability of faces for security applications**. [S.l.: s.n.], 2014. v. 9024. p. 90240l-90240l-10. Disponível em: <http://dx.doi.org/10.1117/12.2041250>.

ÇELIK, Buluç. **S-Loc and My Environment: A new Localization System for Autonomous Robots**. 2013. Diss. (Mestrado) – Boğaziçi University.

MEMÓRIA Visual a Curto Prazo - Habilidade Cognitiva. CogniFit. Fev. 2017. Disponível em: <https://www.cognifit.com/br/habilidade-cognitiva/memoria-visual>. Acesso em: 3 fev. 2017.

COIFMAN, Benjamin et al. A real-time computer vision system for vehicle tracking and traffic surveillance. **Transportation Research Part C: Emerging Technologies**, v. 6, n. 4, p. 271–288, 1998.

CORTES, Corinna; VAPNIK, Vladimir. Support-vector networks. **Machine Learning**, v. 20, n. 3, p. 273–297, 1995. Disponível em: <http://dx.doi.org/10.1007/BF00994018>.

DEGUCHI, K.; KAWANAKA, O.; OKATANI, T. Object tracking by the mean-shift of regional color distribution combined with the pARTICLE-filter algorithms. In: PROCEEDINGS of the 17th International Conference on Pattern Recognition, 2004. ICPR 2004. [S.l.]: IEEE, 2004. 506–509 vol.3. Disponível em: <http://ieeexplore.ieee.org/document/1334577/>.

FÉDÉRATION INTERNATIONALE DE FOOTBALL ASSOCIATION. [S.l.: s.n.], jan. 2017. Disponível em: <http://www.fifa.com/>.

FORSYTH, David A.; PONCE, Jean. **Computer Vision: A Modern Approach**. Upper Saddle River, New Jersey 07458: Prentice Hall Professional Technical Reference, 2002.

GERNDT, Reinhard et al. Humanoid robots in soccer: Robots versus humans in RoboCup 2050. **IEEE Robotics & Automation Magazine**, IEEE, v. 22, n. 3, p. 147–154, 2015.

GONZALEZ, R.C.; WOODS, R.E. **PROCESSAMENTO DIGITAL DE IMAGENS**. Upper Saddle River, New Jersey 07458: ADDISON WESLEY BRA. Disponível em: <https://books.google.com.br/books?id%20=%20r5f0RgAACAAJ>.

HAAR, Alfred. Zur Theorie der orthogonalen Funktionensysteme. **Mathematische Annalen**, v. 69, n. 3, p. 331–371, 1910. Disponível em: <http://dx.doi.org/10.1007/BF01456326>.

HECK, Larry P. et al. Robustness to telephone handset distortion in speaker recognition by discriminative feature design. **Speech Communication**, v. 31, n. 2, p. 181–192, 2000. Disponível em: <http://www.sciencedirect.com/science/article/pii/S0167639399000771>.

HINTON, G. et al. Deep Neural Networks for Acoustic Modeling in Speech Recognition: The Shared Views of Four Research Groups. **IEEE Signal Processing Magazine**, v. 29, n. 6, p. 82–97, nov. 2012.

HOIEM, Derek; EFROS, Alexei A.; HEBERT, Martial. Putting Objects in Perspective. **International Journal of Computer Vision**, v. 80, n. 1, p. 3–15, 2008. Disponível em: <http://dx.doi.org/10.1007/s11263-008-0137-5>.

HOWARD, Andrew G. et al. MobileNets: Efficient Convolutional Neural Networks for Mobile Vision Applications. **CoRR**, abs/1704.04861, 2017. arXiv: 1704.04861. Disponível em: <http://arxiv.org/abs/1704.04861>.

KALMAN, Rudolph Emil. A New Approach to Linear Filtering and Prediction Problems. **Transactions of the ASME–Journal of Basic Engineering**, v. 82, Series D, p. 35–45, 1960.

KAWAS, C.H. et al. Visual memory predicts Alzheimer's disease more than a decade before diagnosis. **Neurology**, American Academy of Neurology, v. 60, n. 7, p. 1089–1093, 2003. eprint: http://n.neurology.org/content/60/7/1089.full.pdf. Disponível em: <http://n.neurology.org/content/60/7/1089>.

KRIEGMAN, D.J.; TRIENDL, E.; BINFORD, T.O. Stereo vision and navigation in buildings for mobile robots. **IEEE Transactions on Robotics and Automation**, v. 5, n. 6, p. 792–803, 1989. Disponível em: <http://ieeexplore.ieee.org/document/88100/>.

KRIZHEVSKY, Alex; SUTSKEVER, Ilya; HINTON, Geoffrey E. Imagenet classification with deep convolutional neural networks. In: ADVANCES in neural information processing systems. [S.l.: s.n.], 2012. p. 1097–1105.

LEONARD, J.J.; DURRANT-WHYTE, H.F. Mobile robot localization by tracking geometric beacons. **IEEE Transactions on Robotics and Automation**, v. 7, n. 3, p. 376–382, jun. 1991. Disponível em: <http://ieeexplore.ieee.org/document/88147/>.

LY, O et al. Rhoban Football Club – Team Description Paper Humanoid KidSize League , Robocup 2016 Leipzig. [S.l.], 2016. Disponível em: <https://www.robocuphumanoid.org/qualification/2016/2d39c6b9b5d117d9488ec1b5cf630792e292e8c9/Rhoban_Football_Club_Humanoid_KidSize_2016_TDP.pdf>.

MARTÍN ABADI et al. **TensorFlow: Large-Scale Machine Learning on Heterogeneous Systems**. [S.l.: s.n.], 2015. Software available from tensorflow.org. Disponível em: <https://www.tensorflow.org/>.

MCCULLOCH, Warren S; PITTS, Walter. A logical calculus of the ideas immanent in nervous activity. **The bulletin of mathematical biophysics**, Springer, v. 5, n. 4, p. 115–133, 1943.

OLIVEIRA VILÃO, Claudio de. **Um sistema de visão computacional monocular para um robô móvel humanoide**. Out. 2015. Diss. (Mestrado) – Centro Universitário FEI.

PERICO, Danilo H. et al. Hardware and Software Aspects of the Design and Assembly of a New Humanoid Robot for RoboCup Soccer. In: 2014 Joint Conference on Robotics: SBR-LARS Robotics Symposium and Robocontrol. [S.l.]: IEEE, out. 2014. p. 73–78. Disponível em: <http://ieeexplore.ieee.org/document/7024259/>.

RUMELHART, David E; HINTON, Geoffrey E; WILLIAMS, Ronald J. **Learning representations by back-propagating errors**. [S.l.]: Nature Publishing Group, 1986. v. 323, p. 533.

RUSSAKOVSKY, Olga et al. ImageNet Large Scale Visual Recognition Challenge. **International Journal of Computer Vision (IJCV)**, v. 115, n. 3, p. 211–252, 2015.

SCHNEKENBURGER, Fabian et al. Detection and Localization of Features on a Soccer Field with Feedforward Fully Convolutional Neural Networks (FCNN) for the Adult-Size Humanoid Robot Sweaty, 2017.

SEEKIRCHER, Andreas; LAUE, Tim; RÖFER, Thomas. Entropy-Based Active Vision for a Humanoid Soccer Robot. In: LECTURE Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics). [S.l.]: Springer Berlin Heidelberg, 2011. 6556 LNAI. p. 1–12. Disponível em: <http://link.springer.com/10.1007/978-3-642-20217-9%7B%5C_%7D1>.

THRUN, Sebastian; BURGARD, Wolfram; FOX, Dieter. **Probabilistic robotics**. Cambridge, MA, USA: MIT press, 2005.

VIOLA, P.; JONES, M. Rapid object detection using a boosted cascade of simple features. In: PROCEEDINGS of the 2001 IEEE Computer Society Conference on Computer Vision and Pattern Recognition. CVPR 2001. [S.l.: s.n.], 2001. v. 1, i-511-i-518 vol.1.

ÍNDICE