UNIVERSIDADE FEDERAL DO ESPÍRITO SANTO

CENTRO TECNOLÓGICO

PROGRAMA DE PÓS-GRADUAÇÃO EM INFORMÁTICA

TIAGO ALVES DE OLIVEIRA

MAPEAMENTO, DETECÇÃO E RECONHECIMENTO DE SEMÁFOROS

VITÓRIA

2014

TIAGO ALVES DE OLIVEIRA

MAPEAMENTO, DETECÇÃO E RECONHECIMENTO DE SEMÁFOROS

Dissertação apresentada ao Programa de Pós-Graduação em Informática do Centro Tecnológico da Universidade Federal do Espírito Santo, como requisito parcial para obtenção do Grau de Mestre em Informática.

VITÓRIA

2014

TIAGO ALVES DE OLIVEIRA

MAPEAMENTO, DETECÇÃO E RECONHECIMENTO DE SEMÁFOROS

COMISSÃO EXAMINADORA

Prof. Dr. Alberto Ferreira De Souza

Universidade Federal do Espírito Santo

Orientador

Prof. Dr. Edilson de Aguiar

Universidade Federal do Espírito Santo

Coorientador

Prof. Dr. Thiago Oliveira dos Santos

Universidade Federal do Espírito Santo

Membro Interno

Prof. Dra. Karin Satie Komati

Instituto Federal do Espírito Santo

Membro Externo

Vitória, 27 de fevereiro de 2014.

Dedico este trabalho
à minha família e
à minha namorada Juliana Silva

AGRADECIMENTOS

Agradeço primeiramente a Deus, pois sem Ele, nada seria possível e não estaria aqui desfrutando desse momento que é tão importante para mim.

Ao meu orientador, professor Alberto Ferreira De Souza, e ao meu coorientador, professor Edilson de Aguiar, pela ajuda, pela revisão técnica deste trabalho e pelos ensinamentos que tornaram este trabalho possível.

Aos meus pais pelo esforço, dedicação e compreensão que, em todos os momentos desta e de outras caminhadas, mostraram-me o caminho certo e me apoiaram em tudo.

Aos amigos do LCAD por sua confiança em minha pessoa, pelo mútuo aprendizado de vida durante nossa convivência, no campo profissional e escolar. Amigos, gratidão eterna!

A CAPES e à FAPES pelo apoio financeiro na forma de bolsas de estudo.

A todos aqueles que, direta ou indiretamente, colaboraram para que este trabalho atingisse aos objetivos propostos.

RESUMO

A segurança durante a condução é um tema de pesquisa muito importante para a indústria automotiva. Uma das tecnologias que podem tornar os carros mais seguros para dirigir é a de detecção automática da presença e o reconhecimento automático do estado de semáforos. Em situações do mundo real, dada à variação de aparência substancial, por exemplo, devido a diferentes condições de luz, clima, mudanças de ponto de vista, formatos diferentes, falta de padronização, o envelhecimento do semáforo e até deformações, métodos robustos são necessários.

Neste trabalho investigamos e implementamos métodos computacionais para mapeamento, detecção e o reconhecimento do estado de semáforos. Nós empregamos o algoritmo Viola-Jones para a detecção de semáforos em imagens capturadas on-line por câmeras, e *Support Vector Machines* (SVM) para o reconhecimento do estado dos mesmos. Para obtenção destas imagens, nós utilizamos o veículo autônomo desenvolvido na UFES denominado IARA – *Intelligent Autonomous Robotic Automobile*.

Além de identificar os semáforos em imagens capturadas pela IARA, é conveniente armazenar suas coordenadas geográficas em mapas para aumentar a confiabilidade da detecção e do reconhecimento do estado dos mesmos no futuro, quando a IARA voltar a encontra-los. Assim, foi desenvolvido uma metodologia e sistemas de software para inserir a pose de semáforos em mapas de grid para veículos autônomos de forma semiautomática e mecanismos para localizar os semáforos em mapas previamente existentes automaticamente.

O sistema proposto foi integrado à IARA, sendo capaz de identificar com grande precisão semáforos em distâncias variando entre 15 e 80 metros, em um tempo de aproximadamente 0,035 segundos por imagem, ou seja, em tempo real. O detector de semáforos obteve taxas de 97% de precisão e 52% de revocação. O reconhecedor de estados alcançou uma taxa de 96,5% de precisão. O percentual dos semáforos cujo estado foi corretamente determinado dentre aqueles detectados foi igual a 96,3%, o que demonstra um bom desempenho do sistema.

ABSTRACT

Safety while driving is very important research topic for the automotive industry. One of the technologies that can make cars safer to drive is the automatic detection of traffic lights and the recognition of their state. In real world situations, given the substantial variation in appearance, for instance due to different conditions of light, weather, changes in viewpoint, different shapes and formats, lack of standardization, aging and even deformations of traffic lights, robust methods are needed.

In this dissertation, we investigate and implement computational methods for mapping, detection and recognition of the state of traffic lights. First, we employ the Viola-Jones algorithm for detecting traffic lights from images captured by online cameras. Thereafter, Support Vector Machines (SVM) is used for the recognition of the state of each traffic light. To obtain the input images, we used the autonomous vehicle developed at the Federal University of Espírito Santo (UFES) named IARA - Intelligent Autonomous Robotic Automobile.

In addition to identifying the traffic lights in images captured by the IARA, it is convenient to store their geographical coordinates on maps to increase the reliability of detecting and recognizing them in the future. Therefore, we developed a methodology and software to semiautomatic insert the pose of traffic lights into maps for helping autonomous driving, as well as ways to find the traffic lights automatically on existing maps.

The proposed system is integrated to IARA, being able to identify with good accuracy traffic lights at distances between 15 and 80 meters in real-time, e.g. approximately 0.035 seconds per frame. The traffic light detector has an accuracy of 97% with 52% of recall. The recognizer reaches an accuracy rate of 96.5%. The percentage of traffic lights whose status was correctly determined among those detected was equal to 96.3%, which shows a good performance of the system.

LISTA DE FIGURAS

Sumário

1 INTRODUÇÃO

A segurança durante a condução é um tema de pesquisa muito importante para a indústria automotiva. Uma tecnologia que poderia tornar os carros mais seguros é a detecção e o reconhecimento automáticos do estado de semáforos. Essa tecnologia tem como objetivo alertar condutores desatentos para a mudança do estado de semáforos à sua frente e fazer com que os cruzamentos sejam mais seguros. Outro objetivo importante é a viabilização de sistemas de condução autônoma de veículos.

Uma alternativa para tal tecnologia seriam semáforos não passivos, isto é, semáforos que transmitem sua posição e estado aos automóveis próximos [1]. No entanto, tais sistemas ativos requerem mudanças significativas no hardware dos semáforos e dos veículos atuais; assim, não acreditamos que venham a ser implementados em larga escala em um futuro próximo.

A detecção e o reconhecimento automáticos de semáforos por meio de câmeras seriam em muito simplificada se a localização, tamanho e formato dos mesmos fossem precisamente definidos. Embora vários destes parâmetros sejam precisamente especificados em leis e normas em praticamente todos os países, em situações do mundo real, dado a variação substancial de sua aparência devido, por exemplo, à falta de padronização, diferentes condições de luz, clima, mudanças de ponto de vista, deformações e até do envelhecimento do semáforo, abordagens automáticas simples não são confiáveis e métodos mais robustos são necessários.

Seres humanos são capazes de detectar e reconhecer uma grande variedade de semáforos existentes de forma eficiente; contudo, para os sistemas automáticos, isso ainda é um desafio. No entanto, devido à alta relevância industrial, a detecção automática de semáforos e o reconhecimento do seu estado vêm atraindo a atenção de muitos pesquisadores nos últimos anos [2] [3] [4] [5] [6] [7] [8] [9] [10] [11]. Apesar de avanços recentes, a detecção e o reconhecimento do estado dos semáforos ainda é um problema complexo do mundo real que merece investigação. Neste trabalho nós apresentamos nossos estudos sobre o tema e descrevemos os métodos que empregamos para a construção de sistemas para o mapeamento

semiautomático de semáforos, sua detecção automática e o reconhecimento automático de seu estado.

Este trabalho está inserido em uma linha de pesquisa do Departamento de Informática da UFES (Linha de Pesquisa em Ciência da Cognição) cujo objetivo de longo prazo é implementar uma arquitetura neural artificial com capacidades cognitivas[1] semelhantes, ou até mesmo equivalentes, às humanas. O grupo de pesquisa envolvido (Grupo de Pesquisa em Ciência da Cognição – GPCC) iniciou seus trabalhos pelo estudo do sentido da visão, por ser o sentido que nos fornece mais informações sobre o mundo externo. Este trabalho evoluiu para um ponto em que, hoje, o GPCC está estudando visão artificial aplicada a problemas de robótica autônoma. Neste contexto, está sendo desenvolvida a IARA – *Intelligent Autonomous Robotic Automobile* – um carro de passeio autônomo baseado no modelo Ford Escape Hybrid (Figura 1).

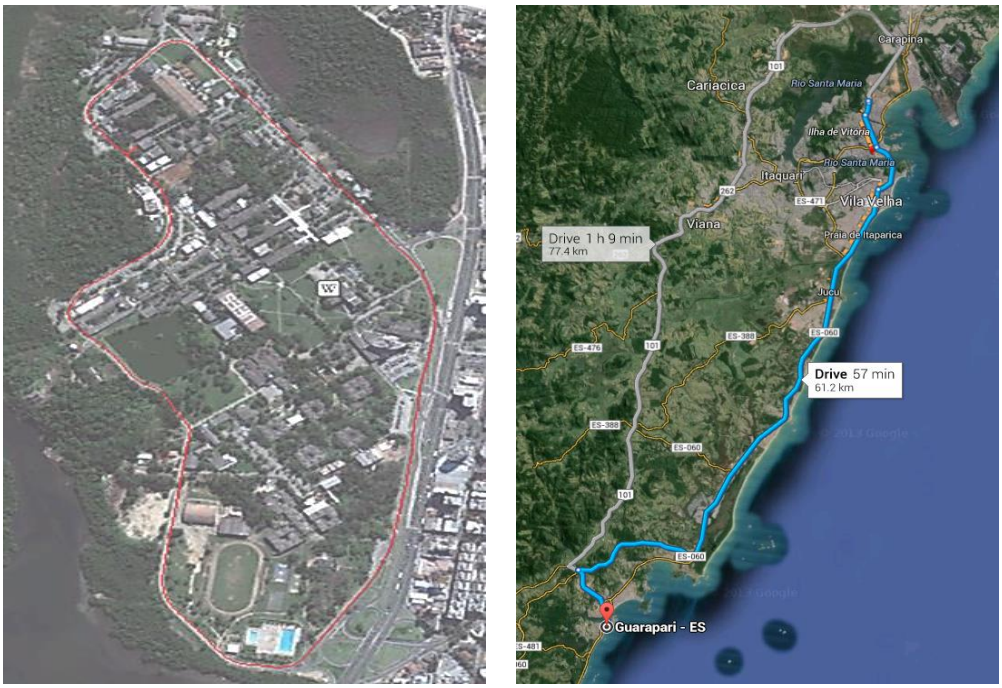
Figura 1 – IARA - *Intelligent Autonomous Robotic Automobile*.

[1] Cognição pode ser definida como a nossa capacidade de compreender o mundo e as ideias por meio dos nossos sentidos e de nossa memória de experiências passadas.

Diversos módulos de software foram e estão sendo desenvolvidos para viabilizar o funcionamento autônomo da IARA em dois cenários: a Volta da UFES e a Ida a Guarapari.

O campus principal da UFES (campus de Goiabeiras) possui um anel viário que o circunda que possui 3.570 metros (Figura 1(a)). Na Volta da UFES, o objetivo é desenvolver pesquisas que confiram à IARA as habilidades necessárias para realizar uma volta ao redor do campus de Goiabeiras da UFES de forma autônoma – este objetivo já foi, em larga margem, alcançado.

Na Volta da UFES as restrições impostas pelas leis de trânsito podem ser, em larga medida, desconsideradas, já que é possível realizá-la em um domingo ou feriado, situações em que o trânsito no anel viário é praticamente nulo. A distância da UFES à cidade de Guarapari, por outro lado, é de 58,5 Km (Figura 1(b)). Na Ida a Guarapari o objetivo é desenvolver pesquisas que viabilizem o aperfeiçoamento da IARA de modo a torná-la capaz de realizar a Ida a Guarapari autonomamente. No caminho da Ida a Guarapari todas as leis de trânsito se aplicam e terão que ser consideradas pelos algoritmos que comandarão a IARA.



(a) (b)

Figura 2 - (a) Caminho da Volta da UFES. (b) Caminho da Ida a Guarapari.

Assim, a principal motivação para este trabalho é contribuir para o alcance do objetivo de Ida a Guarapari por meio do estudo e desenvolvimento de um subsistema de detecção e reconhecimento automáticos do estado de semáforos para a IARA.

1.1 Objetivos

Um sistema de detecção e reconhecimento de semáforos útil precisa resolver os problemas de detecção e reconhecimento independentemente da rotação do semáforo, diferentes condições de iluminação, mudanças de perspectiva, formatos diferentes, problemas com as lâmpadas, oclusão e de todos os tipos de condições climáticas.

Dada uma imagem de uma cena capturada por uma câmera, o problema geral de detecção de semáforos consiste de identificar uma ou mais regiões de interesse nesta imagem que possam conter semáforos usando informações a priori sobre a forma, cor, posição das lâmpadas e características presentes nos semáforos. O problema geral de reconhecimento do estado de semáforos consiste de, dada uma região de interesse em uma imagem, avaliar as informações de iluminação e cor de modo a identificar o estado do semáforo. As soluções para estes problemas presentes na literatura geralmente envolvem a segmentação das cenas (detecção do semáforo), extração de características de regiões do semáforo e o reconhecimento do seu estado [2] [3] [4] [5] [6] [7] [8] [9] [10] [11].

Este trabalho teve como objetivo a investigação e implementação de sistemas de software baseados no algoritmo Viola-Jones [12] e em *Support Vector Machines* (SVM [13]) para a detecção e o reconhecimento do estado de semáforos.

1.2 Contribuições

As principais colaborações deste trabalho foram:

- Desenvolvimento de uma metodologia e implementação de sistemas de fácil utilização para o mapeamento semiautomático de semáforos.
- Implementação de um subsistema de detecção de semáforos baseado no algoritmo Viola-Jones.
- Implementação de um subsistema de reconhecimento do estado de semáforos baseado em SVM
- Integração dos subsistemas de detecção e reconhecimento do estado de semáforos no framework CARMEN, possibilitando o funcionamento em conjunto aos demais módulos de software da IARA.
- Análise e teste dos algoritmos em ambiente real com a IARA.
- Desenvolvimento de vários (muitos) módulos de software de apoio à pesquisa em detecção e reconhecimento de padrões em imagens, particularmente aqueles baseados em Viola-Jones e SVM.
- Criação de uma base de dados de imagens de semáforos para a pesquisa na área.

1.3 Organização do Texto

Esta dissertação está dividida da seguinte forma:

Capítulo 2 – Fundamentação Teórica: apresenta a fundamentação teórica dos algoritmos usados nesse trabalho.

Capítulo 3 – Sistema TLSDR: apresenta o problema de mapeamento, detecção e reconhecimento de estados de semáforos e uma visão geral do sistema TLSDR.

Capítulo 4 – Metodologia: apresenta o hardware, software, a base de dados e as métricas utilizados para avaliar o sistema desenvolvido nesse trabalho, o TLSDR.

Capítulo 5 – Experimentos e Resultados: apresenta os experimentos e os resultados da avaliação do sistema desenvolvido neste trabalho.

Capítulo 6 – Discussão: apresenta trabalhos correlatos e faz uma análise crítica do trabalho realizado.

Capítulo 7 – Conclusão: finaliza o trabalho apresentando a conclusão e trabalhos futuros.

2 FUNDAMENTAÇÃO TEÓRICA

Neste capítulo é apresentada a fundamentação teórica utilizada nesse trabalho. Na Seção 2.1 é apresentado o detector de objetos Viola-Jones utilizado para detectar semáforos. A Seção 2.2 apresenta o conjunto de técnicas SVM que foi utilizado para o reconhecimento do estado de semáforos

2.1 Detector de Objetos Viola-Jones

Detecção de objetos usando o classificador em cascata do tipo *Haar feature* é um método efetivo de detecção de objetos. É uma abordagem baseada em aprendizado de máquina em que uma função cascata é treinada a partir de uma grande quantidade de imagens positivas e negativas. Em seguida, é utilizado para detectar objectos em outras imagens.

Este modelo de detecção de objetos apresentado em [12] emprega um classificador que, ao invés de realizar operações diretamente sob os valores dos *pixels*, utiliza sub-janelas que percorrem a imagem com filtros. Esses filtros representam características (*features*) baseadas nos filtros de Haar, e por isso são também conhecidos como *Haarlike Features*. A Figura 3 demonstra o conjunto de filtros originalmente apresentadas em [12] e usados em testes para a detecção de faces humanas. O valor de cada filtro é dado pela soma dos *pixels* que se encontram dentro dos retângulos brancos subtraído da soma dos *pixels* dentro dos retângulos cinzas.

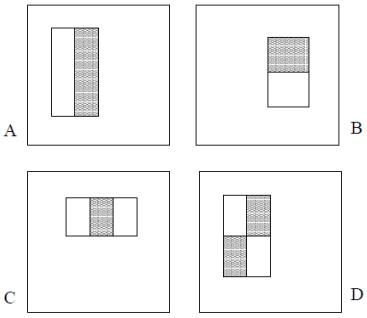


Figura 3 – Conjunto de *features* originais. Retirado de [12].

Com o intuito de melhorar as *features* foi apresentado o conjunto aperfeiçoado e estendido de filtros [14]. Foi removida a característica com quatro retângulos e introduzidas *features* rotacionadas, o que proporcionou melhores resultados em detecção. Detalhes de como calcular as características rotacionadas podem ser encontrados em [14]. A Figura 4 exemplifica as *features* estendidas.

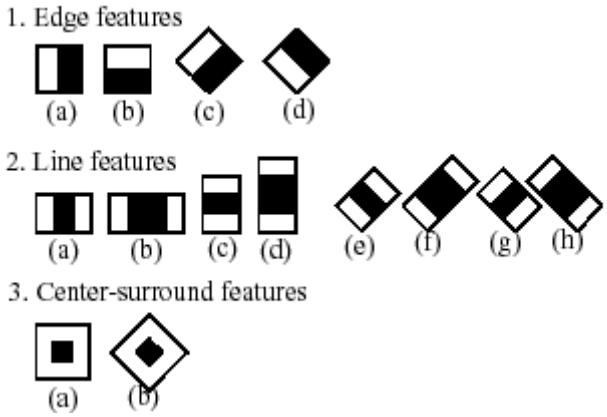


Figura 4 - Conjunto de *features* estendidas. Retirado de http://docs.opencv.org/modules/objdetect/doc/cascade_classification.html.

Com a finalidade de obter desempenho em tempo real, Viola e Jones propuseram uma representação diferente de imagem denominada de "Imagem Integral" (*Integral Image*). Essa representação pode ser conferida na Figura 5, em que ela exemplifica o conceito da Imagem Integral. Nesta imagem, a localização (x, y) contém a soma dos pixels acima e à esquerda do pixel (x, y) (ele inclusive) da imagem original, ou seja:

$$ii(x,y) = \sum_{x<x',y<y'} i(x',y') \qquad \textbf{Equação 1}$$

onde $ii(x, y)$ corresponde a imagem integral e $i(x, y)$ a imagem original. Os valores dos filtros de *haar* são dados pela diferença dos pixels dentro dos quadrados brancos e dos pretos. Utilizando a imagem integral, é possível determinar em tempo constante o valor destes filtros. Ainda na Figura 5 pode ser visto que a soma dos

pixels no retângulo D pode ser computada em quatro acessos a um vetor: o valor da imagem integral no ponto 1 é a soma dos pixels no retângulo A. Então o valor no ponto 2 é $A + B$, no ponto 3 é $A + C$, e no ponto 4 é $A + B + C + D$. A soma dos valores dentro de D é dada por 4 + 1 − (2 + 3). Do mesmo modo pode ser constatado que, uma vez que os filtros são formados por retângulos adjacentes, o valor daqueles formados por dois retângulos pode ser calculado em seis acessos a um vetor, os de três retângulos em oito acessos e os de quatro, em apenas nove acessos.

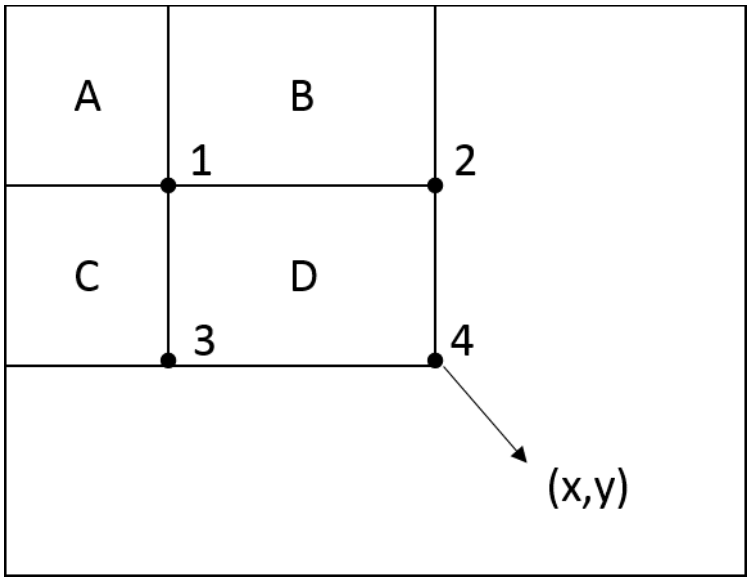


Figura 5 - Exemplo da representação de uma Imagem Integral. Adaptado de [12].

Ao invés de fazer o redimensionamento de toda a imagem por várias vezes, Viola e Jones apresentam a solução de redimensionar a janela de detecção. Uma janela de detecção usa apenas um filtro simples que facilmente pode ser redimensionado utilizando extrapolação do tamanho e posição do filtro. Ao realizar o redimensionamento de uma janela de detecção, não existe a necessidade de se utilizar uma pirâmide, o qual reduz significativamente o tempo de computação necessário.

Um ponto negativo dessa abordagem pode ser verificado em uma janela de detecção do tamanho 24x24 pixels, em que é possível escolher mais de 180000

filtros dentro de apenas uma janela. Para se computar todos os valores desses filtros para cada janela requereria um tempo de processamento excessivamente alto. Dessa forma, há a necessidade de se utilizar apenas um pequeno subgrupo de filtros mais relevantes para formar um classificador eficiente. Para isso, Viola e Jones utilizaram um algoritmo de *boosting* chamado AdaBoost [15].

Adaboost é capaz de combinar sequencialmente vários classificadores fracos em um classificador forte gerado ao final. Esses classificadores são chamados fracos devido ao fato que sozinhos eles não são capazes de classificar grande parte de um conjunto de testes, ou não se espera que sejam muito mais eficientes que o simples acaso. A cada rodada do AdaBoost, o classificador com o menor erro é escolhido entre o conjunto de classificadores fracos. Logo após, as amostras que foram classificadas de forma errada recebem um peso maior na próxima rodada. O algoritmo itera até que o número de rodadas escolhido seja atendido, aumentando o peso cada vez mais em amostras de difícil classificação. Ao final, obtém-se um classificador forte, que é uma combinação ponderada de classificadores fracos.

No método proposto por Viola e Jones, o algoritmo de AdaBoost é adaptado de forma a conter um tipo de filtro (característica) para cada classificador fraco, e ao mesmo tempo é ajustado a fim de minimizar o erro de classificação. Um classificador fraco $h_j(x)$ é definido como:

$$h_j(x) = \begin{cases} 1, se\ p_j\ f_j(x) < p_j\ \theta_j \\ 0, caso\ contrário \end{cases} \qquad \textbf{Equação 2}$$

onde f_j é o resultado do filtro na imagem de entrada x, que é uma janela de detecção do tamanho 24x24 pixels, θ_j é um limiar, p_j é uma paridade que indica a direção do sinal da desigualdade, $0 \leq j \leq J$ e J é o número total de filtros utilizados.

As maiorias das janelas possíveis de detecção não incluirão o objeto desejado, por isso, janelas não contendo objetos devem ser descartadas o mais breve possível. Assim sendo, Viola e Jones realizaram uma combinação de classificadores de forma que sejam representados em uma árvore degenerada, também conhecida como cascata de classificadores. Nesta combinação, cada nó, ou estágio, é executado sequencialmente e corresponde a um classificador AdaBoost (isto é, um

classificador forte). Nos primeiros estágios da cascata, os classificadores possuem um reduzido número de características e são, portanto computados rapidamente. Esses estágios iniciais simples são treinados de forma a rejeitar o maior número de não objetos (falsos positivos) e aceitar todos os objetos (verdadeiros positivos). Estágios subsequentes da cascata crescem em complexidade, contendo um número maior de classificadores fracos e são usados para diminuir o número de falsos positivos. A Figura 6 ilustra o processo de classificação em cascata, onde C representa uma cascata, P simboliza um resultado positivo e N é um resultado negativo. Um resultado negativo em qualquer estágio é rejeitado e o classificador passa para a próxima janela de detecção.

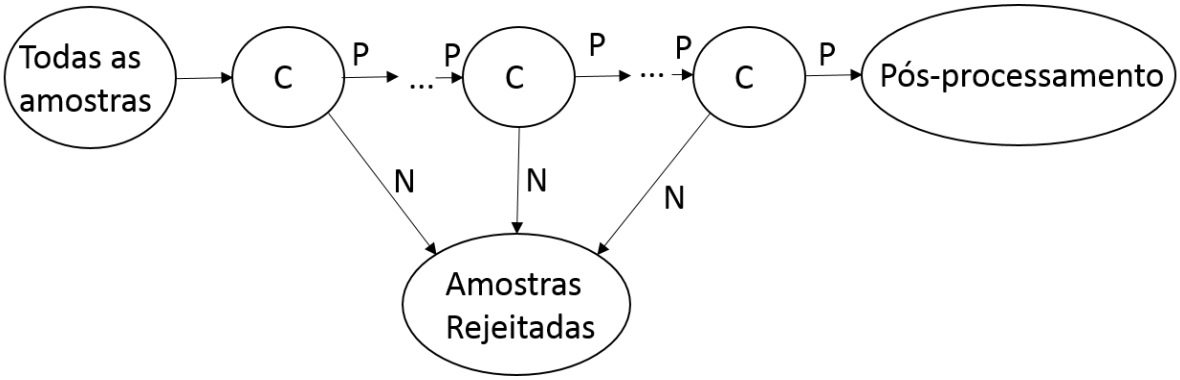


Figura 6 - Descrição esquemática do detector em cascata.

Como não são conhecidas à posição e tamanho do objeto na imagem, a busca é feita fazendo varreduras do objeto várias vezes, em tamanhos diferentes. Em cada passagem é analisada uma janela, chamada janela de busca, com tamanho diferente. Assim, os filtros selecionados no treinamento são escalonados de um tamanho mínimo até o tamanho da imagem.

O escalonamento se dá por meio de um valor chamado fator de escalonamento, um fator de escalonamento 1.1, por exemplo, faz aumentar o tamanho da janela em 10% a cada passada. Esse fator deve ser escolhido a critério do usuário: caso seja alta a eficiência em tempo do algoritmo aumenta, pois menos janelas serão processadas. Porém podem ser perdidos objetos positivos; caso o valor seja baixo, menos objetos serão perdidos, entretanto o processamento será mais lento e poderá aumentar o número de falsos objetos positivos detectados.

2.2 SVM

Support Vector Machines (SVMs) são um conjunto de métodos de aprendizado supervisionado que analisam dados e reconhecem padrões, utilizados para classificação (aprendizado de máquina) e análise de regressão. O algoritmo SVM original foi inventado por Vladimir Vapnik e a sua representação padrão atual foi proposta por Corinna Cortes e Vladimir Vapnik [13].

O SVM padrão é um classificador linear binário não probabilístico, ou seja, prevê, para cada dado de entrada, qual das duas classes possíveis à entrada é um membro. Uma vez que SVM é um classificador, então, dado um conjunto de exemplos de treinamento, com cada exemplo marcado como pertencente a uma das duas categorias, um algoritmo de treinamento SVM constrói um modelo que se prevê um novo exemplo pertence a uma categoria ou outra. Intuitivamente, um modelo SVM é uma representação dos exemplos como pontos no espaço, mapeado para que os exemplos das categorias separadas sejam divididos por uma clara lacuna que é tão grande quanto possível. Novos exemplos são mapeados para o mesmo espaço e prevê-se que pertencem a uma categoria, com base em qual lado da lacuna ele pertence.

Mais formalmente, uma máquina de vetor de suporte constrói um hiperplano ou conjunto de hiperplanos em um espaço dimensional ou espaço de dimensão infinito, que pode ser usado para classificação, regressão ou outras tarefas. Intuitivamente, uma boa separação é conseguida através do hiperplano que tem a maior distância para os pontos dos dados de treinamento mais próximos de qualquer classe (chamada margem funcional), uma vez que, em geral, quanto maior a margem inferior o erro da generalização do classificador. A Figura 7 demonstra um exemplo de classificação e um exemplo de margem máxima.

Considerando que o problema original pode ser indicado em um espaço dimensional finito, acontece frequentemente que os conjuntos de discriminação não são linearmente separáveis naquele espaço. Por este motivo, foi proposto que o espaço finito-dimensional original seja mapeado em um espaço muito mais elevado, presumivelmente facilitando a separação naquele espaço.

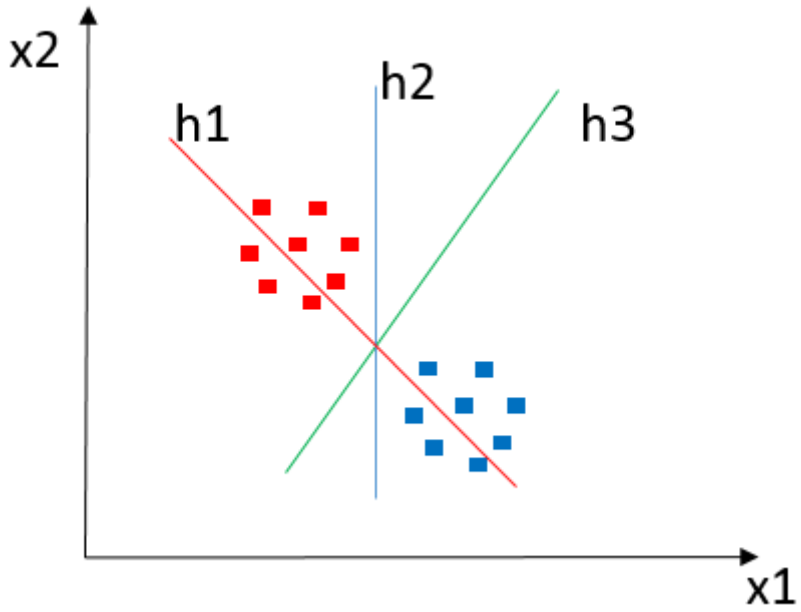


Figura 7 - Exemplo de classificação usando SVM. H1 não separa as classes. H2 faz, mas apenas com uma pequena margem. H3 separa-los com a margem máxima.

Para manter a carga computacional razoável, os mapeamentos usados por esquemas de SVM são projetados para assegurar que o produto escalar possa ser facilmente calculado em termos de variáveis no espaço original, definindo-os em termos de uma função do *kernel* $K(x, y)$ selecionado de acordo com o problema[ref]. Os hiperplanos no espaço de dimensão superior são definidos como o conjunto de pontos, cujo produto escalar com um vetor nesse espaço é constante. Os vetores definindo os hiperplanos podem ser escolhidos para serem combinações lineares com parâmetros $\propto_i$ de imagens de vetores de recurso que ocorrem na base de dados. Com esta escolha de um hiperplano, os pontos x no espaço de recurso que são mapeados para o hiperplano são definidos pela relação da Equação 3, onde C conota um valor constante.

$$\sum_i \propto_i \; K(x, y) = C \qquad \textbf{Equação 3}$$

Nota-se que se $K(x, y)$ torna-se pequeno como y cresce mais longe x, cada termo na soma mede o grau de proximidade do teste ponto a ponto x_i de base x para os

dados correspondentes. Desta forma, a soma de *kernels* acima pode ser usada para medir a proximidade relativa de cada ponto de teste para os pontos de dados originários de um ou outro dos conjuntos para ser discriminados. Observe o fato de que o conjunto de pontos x mapeado em qualquer hiperplano pode ser bastante complicado como resultado, permitindo muito mais complexa discriminação entre conjuntos que não são convexos em tudo no espaço original.

Alguns problemas de classificação binária não possuem um hiperplano simples como critério de separação útil. Para estes problemas, existe uma variante da abordagem matemática que mantém quase toda a simplicidade de um SVM separando o hiperplano. Esta abordagem utiliza estes resultados a partir da teoria da reprodução dos *kernels*:

- Existe uma classe de funções $K(x, y)$ com a seguinte propriedade: Existe um espaço linear S e uma função φ mapeando x para S de tal modo que

  $$K(x, y) = < \varphi(x), \varphi(y) >$$

  O produto vetorial ocorre no espaço S.

- Esta classe de funções inclui:
  - Polinômios: Para cada inteiro positivo d,

    $$K(x, y) = (1 + < x, y >)^d.$$

  - Função de base radial: Para cada número positivo σ,

    $$K(x, y) = exp(-< (x-y), (x - y) >/(2\sigma^2)).$$

  - Perceptron Multi Camada (rede neural): Para um número positivo p_1 e um número negativo p_2,

    $$K(x, y) = \tanh(p_1 < x, y > + p_2).$$

A abordagem matemática usando *kernels* baseia-se no método computacional dos hiperplanos. Todos os cálculos para a classificação de hiperplano usam nada mais

do que produtos vetoriais. Portanto, *kernels* não-lineares podem usar cálculos idênticos e algoritmos de solução, e obter classificadores que são não-lineares. Os classificadores resultantes são hiper superfícies em algum espaço S, mas o espaço S não tem que ser identificado ou examinado..

3 SISTEMA TLDSR

Este capítulo tem como objetivo apresentar o funcionamento do sistema TLSDR (*Traffic Light Detector and State Recognizer*). Na Seção 3.1 é definido o problema desse trabalho. Uma visão geral do sistema é apresentada na Seção 3.2. Na Seção 3.3 é apresentada a solução de mapeamento empregada no TLDSR. A solução de detecção utilizada nesse trabalho é apresentada na Seção 3.4. Por fim, a solução desenvolvida para o reconhecimento é apresentada na Seção 3.5

3.1 Definição do Problema

O problema de identificação de semáforos envolve a detecção de um ou mais semáforos e o reconhecimento do estado dos mesmos em imagens do mundo real.

Detectar um semáforo consiste na tarefa em que dada uma imagem do mundo real identificar onde existem um ou mais semáforos de trânsito. Para detectar um semáforo é necessário saber detalhes sobre a geometria, aparência, posição das lâmpadas do mesmo. Obter essas informações não é uma tarefa simples, pois existem diversos modelos e posições em que o semáforo pode se encontrar. Além disso, condições de iluminação, sol, chuva, iluminação do semáforo, deterioração, defeito e tipo de lâmpadas podem interferir na detecção de semáforos. Também podem ocorrer oclusões, como por exemplo, carros ou caminhões que podem bloquear parcialmente ou completamente a visão do semáforo.

Com um semáforo detectado, reconhecer o seu estado compreende em identificar, entre os três possíveis estados, o estado em que ele se encontra que pode ser:

- Verde: que indica que o cruzamento está livre para passagem;
- Amarelo: que indica que a passagem está prestes a ser fechada;
- Vermelho: indica que a passagem pelo cruzamento está momentaneamente impedida.

Mas a incidência do sol, depreciação do semáforo, lâmpadas queimadas, defeito no semáforo, oclusão, lâmpadas que não sejam de LED (*light emitting diode* - diodo emissor de luz) faz com que o estado possa ser detectado erroneamente. Detectar

um semáforo vermelho como sendo verde pode causar graves acidentes. Sendo assim, ter uma precisão alta de reconhecimento é vital em um sistema de veículo autônomo.

Uma forma de obter informações previa dos semáforos é utilizando o mapeamento de semáforos. Mapeamento de semáforos consiste em ter conhecimento de quando e onde um semáforo vai aparecer na imagem, reduzindo assim a área de busca na imagem para a detecção e tendo uma janela de possíveis lugares onde o semáforo vai aparecer.

3.2 Visão Geral do Sistema

O TLSDR é um detector e reconhecedor de estados de semáforos. O TLSDR recebe como entrada imagens capturadas de uma câmera e as anotações manuais dos semáforos e gera como saída uma lista de regiões de interesse (ROI) que contém os semáforos detectados, à distância até os semáforos detectados e o estado dos semáforos detectados. O TLSDR é formado pelos seguintes passos (Figura 8): Detector, Reconhecedor e Medidor de Distância.

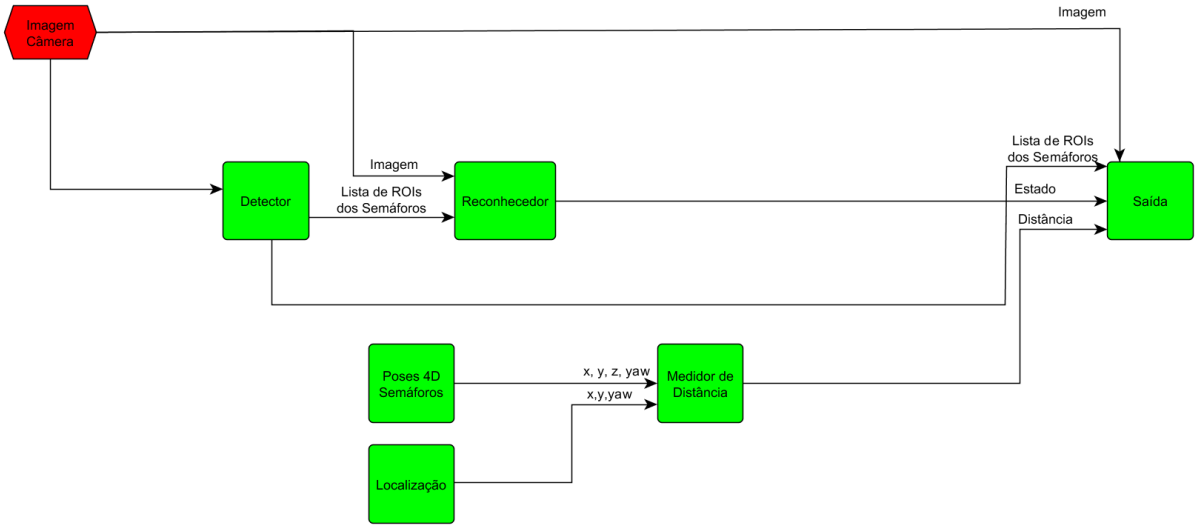


Figura 8 - Visão Geral do sistema TLDSR.

O detector recebe uma imagem capturada de uma câmera e detecta na imagem, caso existam, um ou mais semáforos e retorna a região de interesse dos mesmos para o reconhecedor de estados. O reconhecedor de estados recebe a lista com a região de interesse que contém os semáforos e a imagem capturada e reconhece os estados dos semáforos a partir do recorte da região de interesse da lista na imagem e ao final, ele retorna quais são os estados nos semáforos detectados. O medidor de distância recebe as anotações manuais feitas dos semáforos e uma localização do veículo autônomo e com esses valores calcula a distância até o semáforo mais próximo, retornando essa distância.

Com as saídas do detector, do reconhecedor e do medidor de distância a imagem de entrada é editada de forma a demonstrar se existem ou não semáforos na imagem. Caso existam, os seus contornos são desenhados na imagem de forma que são realçados na cor em que o estado do semáforo foi reconhecido. Além disso, a distância estimada até o semáforo é anotada na imagem.

3.3 Mapeamento

O medidor de distância do TLDSR possui o seguinte bloco (Figura 9) Medidor de Distância. O medidor de distância recebe uma localização (pose) de onde um veículo autônomo está e uma lista contendo as poses e orientações de semáforos anotados anteriormente. Com essas duas entradas, ele calcula qual a menor distância até um semáforo e determina se a orientação dele é a mesma da anotada, fornecendo como saída à distância, em metros, do veículo até o semáforo.

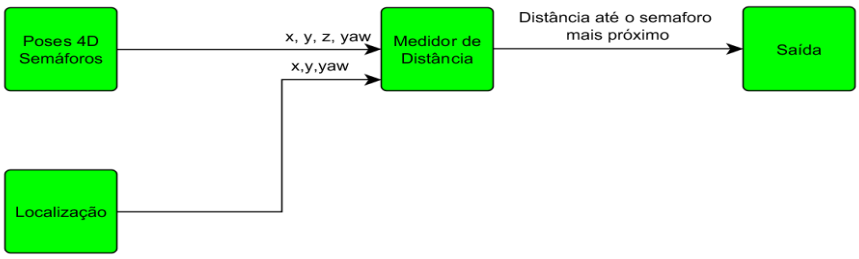


Figura 9 - Diagrama do mapeamento do TSLDR.

3.4 Detecção

O detector do TLDSR possui os seguintes blocos (Figura 10): Recortador e Detector. O Recortador (usado somente para validar e testar o sistema) recebe uma imagem da câmera e realiza um recorte na imagem e repassa para o Detector. O Detector recebe a imagem recortada (caso esteja testando ou validando o sistema) ou a imagem natural (para o treinamento) e retorna a lista de regiões de interesses (ROIs).

Figura 10 - Diagrama do detector do TLDSR.

3.5 Reconhecimento

O reconhecedor do TLDSR possui os seguintes blocos (Figura 11): Reescalador, Pré Processador e o Reconhecedor. O Reescalador recebe a lista dos ROIs dos semáforos detectados pelo nosso Detector e a imagem capturada de uma câmera e realiza o reescalamento dos semáforos detectados na imagem recebida para o tamanho de 9x20 *pixels* e repassa essa imagem para o Pré Processador. O Pré Processador recebe a imagem reescalada, realiza várias diversas transformações nesta imagem e retorna a imagem pré-processada para o Reconhecedor. O reconhecedor então recebe a imagem pré-processada e calcula o estado para cada imagem recebida.

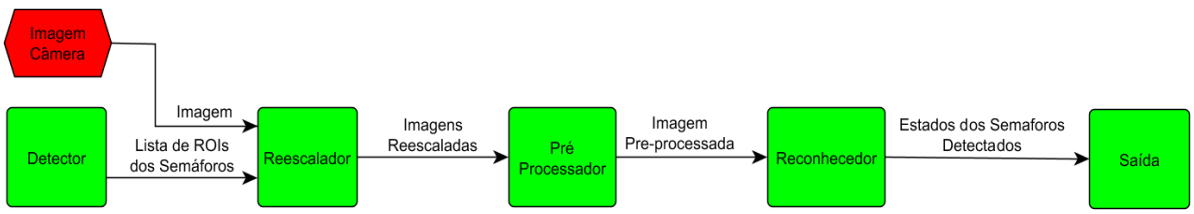


Figura 11 - Diagrama do reconhecedor do estado de semáforos.

4 METODOLOGIA

Neste capítulo é detalhada a metodologia empregada para avaliar experimentalmente o desempenho do sistema proposto neste trabalho. O sistema foi avaliado com imagens reais de semáforos, colhidas por câmera instalada em um veículo autônomo correntemente em desenvolvimento na UFES, denominado IARA (*Intelligent Autonomous Robotic Automobile*).

O hardware da IARA é descrito na Seção 4.1. O conjunto de módulos de software que compõem o sistema de controle da IARA é apresentado na Seção 4.2. Na Seção 4.3 é apresentado o ambiente de desenvolvimento CARMEN, empregado no desenvolvimento dos módulos de software da IARA e também no desenvolvimento do nosso sistema. A arquitetura do nosso sistema é detalhada na Seção 4.4. Na Seção 4.5 é descrito o sistema desenvolvido para o mapeamento semiautomático de semáforos, que foi útil para a visualização dos dados usados na avaliação experimental de nosso sistema. Para avalia-lo, foi construída uma base de dados de semáforos, que é apresentada na Seção 4.6. Por fim, as métricas utilizadas na avaliação são detalhadas na Seção 4.7.

4.1 A Plataforma Robótica IARA

A IARA é uma plataforma robótica experimental baseada em um automóvel de passeio adaptado. Esta adaptação envolveu a instalação no automóvel de: (i) mecanismos para controlar o acelerador, freio, posição do volante, etc.; (ii) sensores; (iii) computadores para receber os dados dos sensores e controlar o automóvel; e (iv) fontes de energia para os computadores e sensores.

Após investigar inúmeras empresas no país e no exterior, a equipe do Laboratório de Computação de Alto Desempenho (LCAD) da UFES que desenvolveu a IARA não encontrou nenhuma empresa que oferecesse tecnologia similar ou superior à oferecida pela empresa norte americana Torc Robotics (http://www.torcrobotics.com) para o acionamento dos atuadores (volante, acelerador, freio, entre outros) do automóvel e disponibilização de energia elétrica para alimentação dos computadores

e sensores necessários para os estudos pretendidos com a IARA [16]. À época da aquisição, a tecnologia da Torc para veículos de passeio somente operava com o automóvel Ford Escape Hybrid. Assim, o LCAD importou este automóvel (Figura 12(a)) dos USA já com as tecnologias de acionamento e disponibilização de energia instaladas pela Torc (Figura 12(b)).

(a) (b)

Figura 12: (a) *Ford Escape Hybrid*; (b) Equipamento de disponibilização de energia desenvolvido pela empresa *Torc Robotics*.

Como sensores, a equipe do LCAD escolheu para a IARA as câmeras de vídeo estéreo Bumblebee XB3 da Point Grey (http://www.ptgrey.com) (Figura 13(a)), o *Light Detection And Ranging* (LIDAR) HDL-32E da Velodyne (http://www.velodynelidar.com) (Figura 13(b)) e o *GPS-aided Attitude and Heading Reference System* (AHRS/GPS) MTi-G da Xsens (http://www.xsens.com), que inclui um GPS e uma *Inertial Measurement Unit* (IMU) (Figura 13(c)) [16].

Para controlar a plataforma robótica, a equipe do LCAD optou por um mini supercomputador (mini cluster) composto por quatro computadores Dell Precision R5500 (Figura 13(d)) – alto desempenho computacional é importante devido à alta demanda computacional tipicamente observada em algoritmos que envolvem o processamento de imagens (um dos principais tópicos de estudo da equipe do LCAD é a cognição visual artificial [16]). Cada máquina do minicluster possui 2 Processadores Intel Xeon 2.13 GHZ, 12 GB de memória DDR3 1333MHZ, 2 HDs SSD 120GB em RAID0, 2 Placas de Rede 1GB, Placa de Vídeo Quadro 600, Placa de Vídeo Tesla C2050 [16].

(a) (b) (c)

(d)

Figura 13: (a) Câmera de vídeo estéreo *Bumblebee* XB3 da *Point Grey*; (b) LIDAR HDL-32E da *Velodyne*; (c) AHRS/GPS MTi-G da *Xsens*; (d) Computador *Dell Precision* R5500.

A Figura 14 mostra a plataforma robótica experimental do LCAD, que possui capacidades equivalentes às mais modernas hoje em estudo no mundo [16].

Figura 14: Plataforma robótica experimental IARA, composta de um automóvel de passeio *Ford Escape Hybrid*, sensores, mini-supercomputador do tipo cluster (apenas 2 dos quatro nós instalados no momento da foto) e fontes de alimentação.

4.2 Sistema de Controle da IARA

A equipe do LCAD desenvolveu todo o sistema de controle para a IARA operar autonomamente [16]. Este sistema é composto por diversos módulos que foram implementados usando técnicas probabilísticas do estado da arte de localização [17], mapeamento [17,18] e navegação [19,20,21,22,23] de veículos autônomos (Figura 15).

Foram instalados na IARA os sensores (em amarelo na Figura 15): GPS e IMU, parte da AHRS/GPS da Xsens; câmeras estéreo frontal e laterais Bumblebee; LIDAR Velodyne (Figura 14); e sensores de velocidade do carro e angulo do volante, v e phi (estes últimos dois sensores fazem parte da solução adquirida da Torc). Os dados destes sensores são pré-processados por *drivers* (em vermelho na Figura 15 – todos desenvolvidos pelo grupo de pesquisa do LCAD) e enviados para filtros (em verde na Figura 15 – todos desenvolvidos pelo grupo de pesquisa do LCAD), que recebem como entrada dados de uma ou mais fontes e geram como saída versões processadas destes dados.

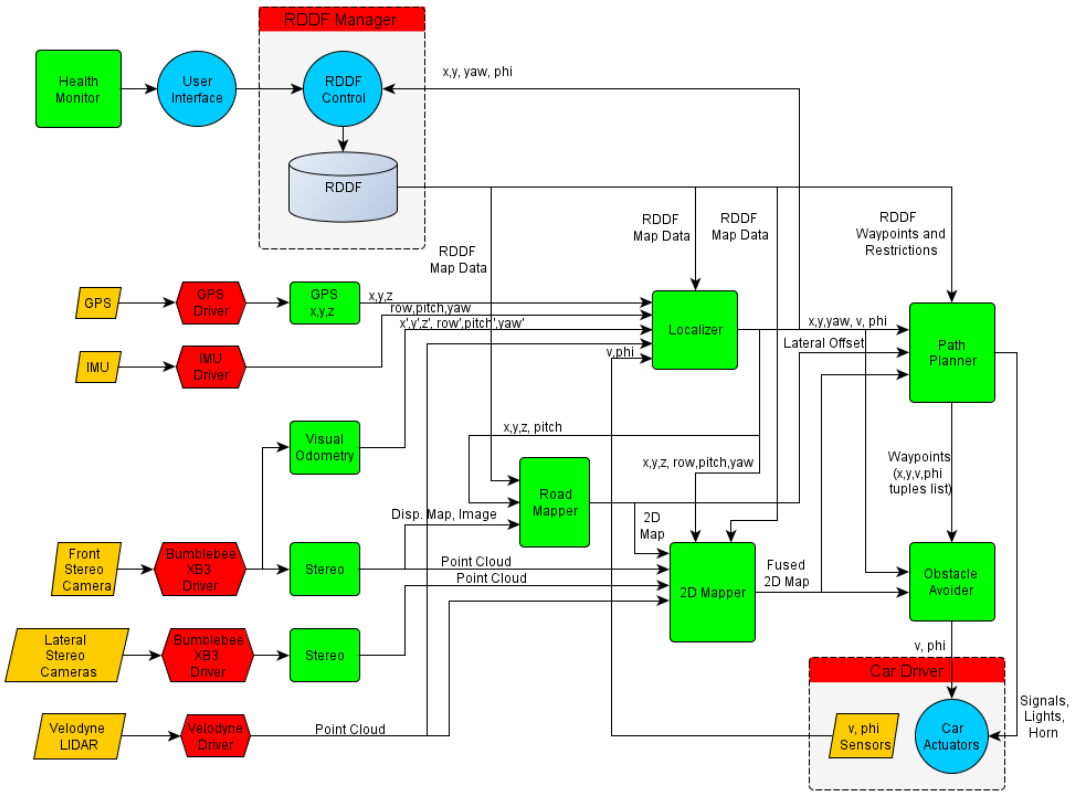


Figura 15: Sistema de Controle da IARA.

A equipe do LCAD desenvolveu os filtros: *GPS x,y,z*, que transforma dados de latitude e longitude de GPS em coordenadas UTM (http://en.wikipedia.org/wiki/Universal_Transverse_Mercator_coordinate_system); V*isual Odometry*, que, a partir de imagens estéreo, produz dados de odometria para a IARA (posição em 6D ao longo do tempo); *Stereo* [24] [25], que, a partir de imagens estéreo, produz mapas de profundidade 3D; *Road Mapper* [26], que, a partir de mapas de profundidade e dados de odometria, produz um mapa de obstáculos e áreas livres à frente da IARA; *Localizer*, que localiza a IARA em 6D no mundo; *2D Mapper*, que cria um mapa on-line da pista e de obstáculos ao redor da IARA; *Path Planner* [27], que gera os sinais de navegação da IARA; *Obstacle Avoider*, que filtra os dados de controle de baixo nível em alta velocidade de modo a evitar colisões eminentes; e *Health Monitor*, que verifica continuamente o bom estado de funcionamento de todos os módulos, reinicia automaticamente módulos que não estejam operando corretamente e informa ao usuário sobre suas ações.

User Interface da IARA permite especificar tarefas para a mesma executar. Para auxiliar o usuário na especificação de tarefas, o Sistema de Controle da IARA dispõe de um *Road Definition Data File Manager* (*RDDF Manager*). Sob o controle da *User Interface*, o *RDDF Manager* armazena e permite a manipulação de dados coletados pela IARA quando em modo de aprendizado ou em modo autônomo. No modo de aprendizado, a IARA é conduzida por um motorista, armazena todos os dados dos sensores e constrói mapas dos ambientes trafegados – estes mapas e dados de sensores são processados e armazenados na forma de um *Road Definition Data File* e de diversos mapas de interesse. Em modo autônomo, o usuário pode controlar a IARA de dentro dela ou de fora dela via Internet. Quando dentro da IARA, o usuário pode especificar um destino em um mapa por meio de um mouse; e quando fora da IARA, o usuário pode especificar um destino por meio de toques na *User Interface* via *tablet* ou telefone celular. A *User Interface* permite ainda visualizar e controlar diversos outros aspectos de operação da IARA.

Além da *User Interface*, outros módulos desenvolvidos permitem a visualização de informações da IARA. Como exemplos desses módulos podemos citar o *viewer_3D* e o *bumblebee_viewer.* O *viewer_3D* é um módulo que recebe dados de sensores e de módulos de controle da IARA e permite a visualização destes dados em três

dimensões em uma tela. Um dos sensores do qual o *viewer_3D* recebe informações é o LIDAR HDL-32E. Este sensor faz uma varredura circular do ambiente a 16 *hertz* usando 32 lasers que medem distâncias individuais, gerando uma nuvem de pontos da região ao redor da IARA. A Figura 16 mostra a interface do usuário do *viewer_3D* com uma nuvem de pontos colhida pelo LIDAR HDL-32E da IARA. Nessa imagem, podem-se verificar obstáculos ao redor do carro além da própria representação do carro no centro da nuvem de pontos.

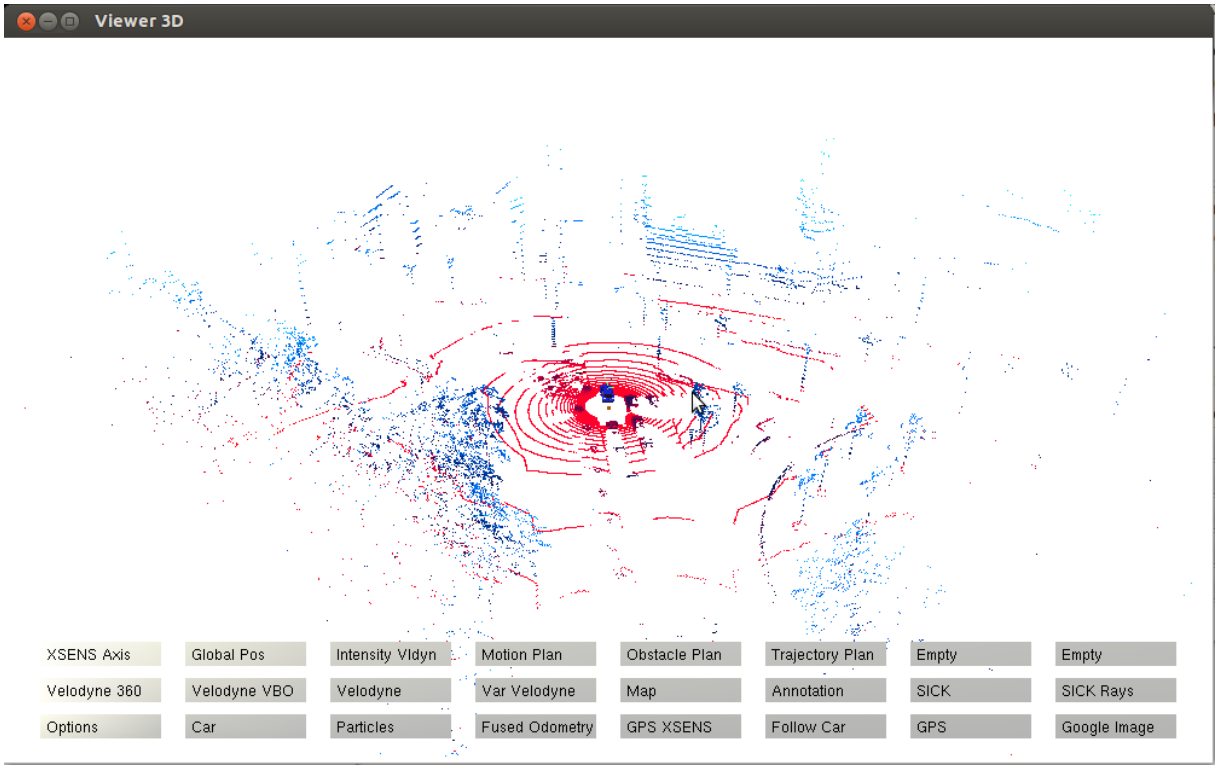


Figura 16 – *viewer_3D* exibindo uma nuvem de pontos colhida pelo LIDAR HDL-32E da IARA.

O *bumblebee_viewer* é um módulo que recebe imagens estéreo da câmera Bumblebee XB3 da Point Grey e exibe-as lado a lado em sua interface com o usuário. A Figura 17 apresenta a interface do usuário do *bumblebee_viewer* com um par de imagens estéreo colhido por uma das câmeras Bumblebee XB3 da IARA.

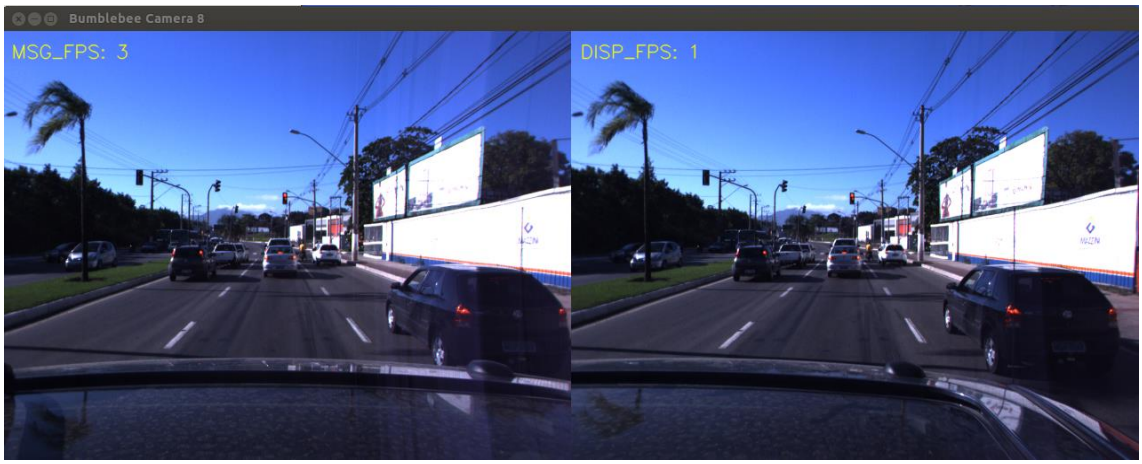


Figura 17 - *bumblebee_viewer* exibindo um par de imagens estéreo colhido por uma das câmeras Bumblebee XB3 da IARA.

4.3 Ambiente de Desenvolvimento CARMEN

Para a implementação dos diversos módulos da IARA mencionados acima, a equipe do LCAD empregou o *framework* CARMEN, desenvolvido pelo grupo de *Sebastian Thrun* em *Carnegie Mellon* e posteriormente em *Stanford* (http://carmen.sourceforge.net). O desenvolvimento da versão *open source* de CARMEN foi descontinuado em 2008; contudo, este *framework* garantiu a seus desenvolvedores a vitória na *Defense Advanced Research Projects Agency* (DARPA) *Grand Challege* de 2005 (http://archive.darpa.mil/grandchallenge05) e o segundo lugar (muito embora tenha completado o percurso em primeiro lugar) na DARPA *Urban Challenge* (http://archive.darpa.mil/grandchallenge/index.asp). Assim, embora existam outros *frameworks open source* hoje disponíveis e mantidos, a equipe do LCAD optou por continuar o desenvolvimento de CARMEN no Laboratório de Computação de Alto Desempenho (LCAD) da UFES (http://www.lcad.inf.ufes.br/wiki/index.php/Carmen_Robot_Navigation_Toolkit) [16]. Correntemente, a equipe do LCAD está negociando com a *Carnegie Mellon* a continuidade do desenvolvimento da versão *open source* de CARMEN pelo seu grupo de pesquisa.

CARMEN viabiliza o desenvolvimento de sistemas compostos por múltiplos programas executáveis (ou módulos) que se comunicam segundo o paradigma *publish-subscribe*. De acordo com este paradigma, um módulo sensor, por exemplo,

pode ser implementado por meio de um programa executável independente que envia (*publish*) mensagens com os dados obtidos do sensor para quaisquer módulos que assinem (*subscribe to*) estas mensagens. Um módulo filtro pode assinar mensagens de vários módulos, manipular (“filtrar”) estas mensagens com algoritmos de interesse e publicar mensagens com seus resultados para vários outros módulos que as requeiram. Um módulo atuador pode receber mensagens de vários outros módulos, executar algoritmos sobre elas e usar o resultado para controlar algo no mundo real.

Um módulo que publica uma mensagem não precisa saber quem as recebe; assim, evitam-se problemas como *dead lock* e *starvation* que dificultam a programação de sistemas distribuídos (sistemas de controle de veículos autônomos são inerentemente distribuídos). Os módulos da IARA rodam no seu mini-supercomputador, com nós empregando o sistema operacional Linux Ubuntu 12.4 e *kernel* de tempo real (https://rt.wiki.kernel.org). O uso de um sistema operacional de tempo real facilita a implementação de algoritmos que lidam com o tempo, como é o caso da maioria dos algoritmos implementados pelos módulos da IARA.

Os diversos módulos da IARA foram implementados e testados no seu arcabouço de simulação de veículos autônomos (não apresentado aqui) e na própria IARA [16]. Vídeos que demonstram diversos resultados destes testes podem ser examinados no canal do LCAD no YouTube: http://www.youtube.com/user/lcadufes. Os resultados alcançados com a IARA foram divulgados pela imprensa escrita e televisiva. As várias reportagens podem ser lidas/assistidas em http://www.lcad.inf.ufes.br/index.php?option=com_content&task=blogcategory&id=23&Itemid=66.

4.4 Arquitetura do Sistema

O sistema desenvolvido dentro do contexto deste trabalho foi implementado como um novo módulo CARMEN do sistema de controle da IARA, denominado *Traffic Light Detector and State Recognizer* – ou TLDSR (Figura 18, destaque). Este módulo recebe mensagens da *Front Stereo Camera* da IARA, e dos módulos

Localizer e *RDDF Manager*, e gera como saída mensagens para o módulo *Path Planner.*

As mensagens recebidas da *Front Stereo Camera* são pares de imagens estéreo publicadas pelo driver da câmera Bumblebee XB3, *Bumblebee XB3 Driver*. As mensagens recebidas do módulo *Localizer* correspondem a poses no formato *x*, *y* e *yaw* da IARA, onde *x* e *y* são as coordenadas do plano do chão no sistema de coordenadas UTM, e *yaw* é sentido para o qual a IARA está apontando, medido em radianos conforme o sistema de coordenadas de CARMEN (o Norte verdadeiro da Terra é o ângulo zero e o ângulo cresce no sentido anti-horário). As mensagens recebidas do RDDF Manager são poses de semáforos mapeados previamente e armazenadas no *Road Definition Data File* gerenciado pelo RDDF Manager.

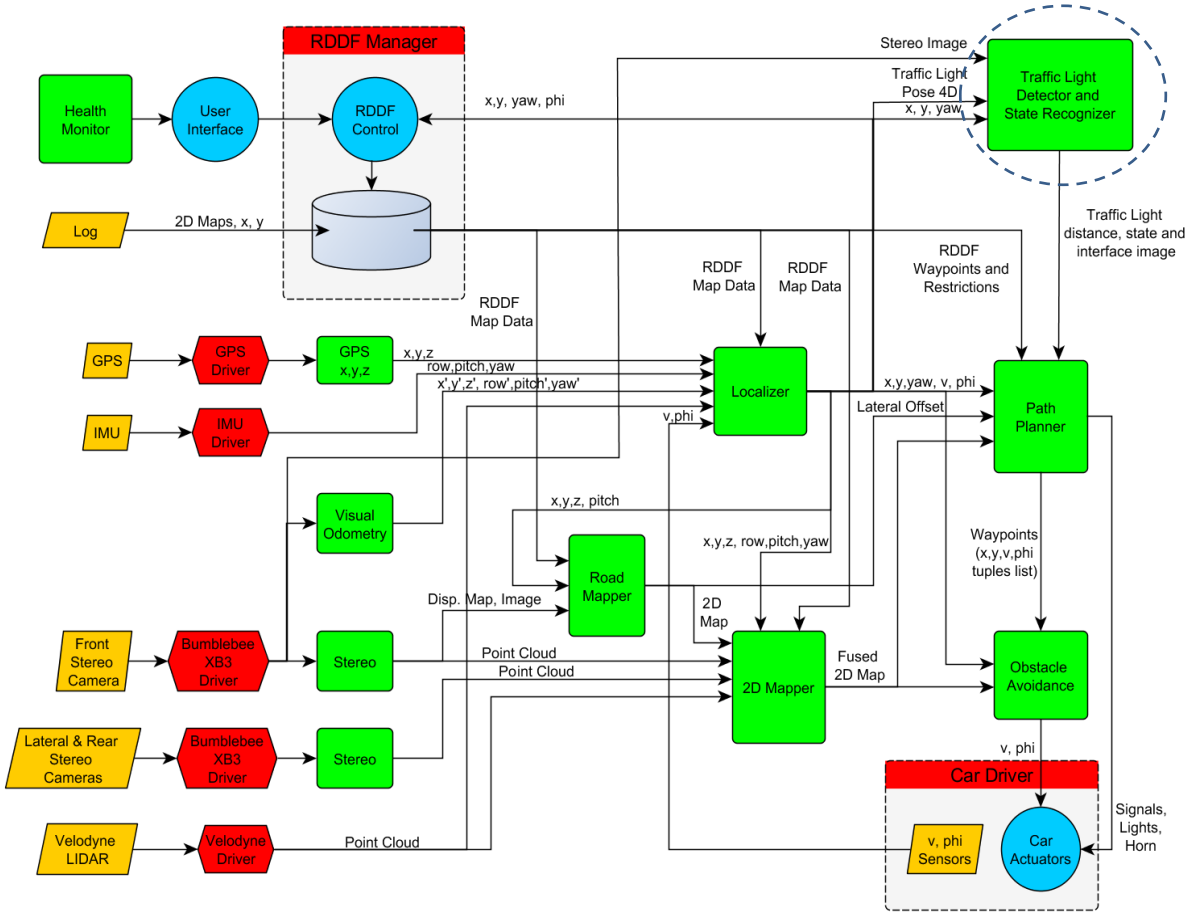


Figura 18 – Sistema de controle da IARA com a inclusão do TLDSR (em destaque na figura).

A mensagem publicada pelo TLDSR contém a distância e o estado (fechado ou aberto – o sinal amarelo é correntemente tratado como aberto) do semáforo a frente da IARA, além de uma imagem equivalente à recebida da *Front Stereo Camera*, mas processada de modo a conter: (i) um retângulo em torno de semáforos detectados; (ii) o estado do semáforo mais próximo, indicado como um círculo na cor verde (aberto) ou vermelha (fechado) no canto superior esquerdo; e (iii) a distância da IARA até este semáforo mais próximo, indicada na forma de texto na parte inferior da imagem. Foi desenvolvida uma interface de visualização desta imagem que permite apreciar o funcionamento do TLDSR.

O TLDSR busca detectar e reconhecer o estado de semáforos presentes nas mensagens recebidas da *Front Stereo Camera*. Quando são detectados um ou mais semáforos em uma imagem, o TLDSR busca na lista de semáforos mapeados previamente (recebida do *RDDF Manager*) aquele que é o mais próximo da atual pose da IARA (recebida do módulo *Localizer*). A orientação da IARA e dos semáforos mapeados previamente são consideradas de modo a evitar a associação incorreta do semáforo à frente com outros semáforos eventualmente armazenados no RDDF.

O TLDSR é composto pelos seguintes blocos (Figura 19): Recortador, Detector, Reescalador, Pre-Processador, Reconhecedor, Processador da Imagem, Medidor de Distância e o Publicador. O Recortador recebe imagens da *Front Stereo Camera* e realiza um recorte na imagem para remover partes da mesma onde nunca aparecem semáforos em condições relevantes, enviando em seguida essa imagem recortada para o Detector. O Detector recebe a imagem recortada e, empregando o algoritmo Haar Cascade [12], gera uma lista com zero ou mais *Regions of Interest* (ROIs) de possíveis semáforos detectados; esta lista é enviada para o Reescalador. O Reescalador recebe imagens da *Front Stereo Camera* e a lista de ROIs, e realiza o reescalamento das ROIs dos semáforos para o tamanho de 10x20 *pixels*; estas imagens reescaladas das ROIs são enviadas para o Pre-Processador. O Pre-Processador realiza diversas transformações nestas imagens reescaladas e envia as imagens pre-processadas para o Reconhecedor. Paralelamente, o Medidor de Distância recebe a localização da IARA e as poses anotadas dos semáforos do *RDDF Manager*, e calcula a distância da IARA até o semáforo mais próximo. O Reconhecedor do estado de semáforos recebe as imagens reescaladas e pré-

processadas e, empregando o reconhecedor de estados baseado em SVM [13], retorna o estado do semáforo em cada imagem recebida para o Processador de Imagens. O Processador de Imagem recebe a lista de ROIs dos semáforos, os estados dos mesmos, a distância até o semáforo mais próximo e a imagem da *Front Stereo Camera*, e gera uma imagem contendo todas essas informações para o Publicador. O Publicador, então, publica estas imagens contendo todas essas informações, juntamente com estas informações em formato que pode ser usado por qualquer módulo da IARA, em especial o módulo *Path Planner*.

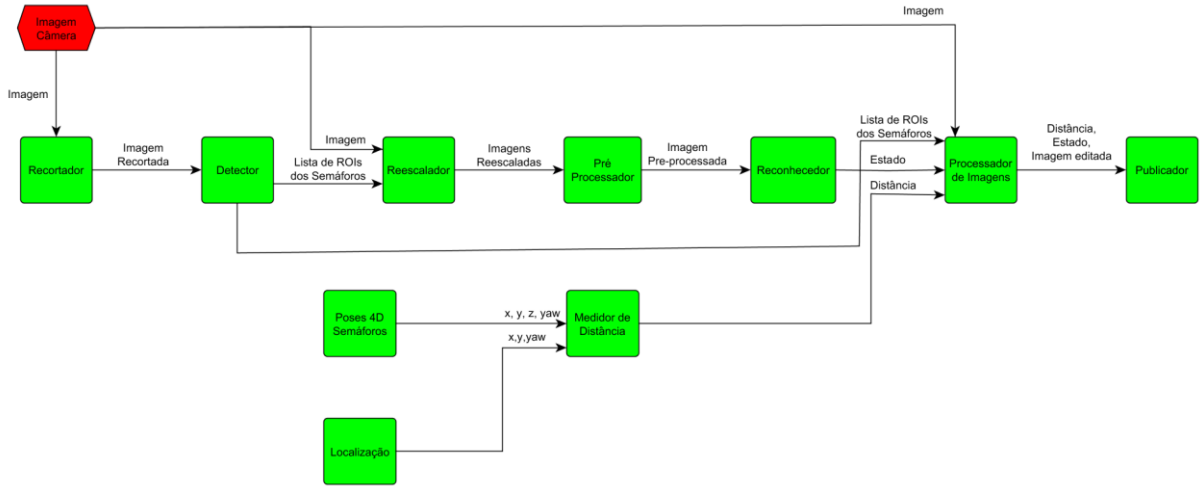


Figura 19 - Diagrama de blocos do sistema TLDSR.

4.5 Mapeamento Semiautomático de Semáforos

Como parte deste trabalho, nós desenvolvemos um método para realizar o mapeamento semiautomático de semáforos, isto é, a inclusão de sua pose no arquivo RDDF manipulado pelo *RDDF Manager* da IARA. Para implementar este método nós alteramos os módulos *viewer_3D* e *bumblebee_viewer* da IARA, e criamos um novo módulo denominado *rddf_annotation_manager*.

Segundo este método, para mapear um semáforo é necessário ter informações sobre sua pose e orientação de modo a tornar possível armazenar esta informação como parte do RDDF. Para obter essas informações é necessário um *log* da região

em que se deseja mapear os semáforos, ou seja, o mapeamento é feito *off-line* de forma semiautomática por um operador.

Com o arquivo de *log* sendo “executado” pelo módulo *playback* da IARA, o operador pode verificar visualmente na interface do *bumblebee_viewer* se existe ou não um semáforo na imagem. Caso haja um semáforo, o operador pode verificar no *viewer*_3D se existe algum ponto na nuvem de pontos que corresponda ao semáforo. O método criado possibilita o clique na região do semáforo na nuvem de pontos para a obtenção e armazenamento de informações, isto é, o *viewer_3D* foi modificado para possibilitar as anotações dos semáforos.

Ao clicar no botão da interface do *viewer_3D* denominado “*Annotation*”, o *viewer_3D* entra em modo de anotação. Neste modo, os pontos da nuvem de pontos são “engordados” de forma a facilitar o clique em um ponto específico, como pode ser visto na Figura 20. Para este modo de funcionamento do *viewer_3D* foi implementada uma função de *picking* que possibilita que, ao se clicar em um ponto (x, y) da tela, este ponto seja transformado em um ponto 3D de um sistema de coordenadas centrado na IARA. Esse ponto 3D é somado à posição do carro em coordenadas UTM do mundo. O resultado corresponde à posição 3D (x, y, z), em coordenadas UTM do mundo, do ponto da nuvem que foi clicado. A orientação da IARA é adicionada a este ponto formando uma pose 4D (x, y, z, yaw) do ponto no mundo.

No modo anotação, este ponto pode ser publicado pelo *viewer_3D* como sendo uma dentre diversas possibilidades de anotações de RDDF que incluem, além de semáforos, placas de trânsitos, faixas de pedestres, locais de parada, catracas, quebra molas e limites de velocidade (ver botões do *viewer_3D* mostrados na Figura 20).

O módulo *rddf_annotation_manager* é um módulo de controle de anotações. Este módulo recebe as mensagens de anotação publicadas pelo *viewer_3D* e gerencia as anotações, armazenando-as no RDDF e publicando-as para quem precisar. Um exemplo do formato de anotações salvo pode ser conferido na Figura 21. O delimitador usado foi o caractere de tabulação. Os três primeiros campos são campos de tipo de mensagem. O quarto campo corresponde à orientação do carro

em radianos. O quinto e sexto campos são, respectivamente, as coordenadas x e y da IARA no formato UTM. O último campo corresponde à altura do ponto anotado.

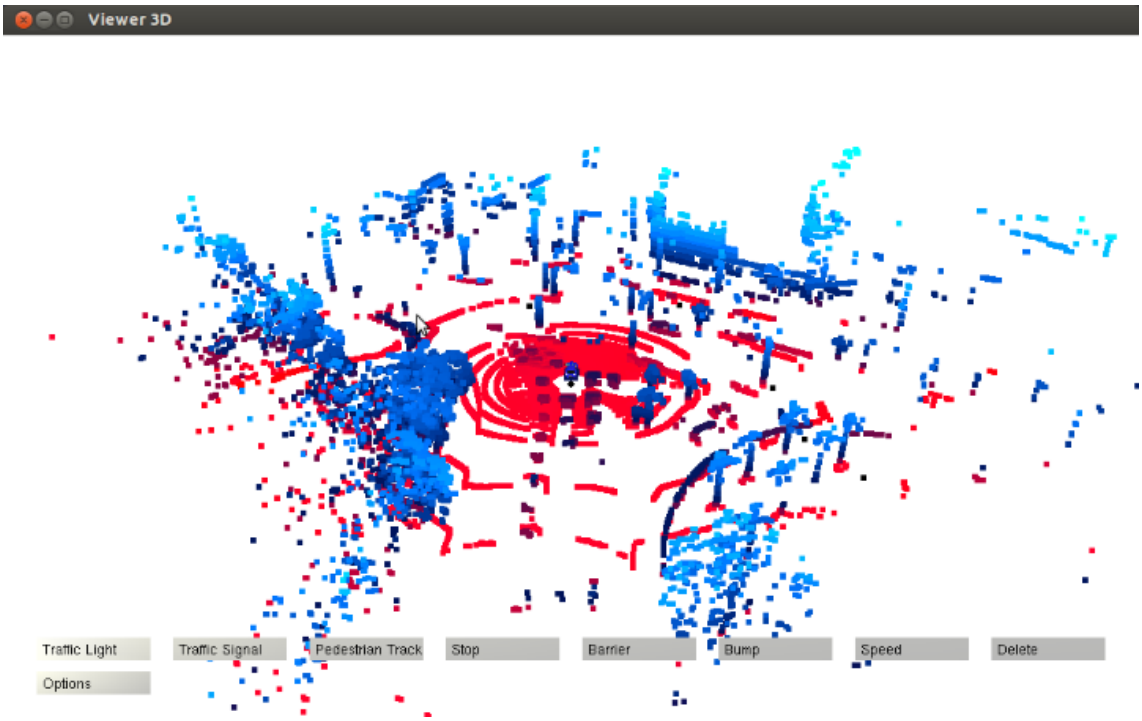


Figura 20 - *viewer_3D* com os pontos "engordados".

```
RDDF_ANNOTATION_TYPE_TRAFFIC_LIGHT      1       0       -1.016236       7757557.472621  -364161.040379  4.784661
RDDF_ANNOTATION_TYPE_TRAFFIC_LIGHT      1       0       -1.016236       7757551.086695  -364161.984872  4.371912
```

Figura 21 - Exemplo de anotações salvas pelo *rddf_annotation_manager.*

O *bumblebee_viewer* foi modificado de forma a projetar na sua interface anotações de semáforos. Para isso, o *bumblebee_viewer* recebe as mensagens das anotações do *rddf_annotation_manager*, a pose do modulo *Localizer* e as mensagens da *Front Stereo Camera*. As poses são armazenas de forma que, sempre que chegar uma nova imagem, é procurada uma pose que foi publicada em um tempo mais próximo do da câmera. Com esses dados é usada pela Equação 4, com:

- coordenadas homogêneas da imagem $(u \quad v \quad 1)^T$;
- distância focal f;

- ponto principal da câmera (c_u, c_u);
- matriz de rotação $R(r) = R_x(r_x)R_y(r_y)R_z(r_z)$;
- vetor de translação $t = (t_x\ t_y\ t_z)^T$;
- coordenadas do ponto 3D $X = (x\ y\ z)^T$.

$$\begin{pmatrix} u \\ v \\ 1 \end{pmatrix} = \begin{pmatrix} f & 0 & c_u \\ 0 & f & c_v \\ 0 & 0 & 1 \end{pmatrix} \left[(R(r)\ \ t) \begin{pmatrix} x \\ y \\ z \\ 1 \end{pmatrix} \right] \qquad \textbf{Equação 4}$$

Essa equação retorna os ponto u e v que correspondem aos pontos (x, y) na imagem da câmera. A Figura 22 demonstra a projeção feita de dois semáforos. O contorno dos semáforos anotados é mostrado em verde.

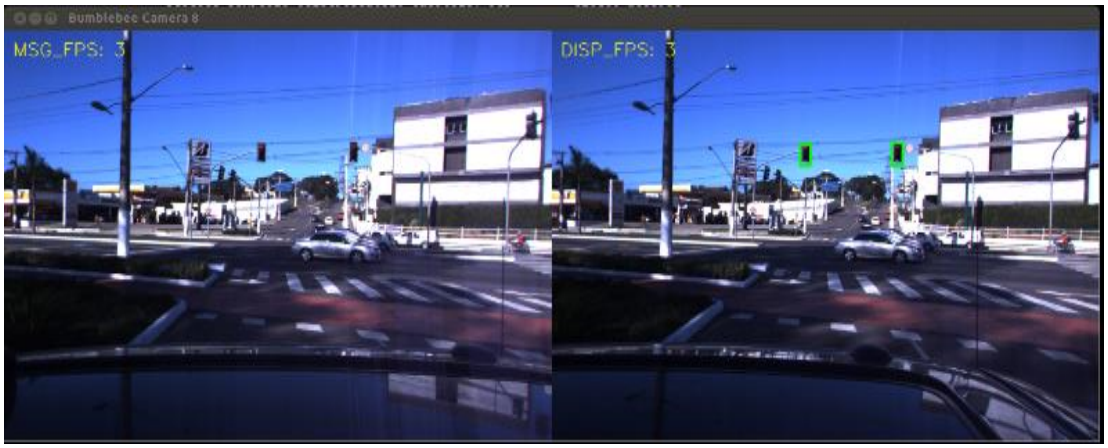


Figura 22 - *bumblebee_viewer* com a projeção dos semáforos na imagem direita.

4.6 Base de Dados de Semáforos

Foi criada uma base de dados de imagens de semáforos para o desenvolvimento do sistema TLDSR. Esta base de dados contém três conjuntos de imagens: (i) um conjunto para o treino do detector e do reconhecedor do estado de semáforos do TLDSR; (ii) um conjunto para a validação do desempenho de detecção e do desempenho de reconhecimento do detector e do reconhecedor do estado de semáforos do TLDSR, respectivamente; e (iii) um conjunto para o teste do

desempenho final destes dois subsistemas do TLDSR. Cada imagem de cada um destes três conjuntos possui um *ground truth*, isto é, uma anotação dos locais na imagem onde se encontram semáforos.

Esta base de dados foi criada a partir de um cenário real experimentado pela plataforma robótica IARA. Neste cenário, a IARA foi conduzida por um percurso de cerca de 6370 metros. As informações capturadas pelos sensores durante o percurso foram gravadas em um arquivo de *log* (este arquivo de *log* está disponível no endereço http://www.lcad.inf.ufes.br/log/traffic_light/log/). A Figura 23 mostra a visão aérea da região percorrida pela IARA com o caminho destacado em vermelho.

Figura 23- Visão aérea da região percorrida pela IARA com o caminho destacado em vermelho.

Para a criação da base de dados de imagens usada nesse trabalho a partir do *log* mencionado acima, foi criado um novo módulo CARMEN denominado *log_generate_images*. O *log_generate_images* recebe mensagens da *Front Stereo Camera* e mensagens do módulo *Localizer*, e salva a imagem direita de cada par estéreo em um diretório especificado pelo usuário, onde o nome de cada arquivo contém as seguintes informações, separadas pelo caractere "_":

- Texto “image”;
- Índice da imagem com 4 dígitos;
- *TimeStamp* da mensagem da *Front Stereo Camera;*
- x, y em coordenadas UTM;

O *log_generate_images* possui um contador interno que é usado para computar o valor do índice da imagem, que começa em 1 e cresce sequencialmente. Um exemplo de nome de arquivo salvo:

image_0035_1379356097.403792_7757541.036719_-364135.774017.png

As imagens salvas possuem o tamanho de 1280x960 *pixels* e são codificadas em RGB. O número total de imagens capturadas para os experimentos apresentados neste trabalho foi 2056 imagens. Alguns exemplos das imagens capturadas podem ser examinados na Figura 24.

Figura 24 - Exemplos de imagens salvas pelo módulo *log_generate_images*.

Para criar a *ground truth* de cada imagem foi desenvolvido um programa UNIX separado do CARMEN denominado *generate_gt*, que possibilita a marcação de regiões de interesse (*Region of Interest* – ROI) em imagens e a sua visualização. Esse programa recebe como entrada dois parâmetros, sendo o primeiro o nome de um arquivo texto que contém uma lista de nomes de arquivos de imagens a serem marcadas, e o segundo o nome do arquivo de saída que conterá a *ground truth* de cada imagem. Este programa permite que, com eventos de clique do mouse, ROIs sejam selecionadas em cada imagem. Com uma região selecionada em uma

imagem, ao clicar a tecla espaço, o programa captura a ROI. Caso o usuário queira salva-la, bastar apertar a tecla “s”, permitindo que outra região seja selecionada. Ao pressionar a tecla “n”, a próxima imagem da lista é buscada para que suas ROIs sejam marcadas e salvas. A interface com usuário do programa *generate_gt* é apresentada na Figura 25. O retângulo em rosa denota uma ROI de um semáforo marcada pelo usuário.

As ROIs são salvas no arquivo especificado pelo usuário. Este arquivo armazena linhas com as seguintes informações, separados por espaço (“ ”):

- Nome da Imagem
- ROI da imagem
    - X superior
    - Y superior
    - X inferior
    - Y inferior

Um exemplo de linha salva no arquivo especificado pelo usuário:

image_0035_1379356097.403792_7757541.036719_-364135.774017.png 650 320 680 350

Se uma imagem tem mais de uma ROI, a mesma imagem aparecerá em mais de uma linha de saída do *generate_gt*.

Exemplo de linha de comando usada para invocar o *generate_gt*:

./generate_gt <arquivo com a lista das imagens> <arquivo de saída>

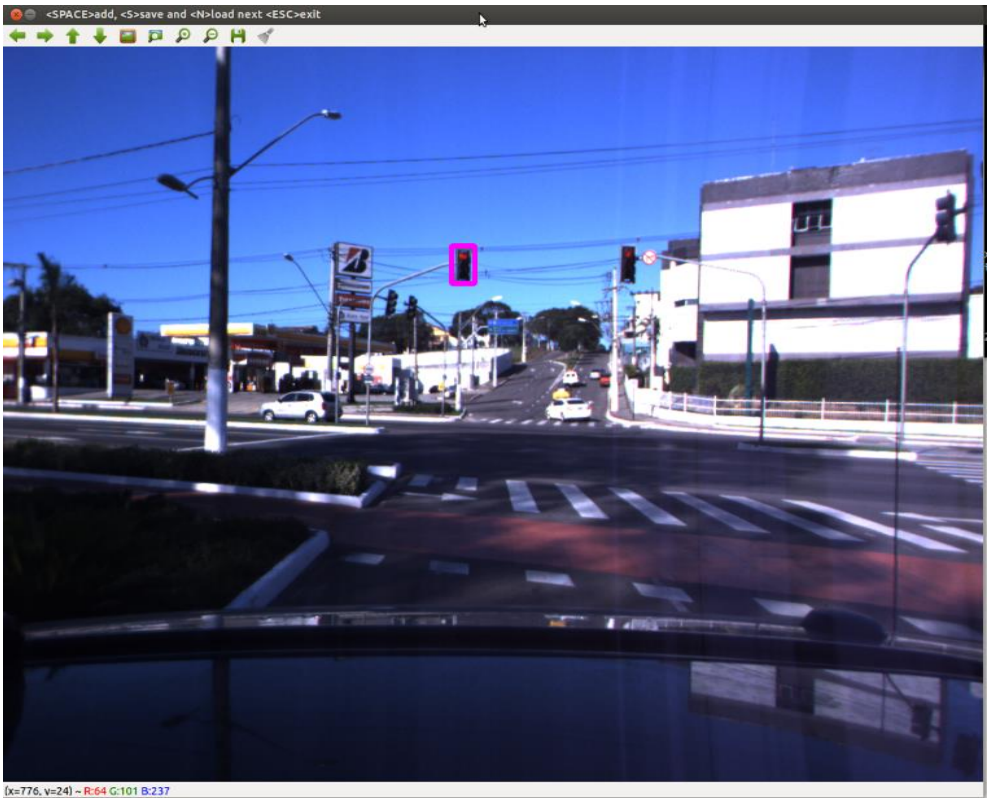


Figura 25 - Interface de marcação dos semáforos do *generate_gt*.

Para dividir as 2056 imagens capturadas nos três subconjuntos de imagens usados nos experimentos (treino, validação e teste), nós criamos dois programas UNIX separados do CARMEN. O primeiro, denominado *create_division*, possibilita a divisão em partes iguais do conjunto de imagens de entrada nos três subconjuntos de imagens. Na verdade, aproximadamente em partes iguais se o total de imagens de entrada não for divisível por três. As imagens de entrada são distribuídas entre os três subconjuntos de forma aleatória.

O programa *create_division* recebe como entrada um arquivo texto com a lista de nomes das imagens de entrada e gera três arquivos de saída, train.txt, test.txt e validation.txt, que recebem as listas de nomes de imagens dos respectivos subconjuntos usados nos experimentos. Exemplo de linha de comando usada para invocar o *create_division*:

./create_division <arquivo com a lista das imagens capturadas>

O segundo programa usado para dividir as 2056 imagens capturadas nos três subconjuntos de imagens usados nos experimentos, denomidado *create_gt_filtered*, cria um arquivo de *ground truth* para cada subconjunto de imagens usando os arquivos de saída do *create_division*. Esse programa recebe como entrada três parâmetros, sendo o primeiro o nome de um dos arquivos de subconjunto de imagens gerados pelo *create_division*, o segundo é o *ground truth* de toda a base e o terceiro é o nome do arquivo de saída, que conterá o *ground truth* filtrado para o subconjunto. Exemplo de linha de comando usada para invocar o *create_gt_filtered*:

./create_gt_filtered <nome do arquivo do subconjunto de imagens> <arquivo de *ground truth*> <arquivo de *ground truth* filtrado>

A quantidade de imagens de cada subconjunto, o total de semáforos nestas imagens, e o número de semáforos em cada estado (vermelho, amarelo ou verde) nestas imagens são apresentados na Tabela 1. Como pode ser observado na tabela, em toda a base de dados apenas um semáforo amarelo foi encontrado.

Tabela 1 – Quantidade de imagens e semáforos para cada subconjunto da base de dados.

	Imagens	Semáforos	Vermelho	Amarelo	Verde
Treino	686	522	207	0	315
Validação	685	519	204	0	315
Teste	686	564	205	1	358

As imagens capturadas, seu *ground truth* e a lista das imagens de cada subconjunto com os seus respectivos *ground truths* filtrados estão disponibilizados no endereço http://www.lcad.inf.ufes.br/log/traffic_light/database.

4.7 Métricas

A saída típica de algoritmos de detecção de elementos de interesse em imagens consiste de uma lista de regiões de interesse (ROIs) retangulares. Cada ROI S detectado pode ser avaliado por meio de sua comparação com um *ground truth* G. A

métrica utilizada nesta comparação pode variar. Neste trabalho nós empregamos o coeficiente de similaridade *Jaccard*, frequentemente usado na literatura [28] [29] [30], e definido na Equação 5.

$$J(S,G) = \frac{|S \cap G|}{|S \cup G|} \in [0,1] \qquad \textbf{Equação 5}$$

Na equação, o termo $|S \cap G|$ denota o número de pixels dos retângulos S *e* G que coincidem, ou seja, é a interseção dos dois retângulos; enquanto que $|S \cup G|$ é a soma dos pixels da imagem formada pela superposição de S e G, ou seja, é a união dos dois retângulos. A medida $J(S,G)$ indica, então, o percentual de superposição entre S e G. Nós definimos como um percentual mínimo de superposição, ou ponto de corte $J(S,G) = 60\%$. Assim, em nossos experimentos, dada uma imagem de teste, se um ROI S retornado pelo nosso detector de semáforos possuir 60% ou mais de superposição com um *ground truth* G da mesma imagem de teste, este S é considerado como acerto (*hit*) do detector, ou seja, um semáforo; caso contrário, é considerado como um erro (*miss*) de detecção.

O coeficiente de similaridade *Jaccard* foi usado neste trabalho (e com este ponto de corte específico) porque ele permite medir o quão boa foi uma detecção no contexto de interesse, que é detectar regiões na imagem de entrada que possam, posteriormente, ser repassadas diretamente para nosso reconhecedor do estado de semáforos.

O desempenho do detector de semáforos será tanto maior, quanto mais frequentemente sua saída coincidir com a *ground truth*. No processo de verificação de coincidência quatro situações podem ocorrer:

- Verdadeiro Positivo (*True Positive* – TP): Existe um semáforo e o detector retorna um ROI com $J(S,G) \geq 60\%$.
- Verdadeiro Negativo (*True Negative* – TN): Não existe um semáforo e o detector não retorna nenhum ROI.
- Falso Positivo (*False Positive* – FP): Não existe um semáforo, mais o detector retorna um ROI.

- Falso Negativo (*False Negative* – FN): Existe um semáforo, mais o detector não retorna um ROI com $J(S,G) \geq 60\%$.

A partir das situações acima é possível computar duas métricas padrões usadas para medir o desempenho de classificadores binários, como é o caso de nosso detector de semáforos:

- Precisão (*Precision*): É a taxa de semáforos detectados que realmente são semáforos. É dada pela divisão do total de semáforos detectados (TP) pelo número total de ROIs retornados (FP + TP), sendo definida pela Equação 6.

$$Precisão = \frac{TP}{TP + FP} \qquad \textbf{Equação 6}$$

- Revocação (*Recall)*: É a capacidade do sistema de achar semáforos. É dada pela divisão do total de semáforos detectados (TP) pelo número total de semáforos que deveria ser detectado (TP + FN), sendo definida pela Equação 7.

$$Revocação = \frac{TP}{TP + FN} \qquad \textbf{Equação 7}$$

Um sistema de detecção de semáforos ideal seria capaz de detectar todos os semáforos sem encontrar nenhum FP. Em outras palavras, em um sistema ideal os valores de precisão e *revocação* deveriam ser iguais a 1. Nós usamos estas métricas para medir o desempenho de nosso detector de semáforos.

Para avaliar o desempenho do reconhecedor do estado de semáforos nós usamos apenas a precisão, uma vez que este opera como um classificador que informa apenas se o estado do semáforo que lhe foi apresentado como entrada está aberto ou fechado. Para cada semáforo de entrada sua saída pode ser um:

- Acerto (*Hit*): o estado do semáforo foi classificado de forma correta;
- Erro (*Miss*): o estado do semáforo foi classificado de forma incorreta.

Com esses dois valores pode ser calculado o desempenho do reconhecedor do estado de semáforos na forma de sua precisão, dada pela Equação 8:

$$Precisão = \frac{Hit}{Hit + Miss} \qquad \textbf{Equação 8}$$

Finalmente, uma outra métrica relevante para sistemas de detecção e reconhecimento de semáforos é o tempo de detecção e/ou reconhecimento. Neste trabalho, para medir o tempo nestes casos nós empregamos a função *gettimeofday()* do Linux, com resultados sumarizados em milissegundos.

5 EXPERIMENTOS E RESULTADOS

Neste capítulo são apresentados os experimentos realizados com o TLDSR e os seus resultados. Na Seção 5.1 são discutidos os experimentos realizados com seu detector de semáforos. Os experimentos realizados com seu reconhecedor do estado de semáforos são discutidos na Seção 5.2. Por fim, na Seção 0, são apresentados os experimentos realizados com o TLDSR como um todo (detecção e reconhecimento simultâneos).

5.1 Experimentos de Detecção de Semáforos em Imagens

Nesta seção apresentamos os resultados dos experimentos realizados com o nosso detector de semáforos. Para a realização destes experimentos, nós utilizamos a base de dados de semáforos descrita na Seção 4.6 e as métricas apresentadas na Seção 4.7.

Com o intuito de reduzir o tempo de resposta do nosso detector de semáforos, as imagens da base de dados não foram utilizadas no seu tamanho original. Ao invés disto, um recorte foi feito na região central e superior das imagens, conforme ilustrado na Figura 26. Na figura, os pontos P_1 e P_2 delimitam a região da imagem que foi recortada para servir de entrada para o nosso detector de semáforos. Com o corte, a quantidade de *pixels* que o algoritmo precisa processar passou de 1280x960 *pixels* para 640x480 *pixels*, ou seja, um quarto do tamanho original. O recorte feito não trouxe perdas no desempenho do algoritmo, pois nenhum semáforo se encontrava na região inferior da imagem ou nas suas extremidades removidas no subconjunto de teste.

A saída de um detector é um vetor de regiões de interesse (ROIs) e, para determinar se um semáforo foi detectado ou não, nós utilizamos o coeficiente de similaridade Jaccard (apresentado na Seção 4.7) medido entre as regiões de interesse detectadas e aquelas constantes da *ground truth.* Um coeficiente de similaridade Jaccard maior ou igual a 0,6 entre um par ROI-*ground truth* é considerado como um semáforo detectado corretamente.

Figura 26 - Exemplo de imagem recortada para realizar a detecção.

Para a realização do treinamento do detector de semáforos usando o subconjunto de imagens treino, nós desenvolvemos um programa UNIX separado de CARMEN denominado *training.* Este programa utiliza dois programas já implementados em OpenCV (http://www.opencv.org) para realizar o treinamento do detector Viola-Jones, que são *opencv_createsamples* e *opencv_traincascade*. O *opencv_createsamples* cria as amostras positivas que são usadas no *opencv_traincascade*. O *opencv_traincascade* realiza o treinamento e gera arquivos no formato XML que nós usamos na validação e teste.

O *training* recebe como entrada quatro parâmetros. O primeiro é o nome de um arquivo contendo uma lista com o nome de imagens positivas, isto é, que possuem semáforos. O segundo é o nome de um arquivo contendo uma lista com as imagens negativas, isto é, que não possuem semáforos. O terceiro é o nome do arquivo com a *ground truth* das imagens listadas no arquivo cujo nome é primeiro parâmetro. O

quarto parâmetro é o nome de um arquivo que contém os parâmetros do *opencv_createsamples* que consideramos relevantes para este trabalho – ver Tabela X, abaixo. O quinto parâmetro é o nome de um arquivo que contém todos os parâmetros do *opencv_traincascade*. Finalmente, o sexto parâmetro é o nome do arquivo de saída do *opencv_traincascade*. Este arquivo contém a arvore de decisão das árvore de treinamento das *cascades* (ver Seção **Erro! Fonte de referência não encontrada.**) no formato XML. Todos os parâmetros dos programas *opencv_createsamples* e *opencv_traincascade*, parte do OpenCV, são detalhados em http://docs.opencv.org/doc/user_guide/ug_traincascade.html.

Exemplo de linha de comando usada para invocar o *training*:

./training <imagens positivas> <imagens negativas> <ground truth> <parâmetros *opencv_createsamples*> <parâmetros *opencv_traincascade*> <arquivo de saída xml>

Para avaliar o desempenho do nosso detector após seu treinamento, nós desenvolvemos dois programas UNIX separados de CARMEN denominados *testing* e *result.*

O programa *testing* recebe como entrada três parâmetros. O primeiro é um arquivo com uma lista de nomes de imagens de teste do detector de semáforos. O segundo é um número, que pode ser 1 ou 0, que indica se o usuário deseja usar o *testing* no modo interativo (1) ou não (0) (ver descrição destes modos de operação abaixo). O terceiro parâmetro é o nome do arquivo de saída, que é um arquivo com a mesma estrutura do gerado pelo *generate_gt* (ver Seção 4.6).

No modo interativo do programa *testing*, para cada imagem de teste, o *testing* abre uma janela com a imagem e os semáforos eventualmente detectados destacados nela com um retângulo vermelho – o ROI do semáforo. Além disso, o *testing* salva o ROI no arquivo de saída e aguarda até que qualquer tecla seja pressionada. Após ser pressionada uma tecla ele exibe a próxima imagem e os seus eventuais semáforos detectados e o processo continua até que todas as imagens sejam exibidas, quando, então, o *testing* termina sua execução. No modo não-interativo, para cada imagem de teste, *o testing* apenas salva os eventuais ROIs detectados no arquivo de saída.

Exemplo de linha de comando usada para invocar o *testing*:

./testing <arq uivo com a lista das imagens de teste> <1|0> <arquivo de saída>

O *result* realiza a comparação entre as ROIs de saída do *testing* com as ROIs de *ground truth* correspondente à lista de imagens de entrada do *testing*. Este programa recebe como entrada quatro parâmetros. O primeiro é o nome do seu arquivo de saída (ver detalhes sobre o conteúdo deste arquivo abaixo). O segundo é o nome do arquivo com a *ground truth*. O terceiro é o nome de um arquivo com uma lista de nomes de imagens de teste do detector de semáforos (o mesmo usado com o programa *testing*). O quarto é o nome do arquivo de saída gerado pelo programa *testing*.

O arquivo de saída do *result* contém cinco linhas com as seguintes informações:

- Precisão do detector de semáforos;
- Revocação (*recall*) *do* detector de semáforos;
- Total de TP;
- Total de FN;
- Total de FP.

Um exemplo de saída do programa *result*:

0.959044
0.541425
281
238
12

Exemplo de linha de comando usada para invocar o *result*:

./result <arquivo de saída> <ground truth> <lista de imagens de teste> <arquivo de saída gerado pelo *testing*>

5.1.1 Experimentos de Validação do Detector de Semáforos

O nosso detector de semáforos, por ser baseado na implementação do algoritmo Viola Jones do OpenCV que possui vários parâmetros, possui diversos parâmetros cujos valores impactam no seu desempenho. A fim de escolher um bom conjunto de valores para estes parâmetros, nós realizamos experimentos de validação (*tunning*), nos quais os parâmetros de treinamento do Viola-Jones que consideramos relevantes para o problema foram examinados para verificar que valores destes parâmetros produziriam o melhor desempenho de detecção, isto é, melhor precisão e revocação do detector.

Nos experimentos de validação foram utilizados os três programas mencionados anteriormente (*training*, *testing* e *result*), juntamente com um script que permitiu a geração de arquivos distintos com as entradas destes programas para cada conjunto de parâmetros examinado. Os parâmetros que consideramos relevantes foram **w**, **h**, **minHitRate** e **maxFalseAlarmRate**, cujo impacto no desempenho de nosso detector de semáforos examinamos em sequência a seguir. Exceto quando descrito de forma diferente, os valores empregados para estes parâmetros foram: **w** = 9, **h** = 20, **minHitRate** = 0,999 e **maxFalseAlarmRate** = 0,5.

Os parâmetros w e h definem a largura e altura, respectivamente, para as quais as ROIs das *ground truths* usadas no treinamento serão escaladas para serem usadas pelo algoritmo Viola-Jones. Os valores examinados para w e h são apresentados na Tabela 2, juntamente com as medidas de Revocação, Precisão, TP, FN, FP e Precisão x Revocação (ver Seção 4.7) observadas quando treinamos nosso detector de semáforos com o subconjunto de treinamento e testamos seu desempenho com o subconjunto de validação da base de dados de imagens apresentada na Seção 4.6.

O desempenho máximo possível é aquele quando a precisão é máxima (igual a 1) e a revocação também é máxima (igual a 1). Na verdade, a Precisão e a Revocação podem ser examinadas em conjunto de forma gráfica, onde a área máxima neste gráfico Revocação x Precisão, que vai do ponto (0, 0) até um ponto associado ao desempenho de uma dada configuração, representa o melhor desempenho combinado de precisão e revocação. Como a Tabela 2 mostra, à medida que

aumentamos a área **w** × **h**, o desempenho (Rev. × Prec.) aumenta. Contudo, ele chega a um máximo quando **w** ≅ 10 e **h** ≅ 20. A Figura 27 mostra o gráfico Revocação × Precisão para os diversos valores de **w** e **h** examinados.

Tabela 2 - Parâmetros largura w e altura h e seu efeito no desempenho do detector de semáforos.

W	H	Revocação	Precisão	TP	FN	FP	Rev. x Prec.
5	10	0,139	0,533	72	447	63	0,074
6	12	0,137	0,807	71	448	17	0,110
7	14	0,212	0,821	110	409	24	0,174
8	16	0,183	0,905	95	424	10	0,166
9	18	0,241	0,812	125	394	29	0,195
10	20	0,258	0,870	134	385	20	0,225
11	22	0,277	0,629	144	375	85	0,174
12	24	0,195	0,849	101	418	18	0,165
13	26	0,164	0,904	85	434	9	0,148
14	28	0,146	0,905	76	443	8	0,132

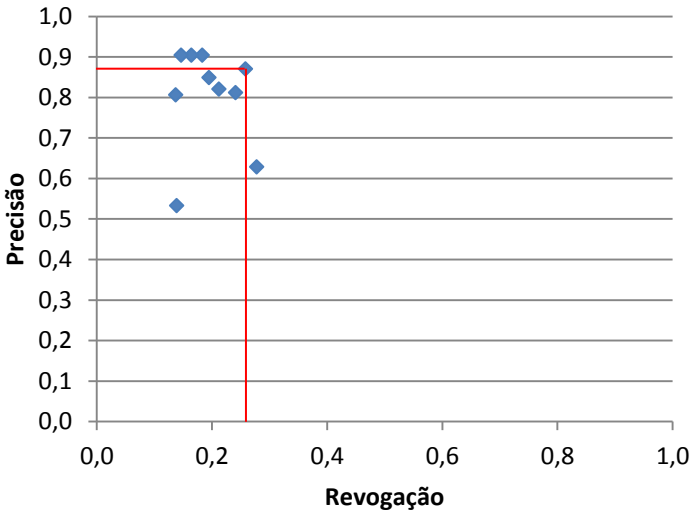


Figura 27 – Revocação x Precisão para diversos valores de w × h. O par w × h de maior área é destacado no gráfico.

Como o gráfico da Figura 27 mostra, existem pares **w** × **h** com desempenhos em termos de Precisão ou Revocação melhores que o par 10 × 20. Isso sugere que existem valores próximos de w e h que produzem melhor desempenho que o par 10 × 20. Para investigar esta hipótese, nós realizamos novos experimentos variando

w no intervalo de 9 a 11, e **h** no intervalo de 16 a 25. O gráfico da Figura 28 – Revocação × Precisão para diversos valores de w × h. sumariza o resultado destes experimentos.

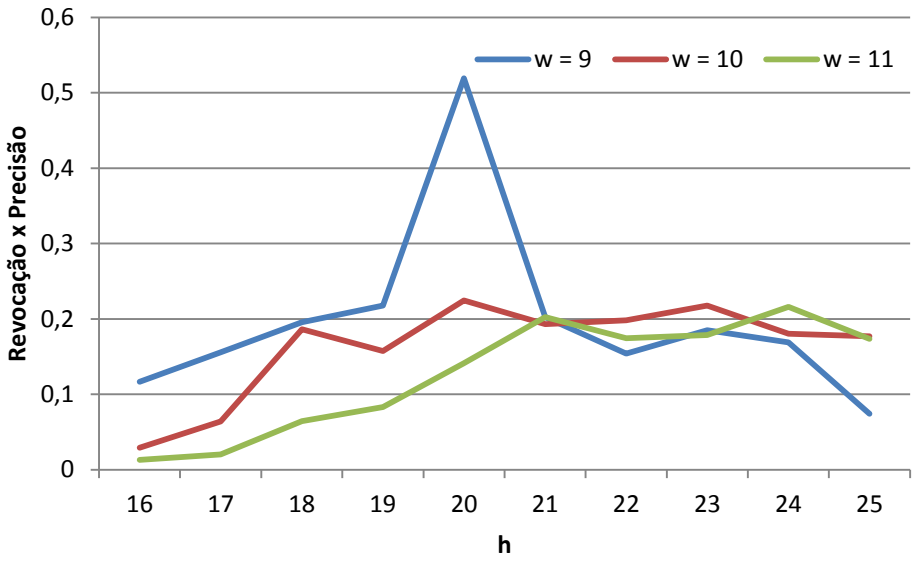


Figura 28 – Revocação × Precisão para diversos valores de w × h.

No gráfico da Figura X o eixo x representa os diversos valores de **h** examinados, enquanto que o eixo y representa o produto Revocação × Precisão. No gráfico são apresentadas três curvas, uma para cada valor de **w** examinado. Como o gráfico mostra, o par **w** × **h** de melhor desempenho foi o com valor 9 × 20. Assim, estes foram os valores de **w** e **h** escolhidos e usados nos demais experimentos deste trabalho.

O próximo parâmetro de treinamento do Viola-Jones que foi examinado foi o **minHitRate**. Este parâmetro especifica a taxa de acerto mínima desejada para cada fase do detector (ver Seção 2.1). A Tabela 3 e a Figura 29 mostram o impacto do **minHitRate** no desempenho do detector de semáforos.

Como a Tabela 3 mostra, o desempenho do detector cresce com **mimHitRate** até que o valor deste parâmetro chegue a 0,999, quando estabiliza. Assim, escolhemos este valor para **mimHitRate**. Com **mimHitRate** neste valor e **w** × **h** = 9 × 20, o desempenho do detector melhora significativamente, como uma comparação entre a Figura 27 e a Figura 29 mostra.

Tabela 3 - Parâmetro minHitRate e seu efeito no desempenho do detector de semáforos.

minHitRate	Revocação	Precisão	TP	FN	FP	Rev. x Prec.
0,99	0,023	1,000	12	507	0	0,023
0,995	0,064	0,971	33	486	1	0,062
0,999	0,541	0,959	281	238	12	0,519
0,9995	0,541	0,959	281	238	12	0,519
0,9999	0,541	0,959	281	238	12	0,519
0,99995	0,541	0,959	281	238	12	0,519

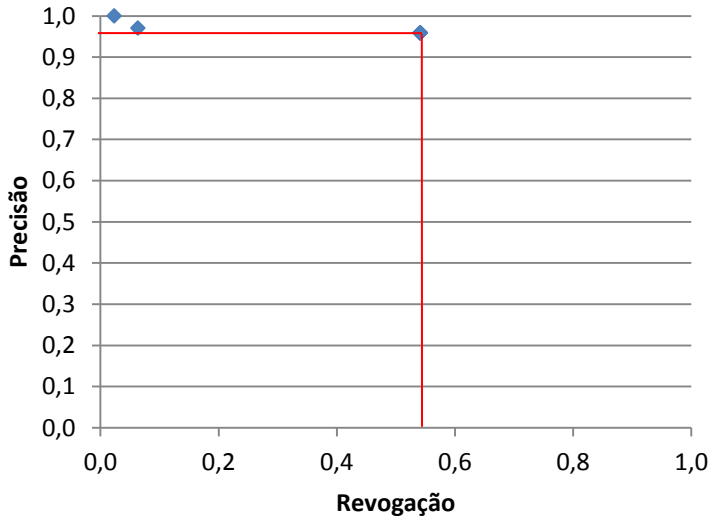


Figura 29 – Precisão x Revocação para diversos valores de minHitRate.

O último parâmetro de treinamento do Viola-Jones examinado foi o **maxFalseAlarmRate**. Este parâmetro especifica a taxa de alarme falso máxima desejada para cada estágio do detector (ver Seção 2.1). A Tabela 4 e a Figura 30 mostram o impacto do **maxFalseAlarmRate** no desempenho do detector de semáforos.

Como a Tabela 4 e o gráfico da Figura 30 mostram, o melhor valor de **maxFalseAlarmRate** é 0,5, muito embora a melhor precisão ocorra com **maxFalseAlarmRate** igual a 0,4 (ver área para cada ponto da Figura 30).

Tabela 4 - Parâmetro maxFalseAlarmRate e seu efeito no desempenho do detector de semáforos.

maxFalseAlarmRate	Revocação	Precisão	TP	FN	FP	Rev. x Prec.
0,4	0,175	0,989	91	428	1	0,173
0,5	0,541	0,959	281	238	12	0,519
0,6	0,210	0,712	109	410	44	0,150
0,7	0,403	0,043	209	310	4596	0,018
0,8	0,418	0,003	217	302	70582	0,001

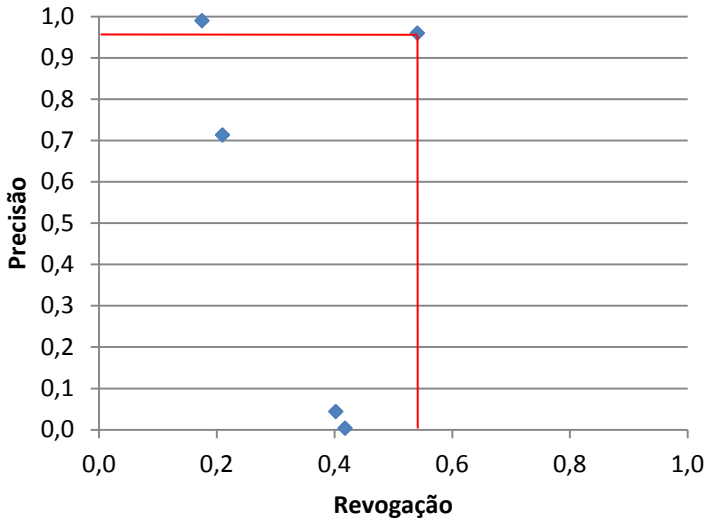


Figura 30 – Precisão x Revocação para diversos valores de maxFalseAlarmRate.

Com os melhores valores dos parâmetros examinados, isto é, **w** = 9, **h** = 20, **minHitRate** = 0,999 e **maxFalseAlarmaRate** = 0,5, os resultados obtidos no processo de validação de nosso detector de semáforos foram os da Tabela 5.

Tabela 5 - Resultados da validação do nosso detector de semáforos.

Imagens	Revocação	Precisão	TP	FN	FP	Rev. x Prec.
659	0,541	0,959	281	238	12	0,519

5.1.2 Experimento de Teste do Detector de Semáforos

O experimento de teste para avaliar o desempenho de nosso detector de semáforos foi equivalente a um dos experimentos realizados com cada conjunto de parâmetros de validação. A única diferença foi que, no caso do experimento de teste, o detector foi treinado com os subconjuntos de treino e validação de nossa base de dados, ao invés de apenas com o subconjunto de treino. Os parâmetros do algoritmo Viola-Jones empregados no treino foram os melhores parâmetros identificados nos experimentos de validação, isto é, **w** = 9, **h** = 20, **minHitRate** = 0,999 e **maxFalseAlarmRate** = 0,5.

O desempenho final de nosso detector de semáforos é apresentado na Tabela 6. Como a tabela mostra, nosso sistema detectou pouco mais da metade dos semáforos, mas com uma alta precisão. Alguns dos semáforos detectados (TP) podem ser examinados na Figura 31. Nessa figura é possível verificar que o detector consegue detectar semáforos de tamanhos variados e em estados diferentes. Também é possível constatar que, em alguns casos: (i) o semáforo detectado possui um tamanho bastante reduzido (as imagens da figura estão na escala original); (ii) não é possível determinar qual é o estado do semáforo; (iii) o semáforo não está muito visível.

Tabela 6 - Resultados dos experimentos do nosso detector de semáforos.

Imagens	Revocação	Precisão	TP	FN	FP	Rev. x Prec.
659	0,523	0,977	295	269	7	0,511

Figura 31 – Exemplos de semáforos detectados corretamente pelo nosso detector (TP, ROIs gerados automaticamente por nosso detector).

A maioria dos falsos positivos (FP) detectados foram semáforos voltados para o sentido contrário à pista trafegada pela IARA, como pode ser visto na Figura 32. Além desses, as outras regiões correspondem a imagens em que o que pode ser visualizado é uma região parecida com um semáforo.

Figura 32 – Regiões detectadas como sendo semáforos e que, na verdade, não correspondem a semáforos (FP, ROIs gerados automaticamente por nosso detector).

Quase metade dos semáforos não foram detectados (FN). Alguns dos problemas que causaram esta alta taxa de falsos negativos foram: (i) baixa qualidade da câmera; (ii) quantidade pequena de amostras de treinamento; (iii) falta de padronização dos semáforos; (iv) tamanho reduzido da imagem do semáforo na imagem de entrada do detector; (v) tipo de semáforo não treinado; e (vi) problemas de iluminação da imagem (principalmente ofuscamento pelo Sol). Todos esses problemas podem ser visualizados na Figura 33, onde são exibidos exemplos de semáforos não detectados (FN).

Figura 33 - Exemplos de semáforos não detectados (ROIs da *ground truth* na escala original).

5.2 Experimentos de Reconhecimento do Estado de Semáforos

Nesta seção são apresentados os resultados dos experimentos realizados com o nosso reconhecedor do estado de semáforos. Para a realização destes experimentos, nós utilizamos a base de dados de semáforos descrita na Seção 4.6 e as métricas apresentadas na Seção 4.7.

Nosso reconhecedor do estado de semáforos assume que o semáforo já foi detectado na imagem de entrada do TLDSR, isto é, que a ROI que o contém já foi identificada. Assim, nos experimentos de validação e teste do reconhecedor, nós usamos as ROIs da *ground truth* da base de dados da Seção 4.6.

Para realizar a cópia das ROIs das imagens da base de dados foi criado um programa UNIX separado de CARMEN denominado *crop_roi*. Este programa recebe como entrada uma *ground truth* filtrada (de treino, validação ou teste – ver Seção 4.6) contendo o nome das imagens e as ROIs a serem recortadas, e as salva no mesmo diretório em que foi executado. O *crop_roi* possui um contador interno que é usado para computar o valor do índice (nome) da imagem da ROI recortada, que começa em 1 e cresce sequencialmente. Um exemplo de nome de arquivo salvo:

25.png

Exemplo de linha de comando usada para invocar o *crop_roi*:

./crop_roi <arquivo de *ground truth*>

Com as ROIs recortadas dos subconjuntos de imagens de treino, teste e validação de nossa base de dados, foi realizada uma anotação manual do estado do semáforo de cada uma delas (das ROIs). Esta anotação foi armazenada em 4 arquivos: red.txt, green.txt, test.txt e validation.txt.

O arquivo red.txt é composto de linhas onde, cada uma, contém o nome de uma ROI com semáforo no estado vermelho do subconjunto de treinamento. O arquivo green.txt é equivalente ao red.txt, mas para os semáforos no estado verde do subconjunto de treinamento. O arquivo test.txt é composto de linhas com as seguintes informações, separadas pelo caractere espaço (“ ”): (i) nome da ROI; e (ii) um número, que pode ser 1 ou -1, que indica, respectivamente, que o semáforo

contido na ROI está no estado vermelho (fechado) ou no estado verde (aberto). O arquivo test.txt contém todas as ROIs do subconjunto de teste. O arquivo validation.txt é equivalente ao arquivo test.txt, mas contém todas as ROIs do subconjunto de validação.

5.2.1 Experimentos de Validação do Reconhecedor do Estado de Semáforos

O nosso reconhecedor do estado de semáforos, por ser baseado na implementação do algoritmo SVM da biblioteca DLIB (http://www.dlib.net), possui dois parâmetros cujos valores impactam no seu desempenho: C e γ. O parâmetro C indica a penalidade (margem máxima, ver Seção 2.2). O parâmetro γ indica a largura do núcleo do kernel (ver Seção 2.2). A fim de escolher um bom conjunto de valores para estes parâmetros, nós realizamos experimentos de validação (*tunning*), nos quais os valores destes parâmetros foram examinados para identificar quais produziriam o melhor desempenho de reconhecimento.

Para a realização do treinamento e validação do nosso reconhecedor, nós desenvolvemos um programa UNIX separado de CARMEN denominado *svm_train*. Este programa recebe como entrada 7 parâmetros. O primeiro é o arquivo red.txt mencionado na seção anterior (ROIs com semáforos no estado vermelho do subconjunto de treinamento). O segundo é o arquivo green.txt, também mencionado na seção anterior (ROIs com semáforos no estado verde do subconjunto de treinamento). O terceiro é o arquivo validation.txt, também mencionado na seção anterior (todas as ROIs do subconjunto de validação). O quarto e o quinto correspondem, respectivamente, à uma largura **w** e à uma altura **h** de imagem (ver abaixo). O sexto e sétimo correspondem, respectivamente, aos valores de C e γ.

Exemplo de linha de comando usada para invocar o *svm_train*:

./svm_train red.txt green.txt valiadation.txt <largura de escalamento> <altura de escalamento> C γ

Nosso reconhecedor de semáforos opera com ROIs de entrada de dimensões fixas. Assim, estas dimensões precisam ser passadas para o *svm_train*. Nós usamos as dimensões identificadas como sendo as melhores para o detector de semáforos (**w** = 9 e **h** = 20).

Com o intuito de realçar o estado do semáforo e, assim, facilitar a tarefa do nosso reconhecedor baseado em SVM, o *svm_train* escala as ROIs para as dimensões **w** x **h** e as pré-processa. O pré-processamento realizado nas ROIs é o seguinte: (i) com a ROI escalada, é realizada a extração de cada canal de cor (R, G e B) da ROI, obtendo uma imagem distinta para cada canal; (ii) para cada pixel i destas imagens é usada a Equação 9, em que R e G denotam os canais vermelho e verde, respectivamente, obtidos durante a extração dos canais. Os resultados do pré-processamento para semáforos no estado verde e vermelho podem ser apreciados na Figura 34 – na Figura 34(a) estão representados alguns semáforos vermelhos pré-processados e na Figura 34(b) alguns semáforos verdes pré-processados.

$$\frac{(R_i - G_i) + 255}{2} \qquad \textbf{Equação 9}$$

(a) (b)

Figura 34 – Exemplos de: (a) semáforos vermelhos com o pré-processamento. (b) semáforos verdes com o pré-processamento.

Os valores dos *pixels* dessas imagens foram normalizados e as imagens resultantes foram usadas como entrada do algoritmo SVM do nosso reconhecedor do estado de semáforos.

Nos experimentos de validação foi utilizado o programa mencionado anteriormente (*svm_train*), juntamente com um script que permitiu a geração de arquivos distintos com as entrada deste programa para cada conjunto de parâmetros examinado. Os valores examinados para C e γ foram (mesma faixa de valores para os dois): 0,00001; 0,00005; 0,00025; 0,00125; 0,00625; 0,03125; 0,15625 e 0,78125. Assim,

avaliamos 64 combinações de parâmetros. A Figura 27 mostra o gráfico da Precisão do nosso reconhecedor do estado de semáforos para os diversos valores de C e γ examinados.

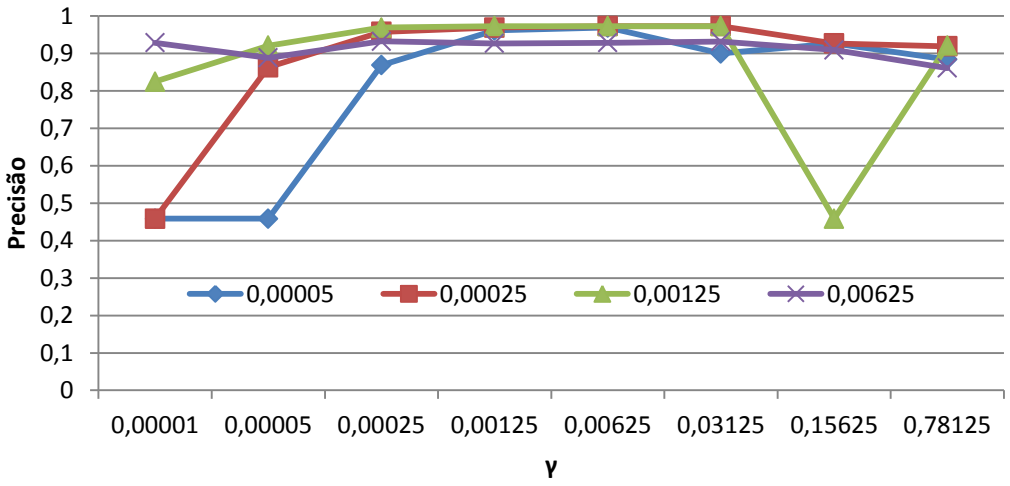


Figura 35 – Precisão para os diversos valores combinados de C e γ.

Como o gráfico da Figura 35 mostra, existe uma faixa valores de C e γ que apresenta o mesmo desempenho máximo. Assim, escolhemos o ponto central desta faixa para C e γ, isto é, a combinação de C = 0,00125 e γ = 0,00125. Estes são os valores de C e γ usados nos demais experimentos deste trabalho.

Com os melhores valores de C e γ encontrados, o desempenho de nosso reconhecedor de semáforos com o subconjunto de validação é o apresentado na Tabela 5.

Tabela 7 - Resultados da validação do nosso reconhecedor de estados de semáforos.

Imagens	Semáforos	Precisão	Acertos	Erros
659	519	0,973	505	14

5.2.2 Experimentos de Teste do Reconhecedor do Estado de Semáforos

O experimento de teste para avaliar o desempenho de nosso reconhecedor do estado de semáforos foi equivalente a um dos experimentos realizados com cada conjunto de parâmetros de validação. A única diferença foi que, no caso do experimento de teste, o reconhecedor do estado de semáforos foi treinado com os subconjuntos de treino e validação de nossa base de dados, ao invés de apenas com o subconjunto de treino. Os parâmetros do SVM empregados no treino foram os melhores parâmetros identificados nos experimentos de validação.

O desempenho final de nosso reconhecedor do estado de semáforos é apresentado na Tabela 8. O reconhecedor obteve uma alta precisão, errando o estado dos semáforos presentes em poucas ROIs. Alguns exemplos de ROI nas quais o reconhecedor acertou o estado do semáforo podem ser apreciados na Figura 36. Ressalta-se que, apesar dos vários formatos dos semáforos e dos tipos de lâmpadas usados nos mesmo, o reconhecedor cumpriu bem a sua tarefa.

Tabela 8 - Resultados do reconhecedor do estado de semaforos.

Imagens	Semáforos	Precisão	Acertos	Erros
659	549	0,965	530	19

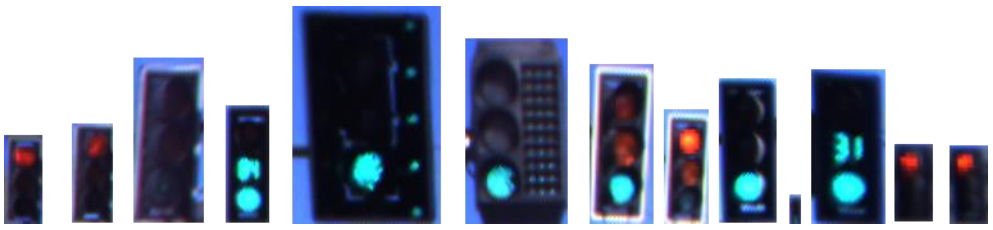


Figura 36 – Exemplos de ROI nas quais o reconhecedor acertou o estado do semáforo.

Algumas das ROIs nas quais o reconhecedor classificou incorretamente o estado do semáforo podem ser examinadas na Figura 37 e na Figura 38, em que são mostradas as ROIs originais, e as mesmas ampliadas para melhorar a sua visualização. Ressalta-se que o estado amarelo não aparece no subconjunto de treinamento, existindo somente na base de teste (ver Seção 4.6).

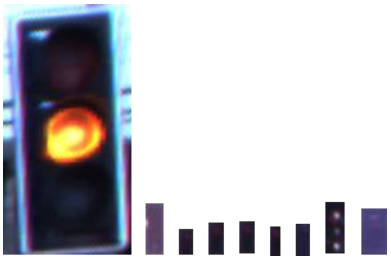


Figura 37 – Exemplos em tamanho original em que o nosso reconhecedor errou o estado do semáforo.

Figura 38 – Exemplos em tamanho ampliado em que o nosso reconhecedor errou o estado do semáforo.

5.3 Experimentos de Detecção e Reconhecimento

Nessa seção, são apresentados os resultados completos do sistema proposto nesse trabalho, o TLDSR, que confere à IARA a capacidade de detecção e de reconhecimento do estado de semáforos.

O TLDSR pode operar off-line, sobre dados advindos de um log, ou on-line. Nos dois casos, o sistema:

- Recebe uma imagem da *Front Stereo Camera* (ver Seção 4.1);
- Recorta da imagem direita do par estéreo a região indicada na Figura 26;
- Usa o detector de semáforos para compor uma lista de ROIs da região recortada;
- Escala as ROIs desta lista para 9×20 *pixels;*
- Usa o reconhecedor do estado de semáforos para reconhecer o estado do semáforo eventualmente presente em cada ROI;

- Calcula a distância até cada semáforo;
- Compõe lista de ROIs contendo semáforos, seus estados e as suas distâncias;
- Gera uma versão modificada da imagem direita original conforme descrito na Seção 4.4;
- Envia uma mensagem CARMEN com a lista supra e com a cópia da imagem direita gerada.

Um exemplo de imagem gerada pode ser visto na Figura 39. Nessa imagem podem ser observadas as seguintes informações: (i) um retângulo em torno de semáforos detectados; (ii) o estado do semáforo mais próximo indicado como um círculo na cor vermelha no canto superior esquerdo; e (iii) a distância da IARA até este semáforo mais próximo, indicada na forma de texto na parte inferior da imagem. O funcionamento do sistema TLDSR pode ser conferido no vídeo do link http://www.youtube.com/watch?v=FCw8--NsYJI.

Figura 39 - Interface do detector e reconhecedor de estados dos semáforos.

Os resultados dos experimentos com o TLDSR podem ser observados na Tabela 9. Como a tabela mostra, a taxa de detecção e reconhecimento combinados é baixa. Isso ocorre porque nosso detector de semáforos possui baixa revocação. Contudo, se considerarmos o percentual dos semáforos cujo estado foi corretamente determinado dentre aqueles detectados (última coluna da tabela), verificamos um bom desempenho do sistema.

Tabela 9 - Resultados de detecção e reconhecimento do estado de semáforos com o TLDSR.

Detectou e Acertou o Estado	Não Detectou	Percentual dos que Acertou, Dentro dos que Detectou
0,432	0,552	0,963

O tempo de execução total para cada imagem, que engloba a detecção dos semáforos e o reconhecimento do seu estado, foi em média igual a 0,035 segundos, possibilitando assim o uso do sistema em tempo real na IARA.

Um exemplo de semáforo detectado e com estado corretamente reconhecido à uma distância de aproximadamente 15 metros da IARA pode ser visto na Figura 40.

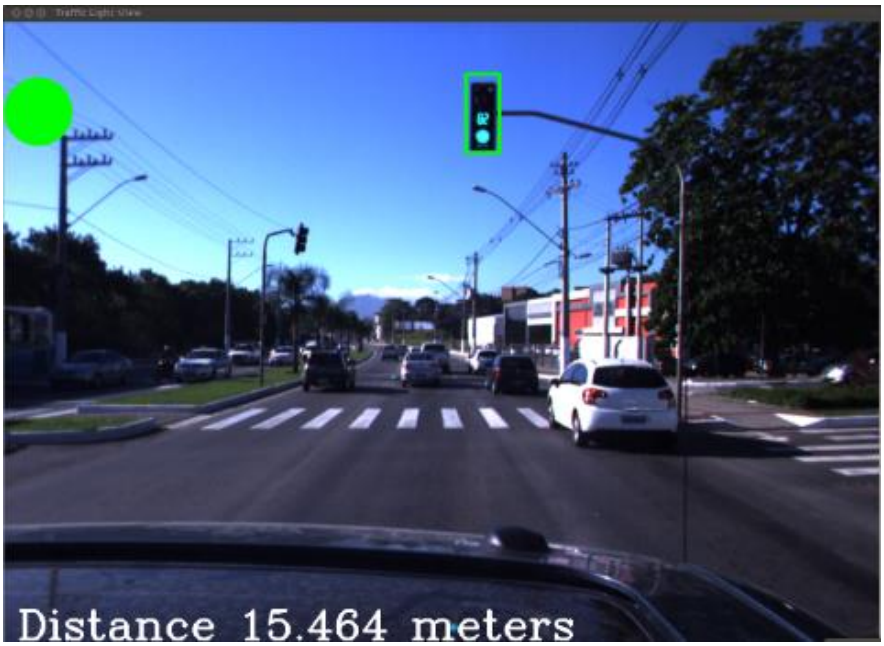


Figura 40 - Exemplo de semáforo próximo detectado e reconhecido.

A distância máxima de detecção de semáforos foi igual a, aproximadamente, 80 metros. A Figura 41 mostra um exemplo de semáforo detectado e cujo estado foi reconhecido corretamente a mais de 80 metros da IARA. A maior distância na qual o sistema detectou um semáforo e reconheceu seu estado corretamente foi 90 metros, que pode ser verificado na Figura 42.

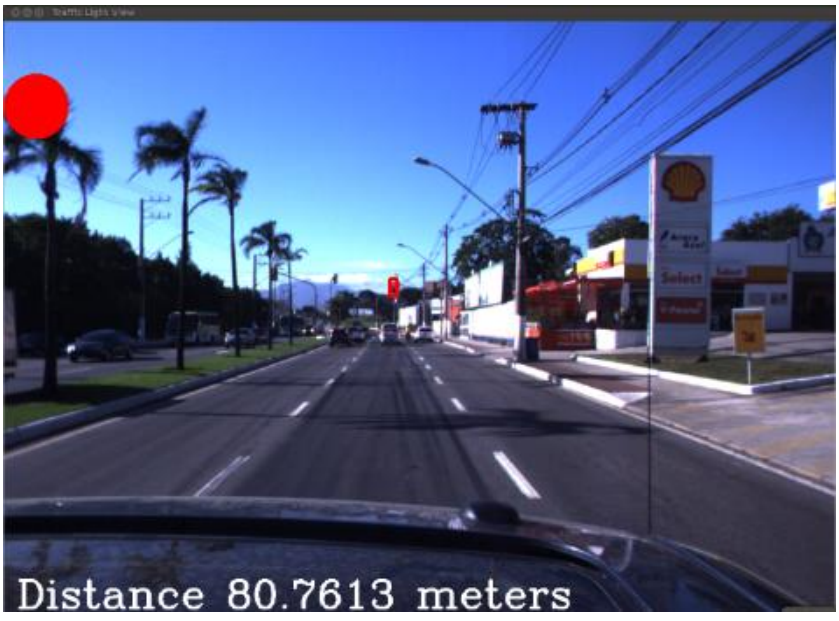


Figura 41 - Exemplo de semáforo detectado e com estado reconhecido à cerca de 80 metros de distância.

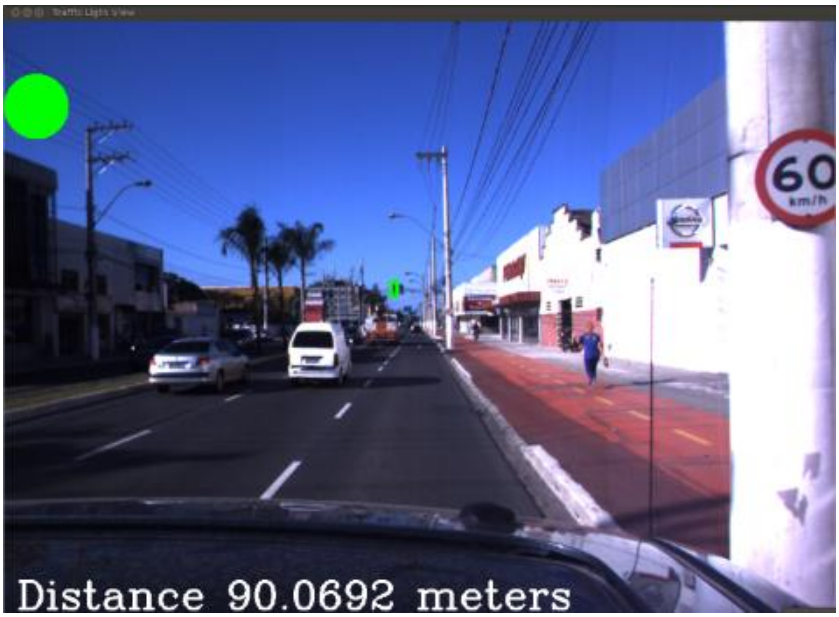


Figura 42 - Exemplo de semáforo detectado e com estado reconhecido à grande distância.

Algumas imagens possuíam brilho intenso na imagem devido ao sol. Mesmo assim o sistema conseguiu detectar semáforos e reconhecer corretamente seu estado. Um exemplo disso pode visto na Figura 43.

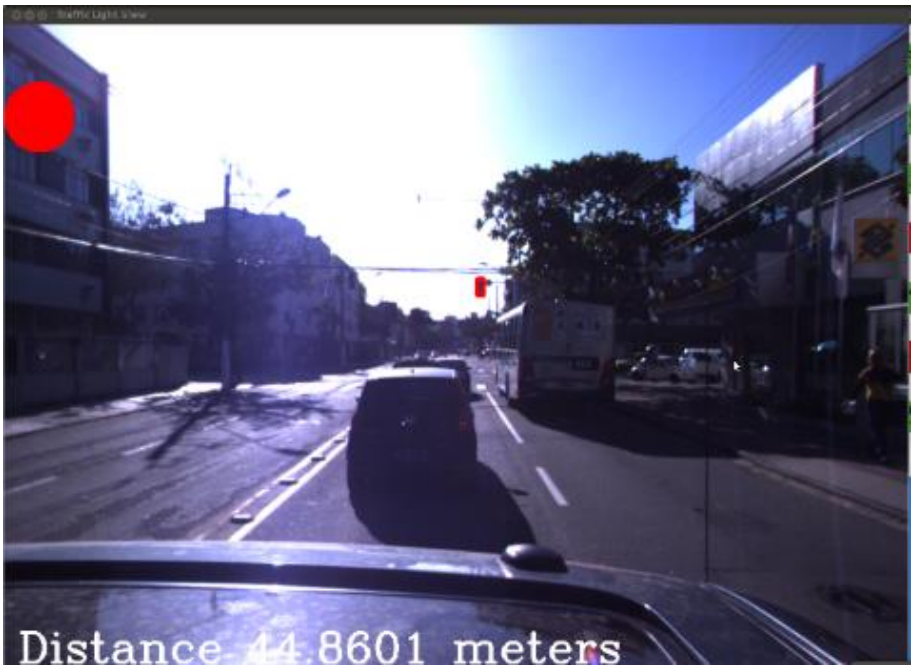


Figura 43 - Exemplo de semáforo detectado em condições de iluminação desfavoráveis.

Um caso de falha no nosso detector, mais no qual, mesmo assim, à distância até o semáforo foi informada é demonstrado na Figura 44. Nesta imagem pode-se verificar o efeito negativo da iluminação (a imagem do semáforo “mistura-se” com a da vegetação ao fundo). A elipse em azul na imagem mostra onde o semáforo deveria ter sido detectado.

Figura 44 - Caso de falha na detecção de semáforo.

6 DISCUSSÃO

Este capítulo discute o trabalho desenvolvido, apresentando os trabalhos correlatos e análise crítica do trabalho.

6.1 Trabalhos Correlatos

O problema de identificação de semáforos pode ser resolvido de diversas formas. Esta seção apresentará trabalhos que resolvem este problema utilizando diferentes abordagens.

O trabalho [5] apresenta uma implementação de detecção e reconhecimento do estado de semáforos usando três passos: segmentação de cor (para a detecção), filtragem e classificação (para o reconhecimento). A segmentação de cores usa uma tabela de referência para cada cor. Aplicando componentes de cor conectadas a imagem é filtrada para selecionar somente a região correspondente a semáforos. O classificador usa redes neurais. O sistema como um todo obteve 90% de detecção e reconhecimento e 2% de falsos positivos.

Já o trabalho [10] apresenta uma comparação entre três métodos de detecção de semáforos. O primeiro é um detector baseado em cor com 95% de reconhecimento a 25Hz. O segundo é um detector baseado em forma e filtro de comparação, com 85% de precisão a 10 Hz. O último é um classificador em *cascade* parecido com nosso trabalho, com uma taxa de 90% de reconhecimento a 2 Hz.

No trabalho apresentado em [7], eles apresentam um detector de semáforos usando informações de cores e a transformadas de Hough para detectar semáforos, com uma taxa de precisão de 86% de acertos à 0.347 segundos por imagem.

O trabalho apresentado em [11] apresenta uma abordagem baseada em detecção da luz do semáforo juntamente com modelos adaptativos de semáforos. Eles obtiveram 95,38% de precisão e 98,41% de precisão para a detecção e reconhecimento nos testes deles.

Já o trabalho [6] compara o trabalho apresentado em [11] com processos de aprendizado. Nesse artigo dois novos métodos que se utilizam de Gentle AdaBoost são apresentados. Os resultados indicam que o método apresentado em [11] é mais eficiente comparado com os de aprendizado.

O trabalho realizado em [8] apresenta uma abordagem de detecção e reconhecimento que utiliza os seguintes passos: (i) extração de cor, clusterização para a detecção); e (ii) checagem de formato circular, checagem de frame, filtragem robusta para o reconhecimento. Os resultados obtidos para detecção e reconhecimento foram 93,9% de precisão com um tempo médio de execução por imagem de 7 ms.

Todos esses métodos apresentados nos trabalhos [5] [6] [7] [8] [10] [11] utilizam de métodos de processamento de imagens para resolver o problema de detecção e reconhecimento de semáforos. Os próximos trabalhos utilizam, além dessas, informações sobre interseções (cruzamentos) das vias e mapeamento dos semáforos.

O trabalho [2] apresenta uma abordagem para o mapeamento, localização e detecção do estado de semáforos. Eles realizam um mapeamento das vias e das regiões onde estão os semáforos previamente com algoritmos de mapeamento. Ao final usam a informação anteriormente salva e realização o rastreamento do semáforo, usando apenas *templates* para falar qual o estado do semáforo. Eles obtiveram 91,7% de precisão para detectar semáforos e identificar o seu estado e 94% de precisão para achar intercessões.

O trabalho [4] apresenta uma abordagem parecida com a anterior, mais utilização o Google Maps (http://maps.google.com) para realizar o mapeamento. Além disso, eles realizam um mapeamento automático de semáforos. Realizam também, a estimativa da pose e com um método baseado em detecção de *blobs* realizam o reconhecimento do estado. Nesse trabalho, os semáforos foram detectados a uma distância de aproximadamente 100 metros com uma precisão de 98% para a detecção e o reconhecimento do estado de semáforos.

6.2 Análise Crítica do Trabalho

Este trabalho, muito embora, há nosso ver, tenha obtido resultados satisfatórios, poderia ser melhorado de diversas formas. Uma delas seria a utilização de uma câmera com melhor resolução, o que possibilitaria obter melhores imagens dos semáforos para o treino do detector e, possivelmente, melhores resultados de detecção.

Outra forma de melhorar este trabalho seria o emprego de outros algoritmos de detecção, tais como HOG+LDA [30], HOG+SVM [31], etc; a fim de comparar qual o melhor método para a detecção, visto que nosso detector detectou em média apenas 50% dos semáforos.

Uma outra forma de melhorar o desempenho do detector seria a utilização da posição esperada do semáforo na imagem, advinda do mapeamento prévio. Com essa posição, seria possível saber onde procurar o semáforo e, com isso, melhorar o desempenho de detecção.

Além de melhorias no detector, poderíamos ainda melhorar o desempenho do reconhecedor de estado de semáforos utilizando outros métodos de reconhecimento de padrões, tais como VG-RAM WNN [32], Deep Learning [33], entre outros.

Outra forma de melhorar o desempenho do sistema como um todo seria a melhoria da instalação física da câmera na IARA de modo a minimizar a ofuscação pela luz do sol. Testes preliminares envolvendo a remoção de imagens ofuscadas da nossa base de dados indicam uma melhora na precisão do sistema como um todo para o subconjunto de teste de 96,5% para 98,2%.

7 CONCLUSÕES

Neste capítulo é apresentado um breve sumário deste trabalho, suas principais conclusões e direções para trabalhos futuros.

7.1 Sumário

A identificação de semáforos (a detecção do semáforo e o reconhecimento do seu estado) é um tema de pesquisa muito importante para a indústria automotiva. Essa identificação tem como objetivo alertar condutores desatentos para a mudança do estado dos semáforos e fazer com que os cruzamentos sejam mais seguros, sendo essencial para navegação assistida e autônoma.

Nesse trabalho foram investigados modelos matemáticos computacionais para solucionar o problema de identificação de semáforos. Para tal, empregamos o algoritmo Viola-Jones para o detector, e SVM para o reconhecedor do estado. Experimentos foram feitos para avaliar o desempenho do detector e do reconhecedor e do sistema completo, envolvendo detecção e reconhecimento.

7.2 Conclusões

Neste trabalho nós desenvolvemos o sistema TLDSR, que foi integrado à plataforma robótica da UFES, denominada IARA, por meio do *framework* CARMEN. O TLDSR é capaz de detectar e reconhecer o estado de semáforos *on-line*. Para avaliar o TLDSR foi criada uma base de dados, contendo imagens utilizadas no treinamento e em experimentos de validação e teste. Esta base de dados foi disponibilizada na Internet.

O sistema foi avaliado com experimentos para a detecção, experimentos para o reconhecimento e experimentos com o TLDSR completo integrado à IARA. O detector de semáforos obteve taxas de 95% de precisão e 50% de revocação. Esses resultados do detector indicam que ele detecta semáforos com alta precisão (poucos

falso-positivos), mais deixa de detectar cerca de metade dos semáforos encontrados (muitos falso-negativos). Isso ocorreu devido à: (i) baixa qualidade da câmera; (ii) quantidade pequena de amostras de treinamento; (iii) falta de padronização dos semáforos; e (iv) problemas de iluminação durante a captura das imagens (ofuscamento da câmera pelo sol).

O reconhecedor do estado de semáforos alcançou uma taxa de 96,5% de precisão. Este desempenho do reconhecedor indica que ele erra pouco o estado do semáforo. Os semáforos cujo estado foi reconhecido de forma incorreta frequentemente tinham sua imagem ofuscada pelo sol ou estavam muito distantes da câmera.

No geral, o TLDSR é capaz de detectar e reconhecer o estado de semáforos a distâncias variando entre 15 e 80 metros em um tempo igual a aproximadamente 0,035 segundos por imagem. O percentual dos semáforos cujo estado foi corretamente determinado dentre aqueles detectados foi igual a 96,3%, o que demonstra um bom desempenho do sistema. Esses resultados combinados indicam que o sistema pode ser usado na IARA para informar sobre a existência e o estado de semáforos.

Ainda há muito a ser melhorado, mas os resultados obtidos podem ser considerados satisfatórios se forem integrados no tempo para melhorar sua confiança.

7.3 Trabalhos Futuros

Uma direção para trabalhos futuros seria a utilização de uma câmera com melhor resolução que a usada neste trabalho. Usar uma câmera de melhor resolução permitiria obter mais informações sobre os semáforos para serem utilizadas na detecção dos semáforos, o que poderia melhorar o desempenho do nosso detector de semáforos.

Outras direções para trabalhos futuros seriam investigar: outros métodos para a detecção de semáforos, outros métodos para o reconhecimento do estado de semáforos, o impacto das imagens com uma baixa resolução, formas redução do

ofuscamento da câmera pelo sol, e de utilização da posição esperada do semáforo na imagem.

8 REFERÊNCIAS BIBLIOGRÁFICAS

1. GRADINESCU, V. et al. **Adaptive Traffic Lights Using Car-to-Car Communication**. Vehicular Technology Conference, 2007. VTC2007-Spring. IEEE 65th. Dublin: [s.n.]. 2007. p. 21-25.

2. LEVISON, J. et al. **Traffic Light Mapping, Localization, and State Detection**. International Conference on Robotics and Automation. Shanghai: IEEE Computer Science. 2011.

3. CHUNG, Y.-C.; WANG, J.-M.; CHEN, S.-W. **A Vision-Based Traffic Light Detection System at Intersections**. Journal of Taiwan Normal University: Mathematics, Science & Technology. [S.l.]: [s.n.]. 2002. p. 67-86.

4. FAIRFIELD, N.; URMSON, C. **Traffic Light Mapping and Detection**. International Conference on Robotics and Automation. Shangai: [s.n.]. 2011. p. 5421-5426.

5. FRANK, U. et al. **Autonomous Driving Goes Downtown**. [S.l.]: [s.n.]. 1998. p. 40-48.

6. DE CHARETTE, R.; NASHASHIBI, F. **Traffic Light Recognition using Image Processing Compared to Learning Process**. Internation Conference on Intelligence Robots and Systems. St. Louis: [s.n.]. 2009. p. 333-338.

7. OMACHI, M.; OMACHI, S. **Traffic light detection with color and edge information**. International Conference on Computer Science and Information Technology. Beijing: [s.n.]. 2009. p. 284-287.

8. PARK, J.-H.; JEONG, C.-S. **Real-time Signal Light Detection**. Internation Journal of Signal Processing, Image Processing and Pattern Recognition. [S.l.]: [s.n.]. 2009. p. 1-10.

9. SIOGKAS, G.; SKODRAS, E.; DERMATAS, E. **Traffic Light Detection in Adverse Conditions Using Color, Symmetry and Spatiotemporal**

Information. International Conference on Computer Vision Theory and Applications. Parma: [s.n.]. 2011. p. 620-627.

10. LINDNER, F.; KRESSEL, U.; KAELBERER, S. **Robust Recognition of Traffic Signs**. Intelligent Vehicles Symposium. Parma: [s.n.]. 2004.

11. DE CHARETTE, R.; NASHASHIBI, F. **Real Time Visual Traffic Lights Recognition Based on Spot Light Detection and Adaptive Traffic Lights Templates**. Intelligent Vehicles Symposium. Xi'an: [s.n.]. 2009. p. 358-363.

12. VIOLA, P.; JONES, M. **Rapid Object Detection using a Boosted Cascade of Simple Features**. 2013 IEEE Conference on Computer Vision and Pattern Recognition. Los Alamitos: IEEE Computer Society. 2001. p. 511-518.

13. CORTES, C.; VAPNIK, V. Support-vector networks. **Machine Learning**, v. 20, n. 3, p. 273-297, set. 1995.

14. LIENHART, R.; KURANOV, A.; PISAREVSKY, V. Empirical Analysis of Detection Cascades of Boosted Classifiers for Rapid Object Detection. **DAGM 25th Pattern Recognition Symposium**, p. 297-304, 2003.

15. FREUND, Y.; SCHAPIRE, R. E. **A Decision-Theoretic Generalization of On-Line Learning**. Journal of Computer and System Sciences 55. [S.l.]: [s.n.]. 1997. p. 119-139.

16. DE SOUZA, A. F. Relatório Técnico. **LCAD**, 2013. Disponivel em: <http://www.lcad.inf.ufes.br/wiki/images/3/31/Relatorio_tecnico_parcial_20101119_20111030.pdf>. Acesso em: 31 jan. 2014.

17. THRUN, S.; BURGARD, W.; FOX, D. **Probabilistic Robotics**. Cambridge, Massachusetts: MIT Press, 2005.

18. MONTEMERLO, M. et al. Junior: The Stanford Entry in the Urban Challenge. **Journal of Field Robotics**, p. 569–597, 2008.

19. KONOLIGE, K. A gradient method for realtime robot control. **In Proc. of the**

. **IEEE/RSJ Int. Conf. on Intelligent Robots and Systems (IROS)**, 2000. 2436-2441.

20 KUFFNER, J. J.; LAVALLE, S. M. **RRT-Connect:** Na Efficient Approach to Single-
. Query Path Planning. Proceedings of the IEEE International Conference on Robotics and Automation. [S.l.]: [s.n.]. 2000. p. 995–1001.

21 CHENG, P.; LAVALLE, S. M. **Reducing Metric Sensitivity in Randomized**
. **Trajectory Design**. Proceedings of the IEEE/RSJ International Conference on Intelligent Robots and Systems. [S.l.]: [s.n.]. 2001. p. 43–48.

22 LAVALLE, S. M.; KUFFNER, J. J. Rapidly-Exploring Random Trees: Progress
. and Prospects. In: _____ **Algorithmic and Computational Robotics:** New Directions 2000 WAFR. [S.l.]: A K Peters/CRC Press, 2001. p. 293-308.

23 DOLGOV, D.; THRUN, S. Path Planning for Autonomous Vehicles in Unknown
. Semi-Structured Environments. **The International Journal of Robotics Research**, v. 29, p. 485–501, 2010.

24 VERONESE, L. P. et al. **Stereo Matching with VG-RAM Weightless Neural**
. **Networks**. Proceedings of the International Conference on Intelligent Systems Design and Applications. [S.l.]: [s.n.]. 2012. p. 309–314.

25 VERONESE, L. P. et al. **Parallel Implementations of the CSBP Stereo Vision**
. **Algorithm**. Anais do 2011 Simpósio em Sistemas Computacionais. [S.l.]: [s.n.]. 2011. p. 12–20.

26 AZEVEDO, V. B. et al. **Real-time Road Surface Mapping Using Stereo**
. **Matching, V-Disparity and Machine Learning**. Proceedings of International Joint Conference on Neural Networks (IJCNN'2013). Dallas, Texas: IEEE Computer Society. 2013. p. 2575-2582.

27 RADAELLI, R. R. **Planejamento de Movimento para Veículos Convencionais**
. **Usando Rapidly-Exploring Random Tree**. Programa de Pós-Graduação em Informática. Vitória-ES. 2013.

28 HOUBEN, S. et al. Detection of Traffic Signs in Real-World Images: The German. . **IJCCN**, Texas, 04-09 ago. 2013. 715-722.

29 DE SOUZA, A. F. et al. **Traffic Sign Detection with VG-RAM Weightless** . **Neural Networks**. Proceedings of International Joint Conference on Neural Networks (IJCNN'2013). Dallas: IEEE Computer Science. 9-13 ago. 2013.

30 WANG , G. et al. **A robust, coarse-to-fine traffic sign detection method**. . Proceedings of International Joint Conference on Neural Networks (IJCNN'2013). Dallas: IEEE Computer Sciense. 09-13 ago. 2013. p. 1-5.

31 MATHIAS, M. et al. **Traffic Sign Recognition - How Far Are We From the** . **Solution?** Proceedings of IEEE Internation Joint Conference on Neural Networks. [S.l.]: [s.n.]. 2013.

32 BERGER, M. et al. **Traffic sign recognition with VG-RAM Weightless Neural** . **Networks**. Proceedings on Intelligent Systems Design and Applications (ISDA). Kochi: [s.n.]. 2012. p. 315-319.

33 BENGIO, Y. et al. **Greedy Layer-Wise Training of Deep Networks**. [S.l.]: [s.n.]. . 2007. p. 153-160.

34 FREUND, Y.; SCHAPIRE, R. E. A Decision-Theoretic Generalization of on-Line . Learning and an Application to Boosting. **Springer-Verlag**, p. 23-27, 1995.

35 MONTEMERLO, M.; ROY, N.; THRUN, S. **Perspectives on standardization in** . **mobile robot programming:** The carnegie mellon navigation (CARMEN) toolkit. International Conference on Intelligent Robots and Systems (IROS). [S.l.]: [s.n.]. 2003. p. 2436-2441.

36 SIMMONS, R. The inter-process communication (IPC) system. Disponivel em: . <http://www.cs.cmu.edu/afs/cs/project/TCA/www/ipc/ipc.html>.

37 HOHPE, G.; WOOLF, B. **Enterprise Integration Patterns:** Designing, Building, . and Deploying Messaging Solutions. 1. ed. [S.l.]: Addison-Wesley Professional,

2003.