DESENVOLVIMENTO DE VEÍCULO AUTÔNOMO - INTELIGÊNCIA CENTRAL E ORIENTAÇÃO POR CÂMERAS

Riccardo L. Delai[1] ; Alessandra Dutra Coelho[2]

[1]Aluno de Iniciação Científica da Escola de Engenharia Mauá (EEM/CEUN-IMT);
[2]Professora da Escola de Engenharia Mauá (EEM/CEUN-IMT).

Resumo. *Este artigo descreve o desenvolvimento da inteligência central de um veículo autônomo orientado por câmeras. O sistema é responsável pela visão e pela navegação do veículo, despachando comandos para uma inteligência periférica para que o veículo faça um movimento de zigue-zague entre cones de trânsito comuns arbitrariamente posicionados. A solução elaborada utiliza técnicas de visão computacional (com a biblioteca aberta OpenCV) e inteligência artificial (principalmente Boosting).*

Introdução

Ver, entender e interagir com o ambiente ao redor parece um conjunto de tarefas simples, mas que levaram milhões de anos de evolução biológica para serem desenvolvidas e aprimoradas. Por isso é impressionante constatar que a engenharia já cobriu parte deste terreno em apenas algumas décadas. Mesmo sem a mesma naturalidade dos seres humanos, sistemas autônomos desempenham algumas tarefas aparentemente rotineiras, mas que requerem compreensão e dinamismo mais sofisticados do que apenas a execução de funções pré-programadas.

Esta rápida progressão da robótica acompanha principalmente a miniaturização e o barateamento da microeletrônica, que praticamente seguiu a chamada Lei De Moore (Moore, 1965),(Kanellos, 2003) desde 1965, dobrando o número de componentes por área de encapsulamento de circuito integrado a cada ano. Isso gerou um infindável leque de inovações e revoluções tecnológicas, utilizando o "cérebro" eletrônico.

Um dos desafios mais interessantes e atuais propostos para a inteligência eletrônica é o de dirigir um carro. Guiar um veículo com segurança não é algo imediato nem para seres humanos, que precisam de tempo de instrução (auto-escolas), prática e adaptação para fazê-lo adequadamente. Dirigir exige uma mistura de conhecimentos prévios, como leis de trânsito, reconhecimento de objetos, noções de velocidade e distância; com decisões de momento, como quanto esterçar o volante ou definir um trajeto ótimo, sem colisões, entre dois pontos quaisquer do percurso, considerando a existência de outros veículos e obstáculos dinâmicos.

Processar todas essas informações simultaneamente, com computadores, é bastante complexo. Um sistema proposto (Kornhauser et al., 2007) no desafio urbano do departamento de defesa norte-americano (DARPA) para veículos autônomos sugere a utilização de um *rack* (estante) de servidores *dual core* (de dois núcleos por processador) montados em uma rede *gigabit* de baixa latência, e nem assim foi capaz de se equiparar a um motorista humano comum.

Entre as funções destes computadores está a de enxergar o ambiente através de câmeras. Isso é particularmente complicado pelo volume de informação a processar em uma dada janela de tempo. Enquanto um sensor analógico simples (por exemplo, um *encoder* de uma roda ou um acelerômetro) envia um sinal de poucas "variáveis" (ou seja, de poucas dimensões), uma imagem contém algo entre centenas de milhares a dezenas de milhões de pontos que são atualizados a cerca de 20 e 30 vezes por segundo.

Para lidar com toda essa informação em tempo real, é necessário utilizar programas e algoritmos altamente eficientes, fazendo uso inteligente de memória e processador. Assim, além de bom conhecimento de programação, é interessante partir de uma plataforma existente e robusta, desenvolvendo aplicações sobre uma base sólida ao invés de partir completamente do zero.

Existem *frameworks* (bases de desenvolvimento) disponíveis para a confecção de aplicações para a visão computacional. Entre eles há soluções comerciais, como módulos para o Matlab™ da MathWorks™ e para o LabView™ da National Instruments ™. Em contrapartida há alternativas de código aberto, mais interessantes do ponto de vista acadêmico do projeto, por possibilitarem o completo entendimento do funcionamento interno do *software*, que ficaria oculto em um programa comercial.

Provavelmente a biblioteca *open-source* (código aberto) mais robusta e popular é a OpenCV, distribuída sob a licença BSD. A OpenCV conta com uma ampla gama de funções, que vão desde interface com o usuário e com câmeras, até elementos de inteligência artificial,

além de toda a parte de processamento de imagem. A biblioteca não está vinculada a nenhum *hardware* específico, o que possibilita a escolha independente dos materiais a serem utilizados. Seu foco principal está exatamente nas aplicações *real-time* (em tempo real), o que a torna interessante para visão robótica.

Por outro lado, a visão computacional por si só não é suficiente para desempenhar as atividades necessárias para fazer funcionar um carro autônomo. Abstrair um sentido maior a partir de uma massa de dados é uma tarefa típica de inteligência artificial, e é em grande parte responsável pelo que o sistema "entende" do mundo que vê.

Selecionando características resultantes da análise feita pelo processamento de imagem, a inteligência artificial é responsável por fazer a síntese e tentar identificar algo para o qual ela foi treinada (resultado ou previsão). Se essa I.A. for treinada para, dada uma massa de dados, identificar se ela pertence a esta ou aquela categoria, a I.A. recebe o nome de classificador.

Esta combinação de inteligência artificial e processamento de imagem é uma ferramenta poderosa, que possibilita a sistemas autônomos "enxergar" ambientes convencionais, expandindo os horizontes da robótica autônoma visual. Através disso é possível levá-la a campo, atuando em ambientes reais ao invés de apenas situações-conceito.

Com isso em mente, este trabalho relata o desenvolvimento da inteligência central (visão computacional e inteligência artificial) de um veículo autônomo simples, cujo objetivo é percorrer pequenos trajetos sem interação humana, guiado apenas por câmeras. Ao contrario da maioria das iniciativas para o projeto de veículos autônomos, como (Reinholtz et al., 2007), (Kornhauser et al., 2007) e (Montemer et al., 2007), que utilizam *clusters* (conjunto de computadores) de pelo menos 2 servidores em *racks*, parte-se de uma configuração muito mais modesta, utilizando um computador *laptop* comum. Além disso, considera-se apenas a visão (câmeras), de forma semelhante a um motorista humano mas diferente das outras propostas que utilizam radares ou sensores de distância a laser.

Por isso, os objetivos também foram simplificados: ao invés de tentar prever as diversas situações possíveis no tráfego, a inteligência desenvolvida almeja apenas encontrar marcos visuais na pista, contornando-os em *slalom* (zigue-zague). Demais obstáculos são desconsiderados, dadas as limitações de *hardware* eletrônico e sensorial. Mas para manter uma funcionalidade interessante, elegeu-se cones de trânsito comuns como marcos de pista.

Por um lado, a opção pelos cones é interessante por serem elementos comuns em situações de trânsito (estacionamentos, desvios, zonas de restrição) e portanto amplamente disponíveis, sem depender de alteração do ambiente de testes ou atuação. Além disso, a cor laranja destes é intencionalmente distinguível do fundo. Por outro, sua forma e sua cor (uma ou duas, dependendo da presença de marcadores refletores) são os únicos parâmetros que o definem, o que dificulta o trabalho da visão computacional, pela ausência de pontos fortes de referência, como cantos (*corners*), arestas internas à silhueta do objeto e textura.

Além de localizar os cones com as câmeras, a inteligência central também é responsável por despachar os comandos para a movimentação do veículo, a serem captados e executados por uma inteligência periférica, microcontrolada e mais simples, que se responsabiliza por fazer a interface eletrônica com os circuitos da movimentação do veículo. Para a inteligência central, no entanto, esse processo deve ser completamente transparente, de forma que a periférica e os circuitos possam ser substituídos sem ser necessária alteração na central.

Materiais e métodos

Materiais

Graças às diversas funcionalidades da OpenCV, foi possível utilizá-la como principal biblioteca de *software*. Optou-se por utilizar a versão 2.2, mas preferindo as interfaces de programação na linguagem C quando possível, por simplicidade e maturidade, além da vantagem de aproveitar todo o trabalho feito em (Delai, 2010). Mesmo que a biblioteca esteja migrando todas as suas funções para C++, as interfaces em C serão mantidas para compatibilidade reversa.

Mas nem todas as estruturas da OpenCV têm interface em C. As inteligências artificias disponíveis são acessíveis apenas por classes C++, motivo pelo qual todo o projeto teria que ser compilado em C++. Assim, o projeto foi desenvolvido em C++, utilizando os recursos de classes da linguagem para as funções criadas, mas preferindo as interfaces C de bibliotecas externas.

Por facilidade de programação e espírito de *software* livre, utilizou-se o Debian GNU/Linux com o compilador GCC/G++ para compilar e executar os programas criados.

O *hardware* sobre o qual estes programas rodam é um Sony Vaio^{TM} VGN-NR310E, com um Intel Pentium^{TM} *Dual Core* a 1.73GHz e 1GB de memória RAM. Toda a comunicação com os dispositivos externos (câmera e inteligência periférica) é feita através das saídas USB do *notebook*.

As câmeras utilizadas são *webcams* USB Newlink WC301.

Os exemplos utilizados para treinar a inteligência artificial foram 100 imagens obtidas da internet, contendo ou não o tipo de cone procurado e classificadas manualmente.

Desenvolvimento

Na busca pelos cones, primeiramente selecionou-se o método de inteligência artificial a ser utilizado. A escolha evidente é um classificador, que apontará se uma região, previamente segmentada e selecionada, representa ou não um cone, dados os parâmetros (numéricos) da análise feita por processamento de imagem.

Tipicamente este tipo de procedimento (detecção de objeto) é implementado por um sistema do tipo Viola-Jones (Viola and Jones, 2004). No entanto, ele requer uma grande quantidade de dados para treinar o classificador adequadamente, já que ele treina uma inteligência artificial para encontrar os pontos de maior interesse no objeto que se deseja classificar, utilizando apenas características Haar. No trabalho original, foram utilizados 4916 exemplos positivos e 9544 exemplos negativos de rostos para treinar o classificador, todos filtrados e marcados manualmente. Além disso, os cones não apresentam textura ou características internas marcantes, apenas em sua relação com o fundo. Quando a característica mais marcante do objeto é sua silhueta, como no caso estudado, Viola-Jones sabidamente não se comporta bem (Bradski and Kaehler, 2008).

Sem a possibilidade de utilizar o Viola-Jones, partiu-se para a busca de inteligências artificiais desvinculadas das características utilizadas. Considerou-se primeiramente os métodos das Árvores Aleatórias e *Boosting*, cujas implementações na OpenCV funcionam de maneira mais imediata (Bradski and Kaehler, 2008).

O método das árvores aleatórias utiliza um grupo (floresta) de árvores de decisão, que "aprendem" os dados rigorosamente. Individualmente, estas árvores têm pouca utilidade por sofrerem de *overfitting* (sobreajustamento) já que criam modelos "interpolados" ao invés de "ajustados". Para compensar isso, o resultado final leva em conta o resultado de todas as árvores da floresta, que escolhem características aleatórias, dentre as fornecidas, para serem treinadas.

Boosting funciona compondo um conjunto de classificadores fracos (ou seja, que não precisam ser muito melhores que uma estimação aleatória) para gerar um classificador forte (Freund and Schapire, 1999). A OpenCV implementa o AdaBoost (*Adaptive Boosting*) binário (do tipo sim-não), que não é problema pois supre a necessidade de classificar uma região fornecida em cone ou não-cone. Estes classificadores fracos recebem pesos para seus votos na classificação final. Ao longo do algoritmo, tenta-se fazer com que os classificadores se concentrem nas características mais difíceis (com maior erro nas primeiras iterações).

Selecionadas então duas opções primárias de inteligência artificial, foi necessário escolher as características com as quais treinar esses classificadores.

Inicialmente pesquisou-se sobre dois métodos de descrição existentes, SURF (*Speeded-Up Robust Features*, Características Robustas Aceleradas) (Lowe, 2006) e SIFT (*Scale Invariant Feature Transform*, Transformação de características invariantes com a escala), (Bay et al., 2004). No entanto, ambos recaem no problema da ausência de *keypoints* (pontos chave) na estrutura dos cones de trânsito.

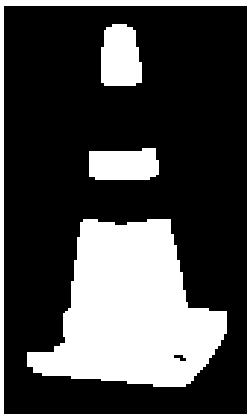
Por isso, decidiu-se criar um conjunto próprio de características que definem um cone, e deixar à inteligência artificial a incumbência de filtrá-las e selecionar as mais importantes. Para compor o modelo, utiliza-se estatísticas de cor (HSV), características do gradiente (módulo e orientação) e relações geométricas (momentos e relações entre momentos).

Todas as informações de cor para compor o modelo são da base HSV (*Hue Saturation Value*), que frequentemente é muito mais representativa em termos de visão computacional do que a base RGB (*Red*, *Green Blue*, Vermelho, Verde e Azul), padrão mais utilizado para a transmissão e exibição de imagens digitais (Russ, 1998).

Percebeu-se que o canal de saturação apresenta boa separação entre o cone, que tem alta saturação, e o fundo, geralmente uma via de trânsito, que mostra baixa saturação (como visto na Figura 1b). Por isso, além das estatísticas de cor HSV, as informações de geometria também são obtidas a partir de manipulações do canal de saturação. Em especial, cita-se a binarização automática pelo método de Otsu (Otsu, 1979), como forma de obter imagens binárias para o cálculo de momentos e relações geométricas.

(a) Cone

(b) Canal de saturação



(c) Máscara

Figura 1: Cores no cone

Além da imagem binarizada por Otsu, é usada também outra imagem binária, a máscara de cor, que é resultado da passagem de um filtro de cor HSV sobre a imagem analisada. Todos os pixels dentro da faixa de cores designada para o cone (laranja) serão 255 (branco) na máscara, e os demais, zero.

Assim, dada uma região retangular (redimensionada para 50x150) extraída de uma imagem, é extraído um vetor de características (descritores). Este vetor contém 89 valores numéricos que descrevem a região, com o objetivo de reconhecer um cone. A seguir são expostos os elementos obtidos para compor o vetor, na ordem em que são processados, e as equações para referência. As equações de descrição por momentos (4, 5 e 6) foram obtidas em (Kilian, 2001).

Para a imagem colorida:

- **1-20**: histograma normalizado de 10x2 dimensões dos valores de cor H (Hue) e S (Saturação) da região.
- **21**: limiar que melhor separa binariamente o canal de saturação, pelo método de Otsu.

Para o canal de saturação (S):

- **22-31**: histograma normalizado de 10 dimensões da intensidade do gradiente (equação 1).
- **32-41:** histograma normalizado de 10 dimensões da direção do gradiente (equação 2).

Para a imagem preto-e-branco separada pelo limiar de Otsu no canal de saturação:

- **42-49:** momentos: $\{m_{x,y} | x+y \leq 3, 0 \leq x < 3, 0 \leq y < 3\}$
- **50 e 51:** centro de gravidade (equação 3).
- **52**: orientação (equação 4).
- **53**: excentricidade (equação 5).
- **54**: afastamento (equação 6).
- **55-62**: valores normalizados das *Freeman Chain Codes* (códigos de cadeia de Freeman).

Para a máscara:

- **63-70:** momentos
- **71 e 72:** centro de gravidade (equação 3).
- **73**: orientação (equação 4).
- **74**: excentricidade (equação 5).
- **75**: afastamento (equação 6).
- **76-83**: valores normalizados das *Freeman Chain Codes* (códigos de cadeia de Freeman, Figura 2).

Para a imagem colorida:

- **84-86**: coeficientes de variação para as cores dos 3 canais HSV. (equação 7).

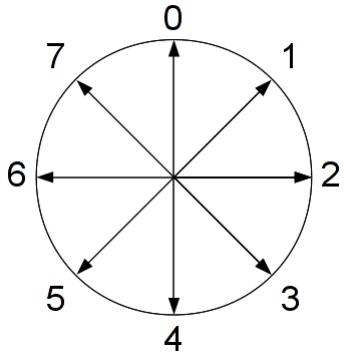


Figura 2: Índices da cadeia de Freeman

- **87-89**: desvio padrão para as cores dos 3 canais HSV.

$$\sqrt{\left(\frac{\delta I(x,y)}{\delta x}\right)^2 + \left(\frac{\delta I(x,y)}{\delta y}\right)^2} \tag{1}$$

$$arctan\left(\frac{\frac{\delta I(x,y)}{\delta y}}{\frac{\delta I(x,y)}{\delta x}}\right) \tag{2}$$

$$\left(\frac{m_{1,0}}{m_{0,0}}, \frac{m_{0,1}}{m_{0,0}}\right) \tag{3}$$

$$\frac{1}{2} arctan\left(\frac{2\mu_{1,1}}{\mu_{2,0} - \mu_{0,2}}\right) \tag{4}$$

$$\frac{(\mu_{2,0} - \mu_{0,2})^2 - 4(\mu_{1,1})^2}{(\mu_{2,0} + \mu_{0,2})^2} \tag{5}$$

$$\mu_{2,0} - \mu_{0,2} \tag{6}$$

$$\frac{\sigma_x}{\bar{x}} \tag{7}$$

Todas as derivadas parciais são calculadas com um núcleo Sobel 3x3 (Scharr). $\mu_{n,m}$ se refere a um momento central de ordem n em x em y. Todas as normalizações são feitas para gerar distribuições (soma dos valores dos histogramas igual a 1). As cadeias de Freeman relacionam todos os pontos de um contorno com seu vizinho, assumindo que este seja composto de linhas finas (apenas 1 pixel de espessura), dando índices para esta relação, conforme a Figura 2. Antes de calcular os coeficientes de Freeman, são computadas as linhas finas que geram os contornos das imagens binárias, que são as regiões de fronteira entre claro e escuro. Por valor normalizado desta cadeia, entende-se a proporção do aparecimento dos índices na criação da cadeia.

Definido o modelo, criou-se um programa que permite selecionar regiões retangulares em imagens maiores, classificando-as manualmente como cone ou não-cone. Assim é possível ligar um conjunto de características a uma resposta, que serão utilizados para treinar o classificador.

Para a massa de dados, foi obtida uma centena de imagens da Internet, contendo ou não cones de trânsito. Destas foram selecionadas 68 imagens, das quais foram manualmente separadas 1200 regiões, sendo 200 regiões com cones completamente visíveis (como na Figura 1a) e as outras 1000 sem nenhum. As demais imagens foram utilizadas como validação, e não foram apresentadas à inteligência artificial para treinamento. Destas últimas extraiu-se 30 elementos positivos (cones) e 100 negativos. A quantidade de exemplos positivos e negativos foi artificialmente dobrada utilizando-se os espelhos horizontais das regiões fornecidas, semelhante ao feito em (Viola and Jones, 2004). No entanto, não se espera que o desempenho dobre, já que

algumas características (cor, por exemplo) se mantém, mas isso ajuda a evitar que o algoritmo de Inteligência Artificial (A.I.) se vicie em um determinado ângulo de visão mais comum nos dados (*biasing*).

As duas soluções de inteligencia artificial foram treinadas e testadas contra o conjunto de validação. Os pesos de classificação foram ajustados para obter no máximo 10% de falso-positivos.

Percebeu-se que, para os dados disponíveis, *Boosting* se comportou melhor, conforme as tabelas 1 e 2, e por isso foi o método escolhido.

		Previsto	
		Positivo	Negativo
Real	Positivo	62.2%	37.8%
	Negativo	5.4%	94.6%

Tabela 1: Matriz de confusão para as Árvores Aleatórias

		Previsto	
		Positivo	Negativo
Real	Positivo	70.3%	29.7%
	Negativo	0%	100%

Tabela 2: Matriz de confusão para o *Boosting*

Com um algoritmo de inteligência artificial capaz de classificar regiões retangulares de tamanho uniforme, é necessário criar um segmentador que busque estas regiões em uma imagem maior, ou seja, todo o campo de visão da câmera.

Isso é feito buscando-se "manchas" claras, de tamanho razoável, na máscara de cores, quando aplicada a toda a imagem. Deve-se aplicar algum grau de erosão na máscara para eliminar regiões muito pequenas, mas o tamanho mínimo depende do tamanho total da imagem, resolução e etc. Para cada uma destas manchas, considera-se seu tamanho máximo horizontal (w) e vertical (h). Para o classificador são passados 3 retângulos, de largura w e alturas h, 2h e 3h, alinhados com a base da mancha. Isso se deve ao fato de muitos cones de trânsito apresentarem uma ou duas faixas refletoras ao longo do comprimento, o que quebra as regiões que passam na mascara de cores buscando o laranja.

Antes de terem suas características extraídas, os retângulos são recortados de modo que suas retas limites (superior, inferior, direita e esquerda) tangenciem os limites da(s) mancha(s). Por exemplo, a reta que limita o retângulo a direita será posicionada sobre ponto mais a direita da mancha (ou conjunto de manchas). Isso é feito para tentar fazer com que os limites do retângulo sejam também os limites do próprio cone.

Localizados os retângulos aceitos como cones pela I.A., filtra-se esse resultado fundindo retângulos que se interceptem.

Uma vez localizados os objetos de interesse e definido o sistema de navegação (que não foi possível de ser feito, como descrito no tópico seguinte), a inteligência central despacha comandos para a periférica através de uma saída serial (ou serial sobre USB). Esta comunicação contém os comandos básicos para o deslocamento do veículo, em especial comandos para o motor de tração (ex. mova-se X metros adiante, ou para trás) e para o controle de direção (ex. vire X graus para a direita, ou esquerda).

Apenas para efeito de protocolo, utilizou-se comandos de uma uma única letra, seguida de um número e uma quebra de linha (\n). O número quantifica o comando, e pode vir precedido de sinal. Por exemplo, 'm-100' significa "mova-se 100cm para trás". A quebra de linha indica o fim do comando. Comandos não sucedidos por números também são aceitos (ex. 'p' para "pare"), mas todos devem conter a quebra de linha ao final.

Resultados e discussão

As Figuras 3 a 6 mostram o comportamento do sistema em algumas situações que não foram usadas para treinamento da A.I.. É importante notar que, embora o índice de falso-positivos tenha sido zero na bateria de validação, ainda podem existir falso-positivos, como na Figura 6.

Mesmo assim, os índices de acerto descritos na tabela 2 são bastante razoáveis para a aplicação. Com mais dados para o modelo, a tendência seria de melhora. Entre a aplicação real (visão do veículo em situações reais) e a simulada (imagens da Internet) foi necessário apenas ajustar os valores da máscara, que são fortemente dependentes da distorção cromática da câmera.

Adicionar mais exemplos à inteligência artificial pode melhorar seu desempenho (Ng, 2008). No entanto, o problema mais crítico ainda é o da navegação.

Figura 3: Exemplo

Figura 4: Exemplo

Figura 5: Exemplo

Figura 6: Exemplo com erros

Sobre o segmentador, sabe-se que este falhará se o cone tiver cor semelhante ao fundo no qual ele se situa, de modo que o ambiente também faça parte da máscara. Isso não é grande problema em asfalto, mas poderia ser prejudicial no caso de algo laranja aparecer em contato com a estrutura do cone na imagem. A utilização de informação de contexto poderia também melhorar a qualidade (ex. cones não fariam sentido se não apoiados no chão), mas isso implicaria em um programa muito mais complicado.

Para efeito de navegação, utilizou-se a área aparente do cone encontrado como referência de distância, mas este parâmetro se mostrou bastante fraco para fazer a estimativa. Idealmente, se utilizaria um sistema de visão estéreo para calcular esta distância, mas percebeu-se que as *webcams* comuns não mantém nenhuma relação de parâmetros (foco, centro ótico, orientação, distorção de cor e outros) entre uma câmera e outra, mesmo entre produtos do mesmo lote. Para a câmera utilizada, mesmo para modelos com apenas 4 números de fabricação de distância, não foi possível construir um conjunto estéreo, tamanha a disparidade entre seus parâmetros.

Das Figuras 7 e 8 pode-se notar, visualmente, que os parâmetros das câmeras utilizadas são bastante diferentes entre si. A imagem foi captada com as duas câmeras alinhadas sobre um eixo horizontal comum. Percebe-se que foco, orientação e até cor são diferentes entre as câmeras, o que dificulta a configuração de visão estéreo.

Por causa disso, não foi possível implementar um sistema de navegação eficiente. Considerou-se técnicas como *chase/evade* (perseguição e desvio) utilizando funções potenciais (de Carvalho Junior, 2007), mas devido às dificuldades de se obter uma boa estimativa da distância ao objeto e a restrições mecânicas no veículo, não se chegou a uma solução adequada em uma situação real. No entanto, em simulações feitas com um *software* desenvolvido, que permite posicionar cones arbitrariamente e testar a ideia da navegação, com todas as distâncias conhecidas, a perseguição e desvio se mostrou simples de implementar e adequada para o veículo.

Assim, uma proposta interessante seria a utilização de sensores de distância, como um sonar, mas interagindo com as câmeras, com o objeto sendo localizado por visão e a distância medida pelo outro sensor. Alternativamente, pode-se seguir a mesma linha deste projeto, mas utilizando câmeras adequadas para a visão estéreo, com relação de parâmetros garantida.

Figura 7: Visão da câmera da direita

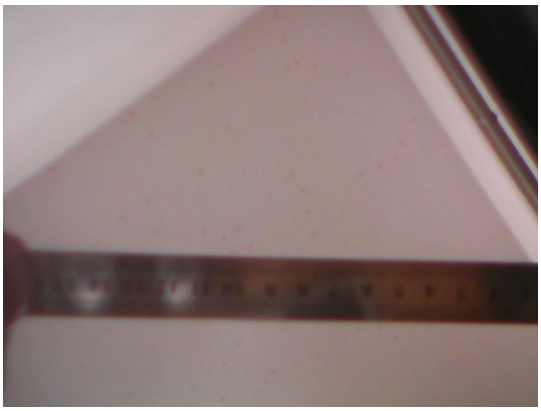


Figura 8: Visão da câmera da esquerda

Conclusões

Os resultados alcançados ainda estão distantes das capacidades dos veículos apresentados em (Kornhauser et al., 2007), (Reinholtz et al., 2007) e (Montemer et al., 2007). No entanto, estes apresentam uma proposta mista (sensores e visão), enquanto este trabalho expõe uma abordagem exclusivamente através da visão.

Através das técnicas citadas, foi possível encontrar os cones com uma boa precisão (como mostrado na Tabela 2). No entanto, não foi possível chegar a um sistema de navegação funcional.

Tanto os trabalhos citados quanto este próprio indicam que a proposta de criar um piloto artificial para um veículo ainda é uma ideia bastante ousada. No entanto, as recentes investidas tanto de empresas do segmento de tecnologia e automotivo, além de acadêmicos e militares apontam para um crescimento da área. Com isso, não seria surpreendente esperar o aparecimento de soluções comerciais, completa ou parcialmente autônomas, nas próximas décadas.

Referências Bibliográficas

Bay, H.; Ess, A.; Tuytelaars, T., e Gool, L. V. (2004). *Distinctive Image Features from Scale-Invariant Keypoints*. International Journal of Computer Vision.

Bradski, G. e Kaehler, A. (2008). *Learning OpenCV*. O'Reilly Media, 1 edition.

Carvalho Junior, H. H. de (2007). *Métodos Inteligentes de Navegação e Desvio de Obstáculos*. Universidade Federal de Itajubá, Programa de Pós-Graduação em Engenharia Elétrica.

Delai, R. L. (2010). *Visão Computacional com a OPENCV – Material Apostilado e Veículo Seguidor Autônomo*. 2º Seminário Mauá de Iniciação Científica.

Freund, Y. e Schapire, R. E. (1999). *A Short Introduction to Boosting*. Journal of Japanese Society for Artificial Intelligence, 14(5).

Kanellos, M. (2003). *Moore's Law to roll on for another decade*. CNET News, http://news.cnet.com/2100-1001-984051.html (acesso em 11/11).

Kilian, J. (2001). *Simple Image Analysis By Moments*. http://public.cranfield.ac.uk/c5354/teaching/dip/opencv/SimpleImageAnalysisbyMoments.pdf (acesso em 11/11).

Kornhauser, A. L.; Atreya, A.; Cattle, B.; Momen, S.; Collins, B.; Downey, A.; Franken, G.; Glass, J.; Glass, Z.; Herbach, J.; Saxe, A.; Ashwash, I.; Baldassano, C.; Hu, W.; Javed, U.; Mayer, J.; Benjamin, D.; Gorman, L., e Yu, D. (2007). *DARPA Urban Challenge Princeton University Technical Paper*. DARPA Urban Challange 2007.

Ku, C.-H. e Tsai, W.-H. (2001). *Obstacle Avoidance in Person Following for Vision-Based Autonomous Land Vehicle Guidance Using Vehicle Location Estimation and Quadratic Pattern Classifier*. IEEE Transactions on Industrial Electronics, 10(1).

Lowe, D. G. (2006). *Speeded-Up Robust Features (SURF)*. European Conference on Computer Vision.

Montemer, M.; Becker, J.; Bhat, S.; Dahlkamp, H.; Dolgov, D.; Ettinger, S.; Haehnel, D.; Hilden, T.; Hoffmann, G.; Huhnke, B.; Johnston, D.; Klumpp, S.; Langer, D.; Levandowski, A.; Levinson, J.; Marcil, J., e Orenstein, D. (2007). *Junior: The Stanford Entry in the Urban Challenge*. DARPA Urban Challange 2007.

Moore, G. E. (1965). *Cramming more components onto integrated circuits*. Electronics Magazine, 38(8).

Ng, A. (2008). *Advice for applying Machine Learning*. Stanford University, http://cs229.stanford.edu/materials/ML-advice.pdf, acesso em 11/11.

Otsu, N. (1979). *A Threshold Selection Method from Gray-Level Histograms*. IEEE Transactions on Systems, Man, and Cybernetics, 9(1).

Reinholtz, C.; Alberi, T.; Anderson, D.; Bacha, A.; Bauman, C.; Cacciola, S.; Currier, P.; Dalton, A.; Farmer, J.; Faruque, R.; Fleming, M.; Frash, S.; Gothing, G.; Hurdus, J.; Kimmel, S.; Sharkey, C.; Taylor, A.; Terwelp, C.; Covern, D. V.; Webster, M., e Wicks, A. (2007). *DARPA Urban Challenge Technical Paper*. DARPA Urban Challange 2007.

Russ, J. C. (1998). *The image processing handbook*.

Viola, P. e Jones, M. (2004). *Rapid Object Detection Using a Boosted Cascade of Simple Features*. Computer Vision and Pattern Recognition.