UNIVERSIDADE DE SÃO PAULO
ESCOLA DE ENGENHARIA DE SÃO CARLOS

FERNANDO ISAMO PELLIM
NUMASHIRI

AVALIAÇÃO DA TÉCNICA DE ENTROPIA DA MISTURA LOCAL APLICADA A COMANDOS VISUAIS

São Carlos
2010

FERNANDO ISAMO PELLIM NUMASHIRI

AVALIAÇÃO DA TÉCNICA DE ENTROPIA DA MISTURA LOCAL APLICADA A COMANDOS VISUAIS

Trabalho de Conclusão de Curso apresentado à Escola de Engenharia de São Carlos, da Universidade de São Paulo.

Curso de Engenharia Elétrica com ênfase em Eletrônica.

Orientador:
Prof. Dr. Adilson Gonzaga

São Carlos

2010

Ficha catalográfica preparada pela Seção de Tratamento
da Informação do Serviço de Biblioteca – EESC/USP

Numashiri, Fernando Isamo Pellin
N971a Avaliação técnica de entropia da mistura local aplicada a comandos visuais / Fernando Isamo Pellim Numasshiri ; orientador Adilson Gonzaga. -- São Carlos, 2011.

Monografia (Graduação em Engenharia Elétrica com ênfase em Eletrônica) -- Escola de Engenharia de São Carlos da Universidade de São Paulo, 2011.

1. Visão computacional. 2. Contornos. 3. Análise multiescala. 4. Detecção de pontos dominantes. 5. Entropia. 6. OpenCV. I. Título.

FOLHA DE APROVAÇÃO

Nome: Fernando Isamo Pellim Numashiri

Título: "Sistema de Visão Computacional para Controle de um Robô"

Trabalho de Conclusão de Curso defendido e aprovado em 11 / 02 / 2011,

com NOTA 7,5 (sete, meio), **pela comissão julgadora:**

Prof. Dr. Evandro Luís Linhari Rodrigues - EESC/USP

Prof. Dr. Maximiliam Luppe - EESC/USP

Prof. Associado Homero Schiabel
Coordenador da CoC-Engenharia Elétrica
EESC/USP

Dedico este trabalho à minha família, pela paciência, suporte, carinho e dedicação que tiveram por mim até hoje..

AGRADECIMENTOS

Quero agradecer a todos os professores pelo conhecimento que me ajudaram a adquirir, em especial aos professores Evandro Luís Linhari Rodrigues e Adilson Gonzaga, por mostrarem o quão interessante e desafiador são os sistemas digitais e embarcados desenvolvidos atualmente.

Por fim, quero agradecer à Universidade de São Paulo, pelo ótimo apoio e oportunidades oferecidas durante meu período de graduação.

RESUMO

NUMASHIRI, I. P. F. **AVALIAÇÃO DA TÉCNICA DE ENTROPIA DA MISTURA LOCAL APLICADA A COMANDOS VISUAIS.** 2010. 63 p. Tese (Graduação) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010.

Neste projeto é proposto um sistema de visão computacional para identificação de comandos visuais. O sistema de visão é baseado num detector de pontos dominantes chamado Multiscale Local Mixing Entropy (LME). Para agilizar o desenvolvimento do software de processamento de imagens, utilizou-se uma biblioteca chamada Open Source Computer Vision (OpenCV), que oferece um framework estável e permite a implementação de algoritmos de processamento fazendo uso eficiente dos recursos da máquina. O sistema foi configurado para detectar setas e as direções para o qual elas apontam, visando futuras aplicações de controle remoto de movimentação de robôs. Testes foram realizados para verificar a eficácia e os limtes do sistema na detecção de comandos visuais.

Palavras-chave: Visão Computacional. Contornos. Análise Multiescala. Detecção de Pontos dominantes. Entropia. OpenCV

ABSTRACT

NUMASHIRI, I. P. F. **EVALUATION OF LOCAL MIXING ENTROPY TECHNIQUE APPLIED TO VISUAL COMANDS.** 2010. 63 p. Thesis (Undergraduate) - Escola de Engenharia de São Carlos, Universidade de São Paulo, São Carlos, 2010.

This project proposes a computer vision system to identify visual commands. The system is based on a dominat point dectetor named Multiscale Local Mixing Entropy (LME). To speed up the software development cycle of the system, the software was developed using a library called Open Source Computer Vision (OpenCV), which provides a stable framework and allows fast implementation of processing algorithms, making efficient use of machine resources. The system was setup to identify arrows and the direction which they point to, aimming at future applications of remote-handling robots. Tests were conducted to evaluate the effectiveness and limits of the system in detectin visual commands.

Keywords: Computer Vision. Contours. Local Multiscale Mixing Entropy. Dominant Point Detection. Entropy. OpenCV

LISTA DE FIGURAS

LISTA DE TABELAS

LISTA DE SIGLAS

CDT	C/C++ *Development Toolkit*
IDE	*Integrated Development Environment*
GCC	*GNU Compiler Collection*
LME	*Local Multiscale Entropy*
OpenCV	*Open Source Computer Vision*

SUMÁRIO

Capítulo 1

INTRODUÇÃO

Com o desenvolvimento da tecnologia da informação em nossa sociedade, será cada vez mais comum encontrar sistemas computacionais embarcados em nosso ambiente. Esses ambientes irão impor necessidades para novos tipos de interação entre homem e máquina, com interfaces naturais e simples de utilizar. Em particular, a habilidade de interagir com equipamentos computadorizados sem a necessidade de equipamentos externos é atrativa.

Hoje, o teclado, o mouse e o controle remoto são utilizados como os principais meios para transferir informações e comandos para um sistema computadorizado. Em algumas aplicações envolvendo informações tridimensionais, tais como visualização, jogos computadorizados e controle de robôs, outras interfaces baseadas em *trackballs*, *joysticks* e *datagloves* são empregadas. Em nossa vida diária, no entanto, utilizamos nossa visão e audição como as principais fontes de informação sobre o ambiente ao redor. Portanto, pode-se indagar até que ponto seria possível desenvolver equipamentos computadorizados capazes de comunicar com humanos de uma maneira similar, entendendo comandos visuais e auditivos.

O propósito desse projeto é desenvolver uma técnica de detecção de "pontos dominantes" para o reconhecimento de direções visando a aplicação em robótica móvel.

1.1 Comandos visuais

Comandos visuais são uma das maneiras mais naturais e rápidas de comunicação entre humanos. A sociedade moderna se beneficia de comandos visuais, seja para instruir motoristas quando devem parar ou onde não é possível virar, indicar se é possível ou não fumar em um determinado ambiente, ou até mesmo instruir uma pessoa a manusear com segurança componentes tóxicos ou frágeis.

1.1.1 Gestos

Gestos manuais fornecem um método único e poderoso de controlar equipamentos computadorizados. Gestos manuais são comumente utilizados para transmitir significado de uma pessoa para outra. Essa fonte de controle natural pode ser o próximo passo

na introdução de sistemas de controle computadorizados de maneira simples e intuitiva. Iniciativas nessa direção incluem sistemas para controle de mouse com gestos das mãos (BRETZNER; LAPTEV; LINDEBERG, 2002), controle de dispositivos eletrônicos através de gestos manuais, indicando opções em menus em forma de pizza (LENMAN; BRETZNER; THURESSON, 2002)

1.1.2 Contorno

A visão computacional inclui muitas técnicas como processamento de imagens, reconhecimento e classificação de padrões, modelagem geométrica e processamento cognitivo. As aplicações possíveis utilizando visão computacional são tantas quanto as tarefas humanas que dependem da visão para serem executadas, as quais podem ser aperfeiçoadas com a inclusão de maior precisão, robustez à fadiga e a capacidade de operar em ambientes inóspitos como, por exemplo, um robô que explore as profundezas do mar, as entranhas de um vulcão ou o solo do planeta Marte utilizando a visão como o centro do seu sistema de navegação.

Juntamente com a cor e a textura, a forma é uma das características básicas em visão computacional. A análise de formas desempenha um papel crucial em: reconhecimento de objetos (AMIT, 2002; MERCIMEK; GULEZ; MUMCU, 2005), reconstrução tridimensional (LAMECKER; WENCKEBACH; HEGE, 2006; SRINIVASAN; LIANG; HACKWOOD, 1990), inspeção de peças industrializadas (MALAMAS et al., 2003; THOMAS et al., 1995), detecção e reconhecimento de faces (LI; JAIN; NETLIBRARY, 2005; FASEL; LUETTIN, 2003), reconhecimento de caracteres (TRIER; JAIN; TAXT, 1996), inspeção e seleção de alimentos (BROSNAN; SUN, 2002), detecção de frutos em árvores (JIMENEZ; CERES; PONS,), análise de movimentos humanos (MOESLUND; HILTON; KRUGER, 2006), recuperação de imagens por conteúdo (SAYKOL; GUDUKBAY; ULUSOY, 2005).

Talvez, devido à onipresença das formas em nossas vidas e a facilidade com que o sistema visual às processa, as definições ao seu respeito são qualitativas e de alto nível, que em geral fazem uso de um vocabulário de formas básicas para compor definições de formas mais complexas. Para o computador, tais definições estão além de sua capacidade, sendo assim, necessitam de uma definição de baixo nível e quantitativa. Não existe uma definição geral e satisfatória do conceito de formas. O conceito apresentado em (DRYDEN; MARDIA, 1998) é o mais comumente usado em visão computacional e em análise estatística de formas, o qual diz o seguinte: Forma é a informação geométrica que permanece, quando os efeitos de posição, de escala e de rotação são excluídos do objeto.

Em todas as aplicações onde é necessário reconhecer ou classificar formas, surge o problema de como representá-las para que possam ser comparadas e classificadas. A representação e a descrição de formas, em geral buscam por meios efetivos de captura a essência de suas características, tal que facilitem o armazenamento, a transmissão, a

comparação e o reconhecimento. Tais características precisam estar em harmonia com a definição de forma, isto é, precisam ser independentes de translação, rotação e escala (tamanho da forma).

Muitas técnicas de representação e descrição de formas tem sido propostas. Em Pavlidis (1982), uma introdução a taxonomia foi feita para organizar as formas, cujas classes mais fundamentais são a representação interna e a representação externa, que atualmente são mais referidas como representação baseada em região e representação baseada em contornos (ZHANG; LU, 2004). Esta última utiliza informações sobre os pontos para representar a forma em oposição à anterior que utiliza informações do objeto como um todo. Há ainda, outras classificações e detalhes sobre técnicas de representação, as quais são discutidas em (ZHANG; LU, 2004; FONTOURA; JR, 2001; LONCARIC, 1998; MARSHALL, 1989).

1.1.3 Pontos dominantes

O trabalho desenvolvido em Attneave (1954) mostrou que as partes menos redundantes são aquelas com maior influência na percepção da forma. O contorno da imagem de um objeto é a região menos redundante, porém, neste ainda existem partes redundantes, que são aquelas onde a direção do contorno permanece constante ou que varia suavemente com inclinação constante, que é o caso das retas e dos arcos de circunferência. Isto quer dizer que nem todos os pontos de um contorno possuem a mesma importância na caracterização de formas. Nos pontos onde a variação da direção é mais acentuada, se concentra a informação sobre a a forma do objeto.

Uma vez detectados esses pontos, que podem receber o nome de pontos dominantes, pontos críticos, singularidades, pontos salientes, vértices ou *corners*, pode-se aproximar a forma original do contorno conectando-os com segmentos de retas ou arcos, sem que a percepção geral da forma seja deteriorada. A segmentação do contorno com uma subsequente aproximação poligonal é uma técnica de simplificação de formas, que facilita muito as tarefas de reconhecimento e classificação. A facilitação se deve a produção de uma forma contendo apenas os seus detalhes mais importantes sem os ruídos e os detalhes irrelevantes do contorno.

1.2 Aplicações

1.2.1 Robôs móveis

Robôs móveis tem a capacidade de se deslocar no ambiente em que se encontram e não estão fixos em um local fixo. Robôs móveis estão no centro de muitas pesquisas atuais. Dentre os vários tipos de robôs móveis, estão desenvolvendo cada vez mais robôs guiados de modo autônomo. Esses tipos de robôs possuem algum tipo de informação

do ambiente ao seu redor e de como chegar em diferentes lugares para realizar tarefas. Nesses tipos de robô, a visão computacional está tendo papel cada vez maior na captura e processamento de informações, reunindo informações que ajudam a construir mapas de navegação, manipulação de objetos e detecção de obstáculos, possibilitando aos robôs executarem tarefas consideradas simples por humanos, como dobrar toalhas (MAITIN-SHEPARD et al., 2010), abrir e fechar portas (CHITTA; COHEN; LIKHACHEV, 2010), remover neve e cortar grama (CHOSET, 2001).

1.3 Objetivos

O objetivo principal desta monografia é avaliar a técnica de detecção de "pontos dominantes" para verificar o desempenho e a viabilidade de integrar os conhecimentos adquiridos aqui com aplicação em robótica móvel.

1.4 Organização do trabalho

Esta monografia é constituída de quatro capítulos de forma a apresentar a teoria necessária para descrever o sistema proposto e apresentar seus resultados. São eles:

- **Capítulo 2** - Neste capítulo apresenta uma visão geral da análise de formas, em específico as técnica utilizada no detector de pontos LME.
- **Capítulo 3** - O capítulo descreve os materiais utilizados na realização dos experimentos e uma descrição em passos do funcionamento completo do sistema de visão computacional desenvolvido, bem como a metodologia empregada na análise do desempenho do sistema.
- **Capítulo 4** - São apresentados os resultados obtidos bem como possíveis trabalhos futuros para aprimorar o sistema. Por último, segue a conclusão do trabalho.
- **Anexos A até G** - Nos anexos de A até G encontram-se todos os arquivos contendo o código criado para a execução do trabalho.

Capítulo 2

ANÁLISE DE CONTORNOS

2.1 Introdução

Este capítulo discutirá conceitos de detecção de pontos dominantes, o funcionamento interno do detector LME, a técnica desenvolvida para calcular o ângulo formado pela seta e o eixo horizontal da imagem.

2.2 Noções Fundamentais

As características extraídas estão no contorno de imagens binárias, porém não são extraídas diretamente do contorno, ou melhor, da sequência de pontos que representa um contorno digital, mas sim do relacionamento entre os pontos do objeto e os pontos do fundo, do qual se extrai o próprio contorno. Este relacionamento informa como é a fronteira entre o objeto e o fundo, se ela é uma fronteira em linha reta, se o fundo se projeta no interior do objeto formando uma concavidade ou se o objeto se projeta no fundo formando uma convexidade. Cada um dos pontos do contorno gerado por esse relacionamento possui um valor que quantifica essas configurações. Uma aplicação direta desses valores é a representação de formas através dos pontos de maior relevância do contorno, que no jargão da visão computacional são chamados de pontos dominantes.

2.3 Detecção de pontos dominantes

Um ponto dominante surge numa curva digital onde dois segmentos de reta (ou aproximadamente retos) adjacentes se encontram, isto é, onde existem duas bordas dominantes em direções diferentes numa vizinhança local. Um ponto dominante é um marco onde a natureza da curva se modifica significativamente. Pontos dominantes representam características importantes de um objeto e desempenham um papel importante na percepção de formas por humanos (ASADA; BRADY, 2009). Em Guru, Dinesh e Nagabhushan (2004) é declarado que a informação sobre a forma está concentrada nos pontos dominantes, os quais provaram ser primitivas descritivas em representação de formas e interpretação de imagens.

A detecção de pontos dominantes ou *corners* é um operação de baixo nível, cuja saída pode alimentar aplicações de alto nível em visão computacional e reconhecimento

de objetos. Alguns exemplares são: casamento de imagens (SMITH et al., 1998; VINCENT; LAGANIÈRE, 2005), decomposição de curvas digitais (ABE et al., 2002; MARJI; KLETTE; SIY, 2005), aproximação poligonal ou segmentação linear por partes (SARFRAZ; ASIM; MASOOD, 2004; CESAR; COSTA, 1995). Os pontos dominantes servem para simplificar a análise de imagens reduzindo a quantidade de dados a serem processados e ao mesmo tempo preservando as informações importantes sobre o objeto (LIU; SRINATH, 1990).

Uma categoria especial de algoritmos de detecção de *corners* aplica o conceito de representação espaço escala para lidar com ruídos e detalhes em diferentes escalas. O espaço escala de curvaturas é construído através da convolução de um contorno (1D) com uma família de funções Gaussianas (1D), cujo desvio padrão crescente representa o crescimento da escala. Os ruídos e os detalhes do contorno, com tamanho inferior ao da abertura da Gaussiana usada na filtragem, são totalmente dissolvidos ao longo dos pixels de sua vizinhança, deixando o contorno mais suave. A curvatura é computada em cada nível do espaço escala e através do exame dos cruzamentos por zero, as concavidades e convexidades do contorno são encontradas. O resultado é um conjunto diferente de *corners* para cada nível da escala. Exemplos de esquemas de espaço escala de curvatura são encontrados em (RATTARANGSI; CHIN, 2002; PEI; LIN, 1992).

2.4 Abordagem multi-escala

Em uma cena os objetos existem em diferentes escalas. Uma maneira comumente encontrada na literatura para se referir à hierarquia de escalas existentes numa imagem é fazer alusão à cena de uma floresta, na qual se tem a imagem global da floresta, a imagem individual de uma árvore e a imagem contendo um zoom das características da árvore, como galhos, folhas e as nervuras das folhas. Cada uma dessas entidades existe em sua própria escala e o sistema visual humano é capaz de se "deslocar" facilmente entre elas, dedicando sua atenção à característica que mais lhe interessar. De maneira semelhante, os contornos possuem protuberâncias e reentrâncias de vários tamanhos, algumas são importantes para a forma geral e outras podem ser descartadas sem afetar o seu reconhecimento, tarefa que é facilmente realizada pelo sistema visual humano, mas para implementá-la artificialmente em sistemas computacionais é necessário conferir-lhes um processamento multi-escala.

Ao se observar uma cena, usa-se um conjunto de fotorreceptores com determinada abertura. Se houver o interesse em observar pequenos detalhes, as aberturas precisarão ser estreitas, significando que uma menor quantidade da cena será registrada. Se o interesse for a observação de detalhes mais gerais, será necessário utilizar aberturas maiores, registrando uma parte maior da cena. Em geral não se sabe, antecipadamente, qual é o tamanho apropriado da abertura, assim, o parâmetro de observação de escala precisa ser livre, capaz de lidar com todas as escalas simultaneamente. Este conceito em utili-

zar aberturas com vários tamanhos diferentes para observar faixas de escalas diferentes é chamado de medição de dados multi-escala.

A idéia básica das representações multi-escala é criar uma pilha de sinais a partir do sinal original, cujo nível cresce de acordo com o crescimento da escala, isto é, em cada nível da pilha existirá uma versão cada vez mais simplificada do sinal original, pois seus detalhes são gradativamente suprimidos com o aumento da escala. Isto significa que um sinal em escala baixa possui detalhes finos, ao contrário de um sinal em escala alta que possui apenas os detalhes mais gerais.

Davis em (DAVIS, 2006) foi um dos primeiros a apresentar a idéia multi-escala na detecção de pontos dominantes. Em seu trabalho, associou a detecção de lados e de ângulos e construiu aproximações do contorno original com diferentes níveis de precisão, gerando uma hierarquia de ângulos e lados que descreviam o contorno com qualquer nível de detalhe.

2.5 LME

Este tópico detalha o funcionamento interno do detector de pontos dominantes baseado em análise multi-escala, desenvolvido em (LOURO; MACHADO; GONZAGA, 2009). Nele serão introduzidos os conceitos de difusão e entropia, utilizados pelo LME para quantificar a difusão em cada ponto do contorno. O desempenho do detector é comparado com mais seis outros detectores descritos em (ROSENFELD; JOHNSTON, 2009; ROSENFELD; WESZKA, 2006; FREEMAN; DAVIS, 2006; BEUS; TIU, 1987; CHETVERIKOV, 2003; SARFRAZ; MASOOD; ASIM, 2006).

2.5.1 Distribuição de pesos

As distribuições de pesos utilizadas no detector funcionam como uma receita, especificando a quantidade de fundo e de objeto que devem ser misturadas. A exigência do algoritmo de que a entropia deve ser inversamente proporcional ao ângulo do ponto dominante leva modificação da distribuição gaussiana.

As equações abaixo transformam a distribuição Gaussiana G, gerando a distribuição G'. É necessário encontrar um valor Δ tal que, ao adicioná-lo ao valor central da distribuição, resulte em um valor igual a 0.5 após a renormalização. O valor do peso central da Gaussiana G é representado por W_c, e o peso temporário é representado por W_t; o valor final é obtido após o passo de normalização realizado na equação 2.3, onde a somatória representa a soma de todos os pesos. A distribuição resultante mantém o valor do peso central inalterado, enquanto os outros pesos são distribuídos aos outros pixels quando a

escala cresce.

$$\Delta = 1 - 2W_c \tag{2.1}$$

$$W_i = W_c + \Delta \tag{2.2}$$

$$G' = \frac{G}{\sum w_{ij}} \tag{2.3}$$

2.5.2 Difusão

A difusão é o processo pelo qual a matéria é transportada de uma parte de um sistema para outra como resultado dos movimentos moleculares aleatórios (CRANK, 1979). A dependência no tempo da distribuição estatística no espaço é dada pela equação 2.4,

$$\partial_t u = div(D \cdot \nabla u) \tag{2.4}$$

onde u é a matéria e D é o coeficiente de difusão, que representa um conjunto de propriedades da matéria que facilitam ou dificultam a difusão e definem se esta é isotrópica ou anisotrópica. O termo ∇u representa a concentração de matéria, ou gradiente, que é uma função vetorial obtida a partir de um campo escalar (imagem). O divergente de um gradiente é equivalente ao ao Laplaciano do campo escalar do qual se originou o gradiente. O Laplaciano pode ser considerado uma generalização da derivada segunda para dimensões mais altas, servindo como uma ferramenta para caracterizar a concavidade de uma função . Do ponto de vista do processamento de imagens, o Laplaciano mede o relacionamento de um pixel com sua vizinhança. Considerando que a difusão é proporcional ao gradiente, espera-se, então que a difusão em imagens seja mais acentuada na interface entre as regiões homogêneas, isto é, nas bordas. Sendo assim, caso seja possível medir a difusão, pode-se criar um detector de bordas. O processo de filtragem por uma Gaussiana causa um efeito de misturar regiões diferentes de uma imagem, enquanto regiões semelhantes permanecem inalteradas, ou seja, em regiões de brilho constante, não se percebe mudança. O processo de mistura proposto utilizando distribuições gaussianas constitui um estágio inicial de difusão. Considerando-se as idéias de Attneave apresentadas em , (LOURO; MACHADO; GONZAGA, 2009) conclui à conclusão de que em um contorno, detectado em decorrência de uma difusão parcial, existem pontos onde a difusão é mais acentuada do que em outros.

2.5.3 Entropia

A entropia é uma grandeza que relaciona a quantidade de estados possíveis de um sistema dinâmico. No contexto de visão computacional, a entropia pode ser utilizada para medir a perda de informação sobre alguma coisa, ou a incerteza sobre alguma informação. O contorno de um objeto é um local onde há incerteza sobre a posição; pode-se estar sobre o objeto ou sobre o fundo. No contorno objeto pode-se encontrar as extremidades de um

objeto, e essas extremides são os locais com maior incerteza ou menor redundância.

$$H = -\sum_{k=1}^{N} p(a_k) \times \log_2 (p(a_k)) \tag{2.5}$$

Na equação 2.5 p_i representa a quantidade de partículas do tipo i divididas pelo total de partículas na mistura, p_i é obtida diretamente a partir da distribuição de pesos rotacionalmente simétrica.

2.5.4 Operação do LME

Este tópico descreve em maiores detalhes a operação do método LME. O método pode ser dividido nas seguintes etapas:

- Determinação do contorno da imagem binária, onde cada ponto possui um valor de entropia;
- Exclusão dos pontos do contorno cujos valores de entropia não correspondam a convexidades ou concavidades, mantendo-se apenas aqueles cujos valores são mais relevantes, os candidatos a pontos dominantes;
- Reavaliação da importância dos candidatos, calculando-se suas entropias em escalas maiores e excluindo-se aqueles que perderam a importância durante esse processo;
- Comparação da importância de cada candidato restante, analisando-se a evolução de sua importância ao longe das escalas, utilizando-se idéias de velocidade e aceleração para caracterizar a variação dos valores de entropia no eixo das escalas.

2.5.5 Características das imagens de entrada

O método atua diretamente sobre imagens binárias, com características específicas. São elas:

- O valor dos pixels do objeto deve ser igual a 0 e os pixels correspondentes ao fundo devem ser iguais a 255;
- O tipo de dados da imagem deve ser real;
- Caso o objeto esteja muito próximo das bordas da imagem, ela deverá sofrer "padding", adicionando 18 pixels em cada borda, que é a matade da maior janela Gaussiana utilizada no algoritmo.

2.5.6 Cálculo das distribuições

Nesse estágio do algoritmo, é calculada a distribuição da janela 3x3. Inicialmente é calculado um filtro Gaussiano 2D, com desvio padrão $\sigma = 0.5$. Em seguida as equações 2.1, 2.2 2.3 são usadas para modificar a distribuição.

2.5.7 Aplicando o detector na imagem

Nesta fase a janela 3x3 percorre a imagem. Em cada pixel, o algoritmo calcula os pesos dos objeto e do fundo englobados pela janela. Os pesos de cada pixel representam a quantidade de substância dispersa na mistura. O parâmetro p_i na equação 2.5 é substituído por esses pesos. Uma representação esquemática desse processo é ilustrada na figura 2.1, que exibe a detecção de um vértice do retângulo; nessa escala (3x3) a entropia para esse tipo de corner é sempre igual a "0.8341″.

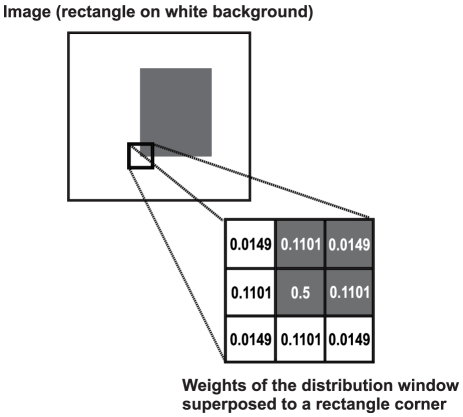


Figura 2.1: Nesse exemplo o peso total do objeto dentro da janela móvel é $p_0 = 0.7351$, e o peso do fundo é $p_1 = 0.2649$. Aplicando os valores na equação 2.5, obtêm-se o valor da entropia no ponto.

A janela deslizante w delimita a vizinhança de interesse através da imagem binária, deslocando-se coluna por coluna e linha por linha de modo similar a rasterização. Considere que a imagem possui dimensões $r \times s$. Na equação 2.6, o índice l varia no intervalo $[2, r-1]$, e o índice c varia no intervalo $[2, s-1]$.

$$W = \begin{bmatrix} a_{l-1,c-1} & a_{l-1,c} & a_{l-1,c+1} \\ a_{l,c-1} & a_{l,c} & a_{l,c+1} \\ a_{l+1,c-1} & a_{l+1,c} & a_{l+1,c+1} \end{bmatrix} \tag{2.6}$$

Para cada janela deslizante delimitada por w, nós computamos os pesos para os pixels pretos e brancos. Primeiro, encontramos as coordenadas de cada pixel pois os pesos estão associados com posição espacial, e então calculamos o peso total para cada tipo de pixel. Na equação 2.7, H_3 é a distribuição proposta para uma janela que representa todos os pesos para essa escala. A saída da operação lógica "AND" é a matriz w_1, que contém

uma combinação de zeros e uns. A equação 2.8 pode identificar a saída da equação 2.7. Se $w_0 = 0$ ou $w_1 = 9$ significa que a janela w se encontra em uma região homogênea (interna ao objeto ou no fundo, respectivamente), e a entropia pode ser levada a zero. Se w_0 possuir um valor entre "0" e "9", significa que a janela w está na fronteira ente o objeto e o fundo, e as equações 2.9, 2.10 e 2.11 determinam os pesos de cada região dentro da janela w.

$$w_1 = and(w, H_3) \tag{2.7}$$
$$w_0 = \sum\sum w_1 \tag{2.8}$$
$$w_2 = H_3 \cdot w_1 \tag{2.9}$$
$$w_{BG} = \sum\sum w_2 \tag{2.10}$$
$$w_{Obj} = 1 - w_{BG} \tag{2.11}$$

Na equação 2.9, a matriz w_1 representa um valor igual a "1" para os pixels de fundo, e um valor igual a "0" para os pixels do objeto. A multiplicação de H_3 resulta na matriz w_2, que representa somente os pesos do fundo. A somatória da equação 2.10 resulta no peso total do fundo dentro da janela w (ou w_2). Como a distribuição H_3 é normalizada, aplicamos a equação 2.11 para calcular o peso total do objeto.

A dispersão medida através da equação 2.5 utilizando os pesos w_{BG} e w_{Obj} como as probabilidades. A imagem resultante do algoritmo é uma imagem de entropia, que mostra valores de baixa intensidade para pontos distantes da borda, alta intensidade para pontos nos vértices das convexidades, e intensidades médias para pontos em segmentos de retas e concavidades. A imagem de entropia apresenta alguns valores não desejados de entropia, e para removê-los, utilizamos um *threshold* duplo, filtrando valores abaixo da configuração mais côncava, e filtrando valores acima da configuração mais convexa. Os valores das faixas de entropias removidas são:

- Entropia < 0.5 indica pixels internos (do objeto ou do fundo)
- Entropia > 0.955 indica pixels em uma protusão muito fina, ou um pixel solitário cercado de fundo (Entropia = 1).

Seguindo a operação de *thresholding*, a imagem de entropia precisa ser limpa outra vez, pois apresenta uma fronteira dupla. O processo de difusão ocorre em ambas as direções gerando uma mistura das partículas do objeto no fundo e vice-versa. Para remover a segunda fronteira, uma subtração de imagem é realizada: "imagem de entropia" - "imagem de entrada".

Para entender o significado das intensidades de entropias, foi analisado 512 configurações em janelas de 3x3 compostas de pixels pretos e brancos. Foram encontradas oito

configurações mais comuns relativas a um contorno nessa pequena escala. Os ângulos são ilustrados na figura 2.2. As primeira linha mostra as quatro possíveis convexidades, a segunda linha mostra os segmentos de reta, e a terceira linha mostra as duas concavidades possíveis.

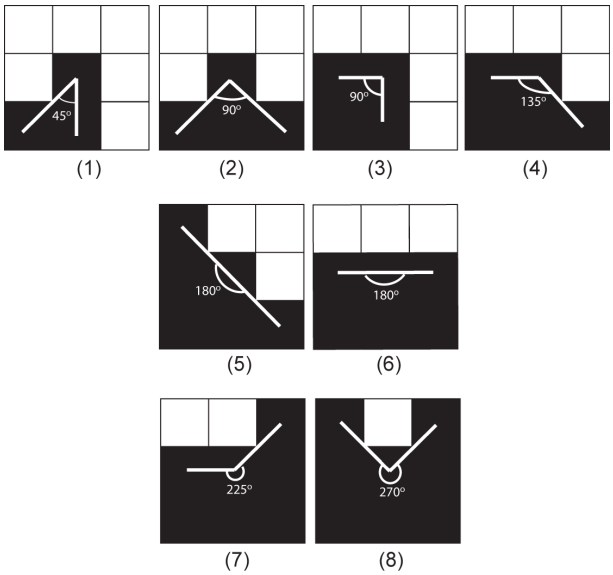


Figura 2.2: As oito configurações de contorno distinguidas pelo detector LME

2.5.8 O método multi-escala

A grande influência de ruído nas baixas escala requer a utilização de um método multi-escala. A solução adotada no detector LME é analisar as escalas maiores somente para candidatos já detectados pela escala inferior e checar o seu comportamento nas escalas superiores. Os pontos ruidoso devem perder sua proeminência com o crescimento da escala. Baseado nessa hipótese, as mesmas configurações representadas na figura 2.2 são reproduzidas nas janelas de tamanho 5x5, 9x9, 17x17, 25x25, 35x35, e as suas entropias foram calculadas utilizando distribuições de mesmo tamanho. A tabela 2.1 mostra os valores de entropia, onde as colunas representam as escalas, e os números dentro dos parênteses se referem aos detalhes de borda mostrados na figura 2.2. Cada um dos candidatos encontrados em uma escala inferior tem sua entropia recalculada em escalas superiores. A entropia do candidato em uma dada escala é comparada, na mesma escala, com as oito entropias da tabela 2.1 e é classificado em uma configuração que representa o valor mais próximo. Isto é feito da seguinte maneira:

- Acessar a imagem da silhueta através das coordenadas do candidato;
- Obter a vizinhança de tamanho desejado em torno das coordenadas;
- Calcular a entropia daquele fragmento;

- Comparar o valor com os valores da escala da tabela 2.1;

Tabela 2.1: Valores entrópicos para as oito configurações de borda em seis janelas

Detalhe	3x3	5x5	9x9	17x17	25x25	35x35
Convexa (1)	0.95443	0.9576	0.96815	0.97803	0.9811	0.98326
Convexa (2)	0.94276	0.92374	0.92792	0.93879	0.94288	0.94595
Convexa (3)	0.83405	0.87514	0.90695	0.93002	0.93694	0.94184
Convexa (4)	0.81127	0.81827	0.84194	0.86607	0.87415	0.88006
Diagonal (5)	0.78679	0.74882	0.75701	0.77870	0.78704	0.79336
Horizontal (6)	0.58397	0.65780	0.71666	0.76115	0.77495	0.78491
Concava (7)	0.54356	0.55595	0.59803	0.64134	0.65599	0.66676
Concava (8)	0.50221	0.43271	0.44733	0.48584	0.50066	0.51185

Janelas maiores possuem maior resolução angular; portanto mais configurações válidas são possíveis. Devido à grande dificuldade em analisar todas as configurações de preto e branco em janelas superiores a 3x3, as novas configurações são classificadas de acordo com um dos possíveis valores da tabela 2.1.

O resultado dessa fase é representado na tabela 2.2, que exibe uma estrutura que grava as coordenadas de cada candidato e seus valores de entropia computados em seis janelas distintas.

Tabela 2.2: Soma dos falsos positivos e falsos negativos

Coordenadas		Entropias calculadas por janela					
Linha	Coluna	3x3	5x5	9x9	17x17	25x25	35x35
...	...	0.83405	0.8556	0.8975	0.9105	0.9088	0.9175

2.5.9 Classificação das novas entropias

Para ajustar os valores de entropia calculados nas janelas maiores, foi desenvolvido um mecanismo de acomodação dos valores de entropia. O funcionamento do mecanismo é definido por um fator de tolerância, computado de maneira independente para convexidades e concavidades. A tolerância é baseada na distância entre pontos de configurações adjacentes de uma mesma coluna da tabela 2.1. Uma tolerância de 100% significa que a entropia calculada será aceita em uma configuração mais poderosa, mesmo se estiver a meio caminho entre uma configuração forte e uma mais fraca. A tolerância padrão é de 0%. Aumentar a tolerância implica na aceitação de mais candidatos a pontos dominantes, ou a aceitação de um corner menos poderoso como candidato. O fator de tolerância é aplicado a todas as entradas da tabela 2.1, com exceção da coluna 3x3, antes de fazer a classificação. As equações 2.12 e 2.13 alteram a convexidades e concavidades da tabela 2.1, respectivamente, onde S_{iT} é a nova entrada da tabela para a configuração i, S_i é a

configuração considerada, e S_{i+1} é a próxima configuração de cima para baixo na mesma coluna da tabela, e T é a tolerância em %.

$$S_{iT} = S_i - (\frac{S_i - S_{i+1}}{2}) \cdot T \tag{2.12}$$

$$S_{iT} = S_i + (\frac{S_{i-1} - S_i}{2}) \cdot T \tag{2.13}$$

A classificação da entropia modificada pela tolerância passa a ser a computação da distância euclidiana mínima entre a entropia e os pontos da tabela.

2.5.10 Filtragem de candidatos

Passando pela saída da fase de classificação, o fluxo de informação flui fluindo de uma escala para outra é mais facilmente percebido. Por exemplo, é mais fácil perceber uma convexidade vista através de uma pequena janela tornar-se uma concavidade quando observada através de uma grande janela. A figura 2.3 ilustra esse comportamento quando a escala aumenta.

O parâmetro de tolerância tem influência diretamente essa fase; uma tolerância maior implica menos transições de uma configuração mais forte para uma mais fraca, resultando em um candidato mais estável através das escalas.

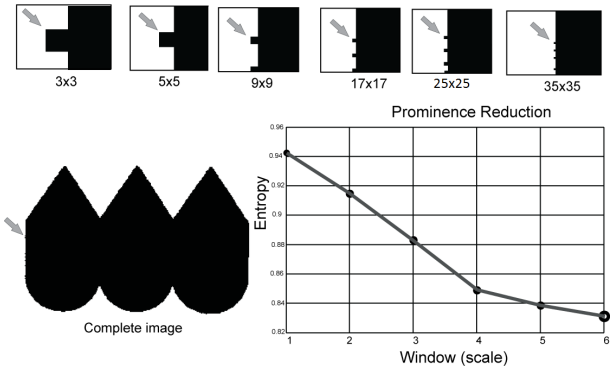


Figura 2.3: Redução de proeminência: a convexidade torna-se um segmento de reta

Os critérios usados para filtrar candidatos sofrem pequenas variações para concavidades e convexidades:

- Para convexidades, são selecionados os candidatos classificados na configuração 4 ou inferior em todas as janelas, cujo fluxo pelas escalas apresenta no máximo uma redução, ou nenhuma. A primeira janela não é considerada na análise de redução de proeminência.

- Para concavidades, os candidatos deve possuir uma configuração igual ou superior a 7, e somente uma redução de em sua proeminência.

2.5.11 Elegendo os pontos dominantes

Existem situações em que uma região possui vários candidatos a pontos dominantes. Nesse tópico descrevemos uma forma de selecionar qual candidato é mais importante em uma determinada região. Apesar de ser possível comparar diretamente os valores de entropia de cada ponto, devemos criar um método que leve em conta a evolução do ponto pelas escalas.

A evolução dos dados pode ser considerada um problema de movimento em mecânica clássica, onde as mudanças de velocidade e aceleração dos valores de entropia ao longo das escalas, representa a variação de detalhe. em Outras palavras, é possível estimar quão rápido um detalhe irá desaparecer quando a escala muda. Nós consideramos o valor de entropia como um deslocamento. O deslocamento que ocorre durante um intervalo de tempo unitário é a velocidade cuja magnitude é a mesma do valor da entropia. A velocidade média v_m é computada tirando-se a média da soma das entropias sobre as seis janelas, conforme descrito na equação 2.14.A aceleração média é obtida através da diferença da entropia da maior janela para a menor janela, dividido por seis unidades temporais, mostrados na equação 2.15. Como tanto a entropia média, quando a taxa de variação da entropia são informações importantes, utiliza-se como fator de comparação o produto da velocidade pela aceleração, ilustrado pela equação 2.16

$$v_m = \frac{\sum_{i=1}^{6} s_i}{6} \tag{2.14}$$

$$a_m = \frac{s_6 - s_1}{6} \tag{2.15}$$

$$F_{comp} = v_m \cdot a_m \tag{2.16}$$

Ao comparar dois candidatos, é necessário especificar o tamanho da região de disputa. Em um contorno, o tamanho dessa região é a distância entre pontos, que depende do tamanho, ou da escala do objeto. O problema da distância é então resolvido pela criação de um parâmetro a ser especificado pelo usuário.

2.5.12 Comparação do desempenho do detector LME

Os resultados das tabelas 2.3 e 2.4 são reproduzidos de (LOURO; MACHADO; GONZAGA, 2009) para justificar a escolha do detector desse projeto. A tabelas 2.3 mostra a detecção de falsos positivos e falsos negativos do detector LME e vários outros detectores clássicos encontrados na literatura. Percebe-se que o LME possui um desempenho superior em relação a todos os outros detectores competidores, e essa foi a razão principal da escolha da utilização do LME nesse projeto.

Tabela 2.3: Comparação de detecções por imagem de ground truth

Detector	IM1		IM2		IM3		IM4		IM5		IM6		IM7		IM8	
Imagem	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN	FP	FN
LME	0	0	0	0	0	0	0	0	0	1	1	2	3	0	1	2
S-A-M06	0	0	0	0	0	0	0	0	0	1	0	3	5	0	0	2
CS99	0	0	0	0	0	0	7	0	4	0	4	0	8	0	0	9
BT87	2	0	0	0	0	0	1	2	3	1	0	8	3	5	1	13
FD77	2	0	0	0	5	0	3	2	6	1	2	12	2	4	0	21
FD75	3	0	2	0	5	0	13	1	6	1	2	7	6	0	0	16
RJ73	4	0	2	0	5	0	12	2	5	1	4	11	5	0	0	16

Tabela 2.4: Soma dos falsos positivos e falsos negativos

Algoritmo	Total FP	Total FN	Total
LME	5	5	10
S-A-M 2006	5	6	11
CS99	23	9	32
BT87	10	29	39
FD77	20	40	60
FD75	37	25	62
RJ73	37	30	67

2.6 Orientação da seta

A técnica empregada para calcular o ângulo que a ponta seta faz com o eixo horizontal da imagem é ilustrada na figura 2.4.

Considerando um sistema de coordenadas cartesiano imaginário, alinhado com os eixos horizontais e verticais da imagem, calcula-se o ponto PN, ponto médio entre $P1$ e $P2$. A seguir calcula-se o coeficiente angular da reta que passa por PN e $P3$.

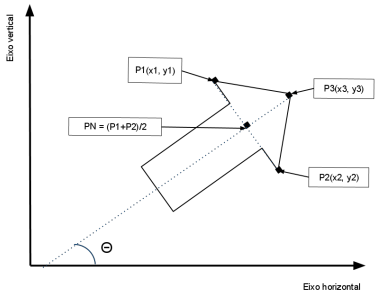


Figura 2.4: Representação do sistema de referências utilizado para calcular o ângulo de inclinação da seta em relação ao plano da câmera

$$P_N = \frac{P_1 + P_2}{2} \tag{2.17}$$

$$\Theta = \arctan \frac{y_3 - y_n}{x_3 - x_n} \tag{2.18}$$

O cálculo do ângulo Θ é realizado pela função $arctan2$, localizada na biblioteca $< math >$.

```
float theta = std::arctan2(y3 - yn, x3 - xn);
```

a função acima identifica o quadrante ao qual o ângulo pertence pela combinação dos sinais dos argumentos. Os ângulos retornados pela função se encontram na faixa $[-\pi, \pi]$.

Capítulo 3

MATERIAL E MÉTODOS

Para realizar os experimentos e desenvolver o *software* do trabalho, foi utilizado o *notebook* com *webcam* embutida, e sistema operacional Linux.

3.1 Preparação do ambiente de programação

As ferramentas utilizadas durante o desenvolvimento do projeto foram escolhidas primeiramente pelo fato de estarem disponíveis gratuitamente na internet, e pela facilidade de combinar e configurá-las para gerar um ambiente de desenvolvimento completo. Foi utilizada a IDE *Eclipse* em conjunto com o *plugin CDT* para habilitar o desenvolvimento de código em C++. Para compilar e linkar o projeto, foi utilizada a coleção de compiladores *GCC*. Optou-se por desenvolver as rotinas de processamento utilizando o *framework* da biblioteca OpenCV por causa das várias vantagens fornecidas:

- Rotinas para localização, configuração e manipulação de câmeras;
- Rotinas para abrir imagens e vídeos dos mais variados formatos;
- Facilidade para executar rotinas de processamento em paralelo. É possível controlar o número de núcleos utilizados, e determinar a quantidade de rotinas executando paralelamente;
- Utilização eficiente dos recursos do computador. Implementação eficiente e correta de algoritmos para conversão de escala de cores, conversão de imagens, e acesso à vários algoritmos clássicos de processamento de imagens.

3.2 Metodologia

O sistema de visão computacional construído foi dividido em três partes. A primeira parte consiste de rotinas de pré-processamento das imagens provenientes da câmera. A segunda parte consiste na extração de características das imagens preprocessadas, e a última parte consiste na comparação e classificação das características extraídas, decidindo se o objeto analisado é similar ao objeto procurado.

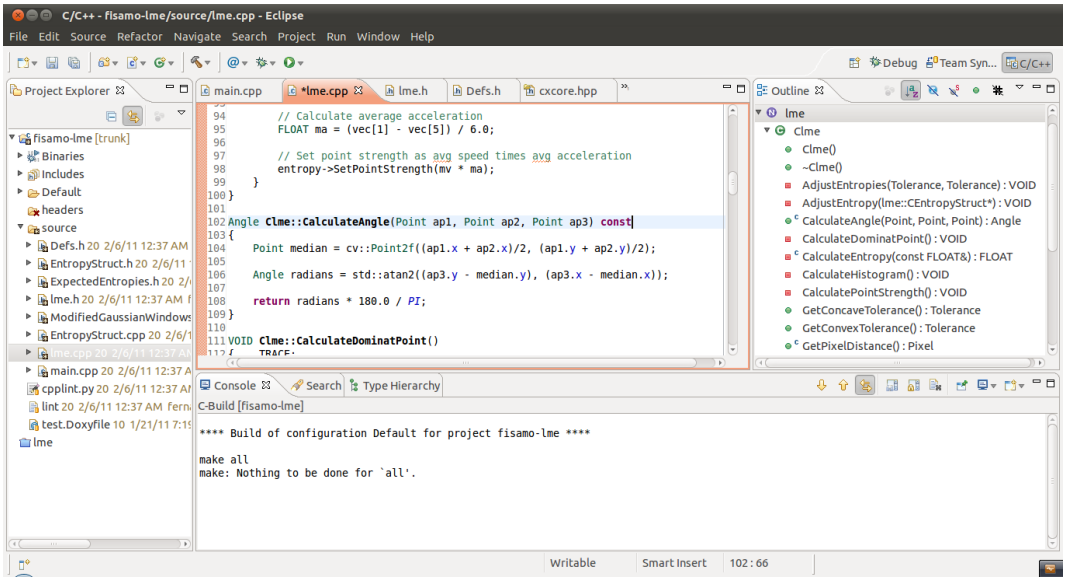


Figura 3.1: Ambiente de programação, exibindo exemplo de utilização do código

3.3 Pré processamento

Nessa etapa são detalhados as operações utilizadas para obter uma representação alternativa dos dados contidos na imagem. Os objetos que temos interesse em detectar podem ser diferenciados do restante dos elementos que compõem a imagem através de seu formato específico. De acordo com BRADSKI (2008), o pré-processamento de imagens consiste na aplicação de operadores com o objetivo de atingir resultados cujo significado é definido no contexto de imagens gráficas, visuais. As etapas do pré-processamento podem ser divididas nos seguintes passos:

- Conversão para escala de cinza;
- Remoção de ruído;
- Conversão para imagem binária;
- Localização de objetos.

3.3.1 Conversão para escala de cinza

Como o interesse nas imagens recebidas está na forma e não nas cores dos objetos que serão reconhecidos, as imagens recebidas são convertidas em escala de cinza, através da função cvCvtColor():

A função cvcvtColor converte um espaço de cores (número de canais) em outro enquanto que supõe que o tipo de dado seja o mesmo. O tipo exato de conversão é especificado pelo argumento code.

Todos os espaços de cores utilizam a seguinte convenção: imagens de 8-bits estão na

faixa 0-255, imagens de 16-bits estão na faixa 0-65536, e números em ponto flutuante 0-1.0. Quando imagens em escala de cinza são convertidas para imagens coloridas, todos os componentes da imagem resultante são tomados como iguais; no entanto para conversões reversas (RGB ou BGR para escala de cinza), o valor em escala de cinza é computado usando a fórmula

$$Y = (0.299)R + (0.587)G + (0.114)B \tag{3.19}$$

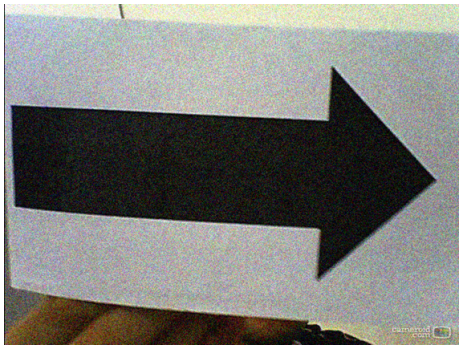


Figura 3.2: Imagem capturada da câmera, exibindo ruído do tipo salt and pepper

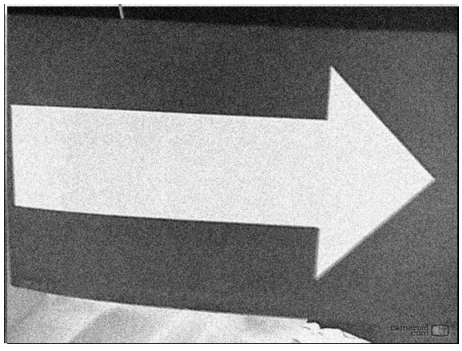


Figura 3.3: Conversão da imagem para escala de níveis de cinza, com ruído ainda presente

3.3.2 Remoção de ruído

A imagem resultante após da etapa anterior possui ruído do tipo *salt and pepper*. Para remover esse ruído, pode-se aplicar filtros lineares na imagem, sendo que os mais comuns são os filtros de média e de mediana. O filtro de mediana foi escolhido devido à natureza do ruído presente na imagem, pois os níveis de intensidade do ruído não eram extremos, e a imagem resultante possui as bordas nítidas como na imagem original. Nesse filtro cada um dos valores dos pixels de saída é determinado pelo mediana do conjunto de pixels vizinhos presentes numa janela de lado 5.

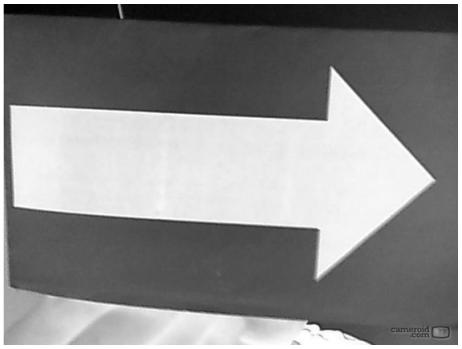


Figura 3.4: Ruído removido, aplicando filtro de mediana

3.3.3 Conversão para imagem binária

Converter uma imagem em escala de cinza para escala monocromática é uma tarefa comum de processamento de imagens. Nesse projeto em particular, imagens binárias são um dos pré-requisitos do detector LME. Em (OTSU, 1975), um método de binarização de imagens é descrito, chamado de método de Otsu, que foi aplicado na imagem 3.4. O método foi escolhido por calcular automaticamente os valores de *threshold* para cada pixel, possibilitando a identificação de símbolos independente da cor dos mesmos.

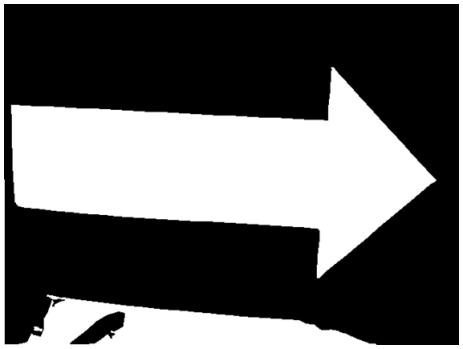


Figura 3.5: Conversão da imagem para escala binária, exibindo dois objetos distintos

3.3.4 Localização de objetos

A imagem em escala binária da figura 3.5 pode ser utilizada como entrada do detector LME. No entanto, o detector irá encontrar os pontos dominantes de todos os objetos, do fundo, tornando complexa a tarefa de associar cada ponto dominante ao seu respectivo objeto. A solução encontrada foi subdividir a imagem em pedaços menores, onde cada pedaço contém um único objeto. A biblioteca OpenCV possui duas rotinas que facilitam a implementação dessa tarefa.

A primeira rotina, cv::findContours(...), calcula os pontos que fazem parte dos contornos de cada objeto da imagem. A segunda rotina, cv::boundingRect, calcula as coordenadas do retângulo delimitador do conjunto de pontos. Dessa maneira foi possível dividir a imagem em sub-imagens, e alimentar o detector LME com uma sub-imagem por vez.

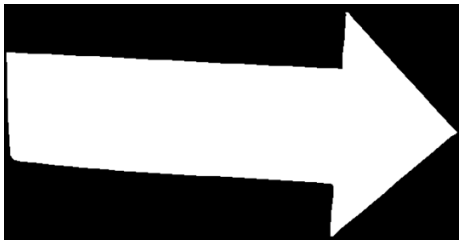


Figura 3.6: Primeiro objeto segmentado

Figura 3.7: Segundo objeto segmentado

3.4 Algoritmo LME

Aplicando o algoritmo LME na figura 3.6, resulta na figura 3.8, onde estão pintados os pontos dominantes encontrados pelo algoritmo.

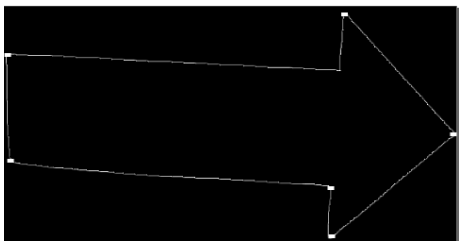


Figura 3.8: Pontos dominantes calculados pelo LME destacados na figura

Os objetos a serem identificados consistem de setas impressas em folhas de papel, afixadas em superfícies planas e perpendiculares ao eixo principal da câmera, indicando possíveis direções de navegação.

As imagens são acessadas da câmera a partir do código exibido no anexo H, das linhas 11 até as linhas 29. A câmera transmite quadros codificados em escala RBG, com 8 bits por canal. Essa imagem é então convertida para uma escala de cinza, segundo a equação abaixo

3.5 Pós Processamento

Após obter os pontos dominantes de um objeto, torna-se necessário comparar de algum modo os pontos dominantes obtidos de uma imagem de referência. A solução adotada foi converter a lista de pontos dominantes, com suas respectivas entropias, em um histograma. O histograma é dividido em oito faixas de tal maneira que cada uma das oito configurações de entropia da janela 3x3 fique dentro de uma das faixas. Para comparar a semelhança entre dois histogramas, foi utilizada uma métrica de distâncias, detalhada em (BHATTACHARYYA, 1943; THACKER; AHERNE; ROCKETT, 1997).

$$d_{Bhattachryya}(H_1,\ H_2) \quad = \quad \sqrt{1 - \sum_i \frac{\sqrt{H_1(i) \cdot H_2(i)}}{\sqrt{\sum_i H_1(i) \cdot \sum_i H_2(i)}}} \tag{3.20}$$

A equação 3.20 descreve a fórmula da distância entre dois histogramas. Valores próximos de zero indicam alta semelhança entre os histogramas, enquanto que valores próximos de 1 indicam baixa semelhança. Histogramas perfeitamente idênticos possuem distância 0, enquanto que histogramas completamente distintos tem distância 1.

Durante os experimentos realizados, verificou-se que uma distância inferior a 0.2 foi suficiente para considerar como idênticos dois histogramas distintos de uma seta. Um histograma típico de uma seta é representado na figura 3.9

Figura 3.9: Histograma dos pontos dominantes da figura

Capítulo 4

RESULTADOS E CONCLUSÕES

4.1 Resultados

Nesse capítulo serão discutidos os testes realizados para verificar o desempenho do sistema proposto.

4.1.1 Verificação de falsos positivos

Para verificar a incidência da detecção de falsos positivos, o sistema foi alimentado com uma base de dados de aproximadamente 1400 imagens, obtida em (LATECKI; LAKAMPER; ECKHARDT, 2010), compostas de contornos de objetos, plantas e animais. Cada contorno foi preenchido, e sofreu uma transformação aleatória, composta de uma rotação sobre o eixo perpendicular ao plano da imagem, ou de um escalonamento. Os resultados obtidos encontram-se na tabela 4.1

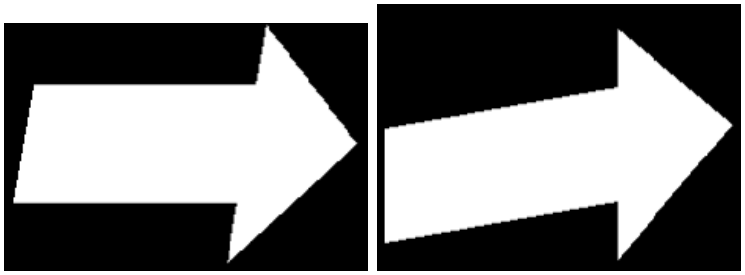
Tabela 4.1: Falsos positivos

Algoritmo	Falso Positivo	Negativo	Total
LME	18	1394	1404

Inicialmente o teste foi executado sem a aplicação de transformações aleatórias, mas visto que a quantidade de falsos positivos era muito próxima de zero, decidiu-se modificar a base de imagens para obter maiores informações à respeito do desempenho do sistema. A quantidade de falsos positivos aumentou consideravelmente quando o fator de escala diminuiu muito as dimensões dos objetos. Quando as dimensões se aproximaram do tamanho da janela mínima detectável, de aproximadamente 32 por 24 pixels, a quantidade e a entropia dos pontos dominantes detectados no objeto se alterou, objetos com formatos diferentes de setas apresentaram em alguns casos configurações de pontos dominantes similares a das setas.

4.1.2 Verificação de falsos negativos

A verificação de incidência de falsos negativos foi um pouco mais complexa, pois não foi encontrada uma base de dados de setas disponível livremente na web. O sistema foi alimentado com uma base de dados construída a partir de imagens obtidas da internet,



(a) *Skew* horizontal (b) *Skew* vertical

Figura 4.1: Transformações afins

selecionadas para construir uma base de imagens contendo setas modificadas aleatoriamente por transformações do tipo *skew*, conforme ilustrado nas figuras 4.1a e 4.1b. As imagens em que as setas possuem cores mais claras que o fundo (Ex: seta branca sobre fundo verde), quando transformadas geraram um objeto de cor 0 e fundo 1. Como o detector LME exige justamente o contrário, essas imagens tiveram seus valores de pixel invertidos, permitindo ao detector operar de maneira correta.

Tabela 4.2: Falsos negativos

Algoritmo	Falso Negativo	Positivo	Total
LME	20	122	142

Os resultados obtidos da tabela 4.2 mostram que o sistema possui uma taxa de acertos próxima de 86%. Um das dificuldades encontradas foi ajustar o valor do parâmetro de distância entre pixels dos pontos dominantes. Um valor pequeno de distância favorece a detecção de objetos pequenos e deteriorava a dos grandes. Um valor grande de distância atuava de maneira inversa.

4.1.3 Tamanhos mínimos e máximos de objetos

O tamanho máximo e mínimos para objetos é reproduzido em escala 1:4 na figura 4.2. O tamanho dos objetos reconhecidos pelo sistema está limitado inicialmente pela câmera utilizada. Como as imagens que a câmera transmite possuem resolução de 640x480 pixels, o maior objeto detectável deve estar contido dentro de um retângulo de 640x480 pixels. A inclusão de um tamanho mínimo de objeto detectável foi necessária devido à presença de grande quantidade de objetos pequenos, que representam ruído, sendo gerados na etapa de pré-processamento. Todos os objetos pequenos eram processados pelo detector LME, o que tornava a execução do sistema bastante lenta. A escolha do tamanho da janela foi baseada em ensaios experimentais, onde foi feita uma comparação simples do tempo de processamento de cada imagem e do tamanho mínimo dos objetos analisados.

Tabela 4.3: Limite das dimensões para os quais é possível reconhecer objetos

Algoritmo	Dimensões mínimas	Dimensões máximas
LME	32x24	640x480

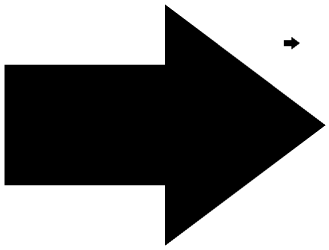


Figura 4.2: Setas de 32x24 e 640x480 pixels, representadas em escala 1:4 ilustrando a proporção da maior e menor setas detectadas pelo sistema

4.1.4 Performance

Devido à dificuldade de encontrar o código fonte de sistemas que operam de maneira similar, não foi possível realizar um estudo comparativo de desempenho. A solução utilizada foi estimar os recursos de *hardware* consumidos pelo sistema quando em execução. Utilizando a ferramenta de linha de comando top, disponível no terminal de comando do Linux, foi monitorado o consumo de ciclos do processador e quantidade de memória utilizada durante a execução do aplicativo.

Tabela 4.4: Consumo de recursos do computador

Algoritmo	Utilização do processador	Consumo de memória
LME	12 a 24%	9 a 13MB

A tabela 4.4 demonstra que, apesar da quantidade alta de cálculos em ponto flutuante realizados para cada imagem fornecida pela câmera, o uso de recursos do computador não foi muito alto. O processador no qual os testes foram realizados é um Intel Core 2 Duo 2.2GHz.

4.1.5 Precisão do ângulo da seta

Quando uma seta é positivamente identificada, um algoritmo simples, baseado na geometria da mesma, calcula o ângulo formado pela reta que divide a seta em duas partes iguais, e um sistema de coordenadas cartesiano virtual alinhado com as bordas da imagem. A precisão do ângulo está diretamente ligada ao tamanho do objeto estudado.

Tabela 4.5: Precisão do ângulo em função das dimensões em pixels da seta

Algoritmo	32x24	256x196	480x320
LME	9°	3°	1°

Esta medida foi feita visando aplicações futuras para controlar um robô. A seta e a direção para o qual aponta poderão servir de base para um sistema de navegação baseado em símbolos visuais.

4.2 Conclusões

Iniciou-se o projeto com o estudo da organização e funcionamento de um sistema de visão computacional. Foram estudados algoritmos de extração de características e reconhecimento de objetos. Foi escolhido o detector LMS para encontrar os pontos dominantes dos contornos devido ao desempenho superior obtido quando comparado aos algoritmos e técnicas tradicionais. O sistema foi utilizado tendo como entrada uma imagem estática, contendo somente uma seta, para criar um padrão de referência para comparar com outros objetos. Com os resultados obtidos foi possível detectar os pontos dominantes da seta e montar um histograma baseado na quantidade de pontos dominantes com mesmo valor de entropia. O histograma obtido passou então a ser utilizado como base de comparação no reconhecimento de setas.
Limitações encontradas no sistema:

- taxa de identificação de falsos positivos próximas a 2% em luz ambiente;
- falsos negativos negativos próximos a 14% em luz ambiente;
- tolerância de aproximadamente 7° em transformações de distorção *(skew transform)*;
- tamanho mínimo dos objetos reconhecidos de 32x24 pixels;
- precisão do ângulo formado entre a seta e o eixo horizontal da imagem variando de 1° a 9°;

Para os casos contidos dentro das restrições citadas acima, pode-se considerar que o sistema detecta de forma robusta os objetos especificados. Assim, pode-se concluir que o sistema é viável para aplicações de detecção de objetos, devido à sua capacidade de reconhecer formas com grande variação na escala e independente de rotação.
O trabalho realizado nesse projeto está longe de ser concluído. Dentre os melhoramentos propostos destacam-se:

- Cálculo automático do parâmetro de distância entre pontos dominantes. Usado para estabelecer o ponto dominante em uma dada região da imagem, este parâmetro afeta a capacidade do sistema de reconhecer objetos grandes e pequenos. Uma distância pequena, próxima de 7 pixels facilitou o reconhecimento de objetos pequenos, próximos do limite estipulado, e dificultou a identificação de objetos grandes, devido à grande quantidade de pontos dominantes gerados. Uma distância grande, próxima de 40 pixels, teve o efeito inverso. Especula-se que exista uma função que calcule o

valor ótimo do parâmetro baseado no tamanho dos objetos, maximizando a taxa de identificação do algoritmo;

- Melhorar o desempenho do algoritmo LME. Atualmente, a primeira transformação linear, que varre a imagem com uma janela 3x3, é executada de modo ineficiente. Sabe-se que os candidatos a pontos dominantes estão entre os pontos pertencentes ao contorno do objeto, e é possível aplicar a transformação linear somente nesse conjunto de pontos, ao invés da imagem inteira. Essa mudança poderá melhorar muito o desempenho do sistema, visto que essa é uma operação de alto custo computacional, e é executada a cada vez que o sistema recebe uma nova imagem;

- Reimplementar rotinas de cálculos auxiliares do sistema. Atualmente todas as operações e parâmetros do sistema são implementadas utilizando números representados em ponto flutuante. Dada a quantidade limitada de valores possíveis para as entropias, pode-se substituir o cálculo do logaritmo usado na equação de entropia por uma tabela fixa, aumentando consideravelmente o desempenho do sistema, dado que a função logaritmo é calculada duas vezes para cada candidato a ponto dominante.

- Utilizar os recursos avançados disponíveis na biblioteca OpenCV. A versão 2.2 da biblioteca introduziu funções geradoras de filtros personalizáveis, que permitem combinar em um único filtro uma combinação de operações, agilizando algumas etapas do sistema, como por exemplo, a etapa de pré-processamento;

- Executar em paralelo as rotinas ortogonais do sistema. Várias das rotinas do sistema podem ser processadas em paralelo, pois funcionam de modo independente umas das outras. Descobertos os candidatos a pontos dominantes, pode-se calcular o valor das entropias das janelas restantes em paralelo, pois as janelas realizam somente operações de leitura da imagem.

A baixa tolerância do sistema a transformações afins já era esperada, dado que essas transformações alteram os ângulos dos pontos dominantes de toda a imagem. Mesmo assim, o sistema conseguiu identificar com sucesso setas que sofreram rotações de cerca de 7° em cada um dos eixos contidos no plano da imagem. Uma possível solução para o problema seria construir um banco de imagens setas pré deformadas, e calcular a distância do histograma do objeto em relação a todos os histogramas do banco.

O sistema desenvolvido mostrou-se apropriado para futuras aplicações envolvendo locomoção de robôs, aplicação essa que motivou inicialmente o desenvolvimento do sistema. Outra aplicação interessante do sistema será fazê-lo operar como um auxiliar de tomadas de decisões em sistemas de navegação veiculares autônomos avançados, visto que pode reconhecer com facilidade os símbolos das placas de trânsito.

REFERÊNCIAS[1]

ABE, K. et al. Comparison of methods for detecting corner points from digital curves-a preliminary report. In: IEEE. *Document Analysis and Recognition, 1993., Proceedings of the Second International Conference on*. [S.l.], 2002. p. 854–857. ISBN 0818649607.

AMIT, Y. *2D object detection and recognition*. [S.l.]: MIT Press, 2002.

ASADA, H.; BRADY, M. The curvature primal sketch. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, n. 1, p. 2–14, 2009. ISSN 0162-8828.

ATTNEAVE, F. Some informational aspects of visual perception. *Psychological review*, v. 61, n. 3, p. 183–193, 1954.

BEUS, H. Lynn; TIU, S.S.H. An improved corner detection algorithm based on chain-coded plane curves. *Pattern Recognition*, Elsevier, v. 20, n. 3, p. 291–296, 1987. ISSN 0031-3203.

BHATTACHARYYA, A. On a measure of divergence between two statistical populations defined by probability distributions. *Bull. Calcutta Math. Soc*, v. 35, p. 99–109, 1943.

BRETZNER, L.; LAPTEV, I.; LINDEBERG, T. Hand gesture recognition using multi-scale colour features, hierarchical models and particle filtering. In: PUBLISHED BY THE IEEE COMPUTER SOCIETY. *fgr*. [S.l.], 2002. p. 0423.

BROSNAN, T.; SUN, D.W. Inspection and grading of agricultural and food products by computer vision systems–a review. *Computers and Electronics in Agriculture*, Elsevier, v. 36, n. 2-3, p. 193–213, 2002. ISSN 0168-1699.

CESAR, R.M.; COSTA, L. da Fontoura. Piecewise linear segmentation of digital contours in O (N. LOG (N)) through a technique based on effective digital curvature estimation. *Real-Time Imaging*, Elsevier, v. 1, n. 6, p. 409–417, 1995. ISSN 1077-2014.

CHETVERIKOV, Dmitry. A simple and efficient algorithm for detection of high curvature points in planar curves. In: PETKOV, Nicolai; WESTENBERG, Michel A. (Ed.). *Computer Analysis of Images and Patterns*. [S.l.]: Springer Berlin / Heidelberg, 2003, (Lecture Notes in Computer Science, v. 2756). p. 746–753.

CHITTA, S.; COHEN, B.; LIKHACHEV, M. Planning for autonomous door opening with a mobile manipulator. In: IEEE. *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. [S.l.], 2010. p. 1799–1806. ISSN 1050-4729.

[1] Elaborado de acordo com a ABNT, sob a norma NBR 6023.

CHOSET, H. Coverage for robotics–A survey of recent results. *Annals of Mathematics and Artificial Intelligence*, Springer, v. 31, n. 1, p. 113–126, 2001. ISSN 1012-2443.

CRANK, J. *The mathematics of diffusion*. [S.l.]: Oxford University Press, USA, 1979. ISBN 0198534116.

DAVIS, L.S. Understanding shape: Angles and sides. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 3, p. 236–242, 2006. ISSN 0018-9340.

DRYDEN, I.L.; MARDIA, K.V. **Statistical shape analysis**. [S.l.]: Wiley New York, 1998.

FASEL, B.; LUETTIN, J. Automatic facial expression analysis: a survey. *Pattern Recognition*, Elsevier, v. 36, n. 1, p. 259–275, 2003. ISSN 0031-3203.

FONTOURA, L. Da; JR, R. Marcondes. **Shape analysis and classification: theory and practice**. [S.l.]: CRC Press: NJ, 2001.

FREEMAN, H.; DAVIS, LS. A corner-finding algorithm for chain-coded curves. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 3, p. 297–303, 2006. ISSN 0018-9340.

GURU, DS; DINESH, R.; NAGABHUSHAN, P. Boundary Based Corner Detecion and Localization Using New "Cornerity" Index: A Robust Approach. IEEE Computer Society, 2004.

JIMENEZ, AR; CERES, R.; PONS, JL. A survey of computer vision methods for locating fruit on trees.

LAMECKER, H.; WENCKEBACH, T.H.; HEGE, H.C. Atlas-based 3D-shape reconstruction from X-ray images. *Pattern Recognition*, v. 1, p. 371–374, 2006. ISSN 1051-4651.

LATECKI, L. J.; LAKAMPER, R.; ECKHARDT, U. *MPEG7 CE Shape-1 Part B (database containing 1400 binary shape images)*. 2010. Disponível em: <http://www.imageprocessingplace.com/root_files_V3/image_databases.htm>.

LENMAN, S.; BRETZNER, L.; THURESSON, B. Computer vision based hand gesture interfaces for human-computer interaction. *Royal Institute of Technology, Sweden*, 2002.

LI, S.Z.; JAIN, A.K.; NETLIBRARY, Inc. *Handbook of face recognition*. [S.l.]: Citeseer, 2005. ISBN 038740595X.

LIU, H.C.; SRINATH, MD. Corner detection from chain-code. *Pattern Recognition*, Elsevier, v. 23, n. 1-2, p. 51–68, 1990. ISSN 0031-3203.

LONCARIC, S. A survey of shape analysis techniques. *Pattern recognition*, Elsevier, v. 31, n. 8, p. 983–1001, 1998. ISSN 0031-3203.

LOURO, Antonio; MACHADO, Will; GONZAGA, Adilson. **Multiscale Local Mixing Entropy**. São Carlos: [s.n.], 2009.

MAITIN-SHEPARD, J. et al. Cloth grasp point detection based on multiple-view geometric cues with application to robotic towel folding. In: IEEE. *Robotics and Automation (ICRA), 2010 IEEE International Conference on*. [S.l.], 2010. p. 2308–2315. ISSN 1050-4729.

MALAMAS, E.N. et al. A survey on industrial vision systems, applications and tools. *Image and Vision Computing*, Elsevier, v. 21, n. 2, p. 171–188, 2003. ISSN 0262-8856.

MARJI, M.; KLETTE, R.; SIY, P. Corner detection and curve partitioning using arc-chord distance. *Combinatorial Image Analysis*, Springer, p. 512–521, 2005.

MARSHALL, S. **Review of shape coding techniques. *Image and Vision Computing*, Elsevier, v. 7, n. 4, p. 281–294, 1989. ISSN 0262-8856.**

MERCIMEK, M.; GULEZ, K.; MUMCU, T.V. Real object recognition using moment invariants. *sadhana*, Springer, v. 30, n. 6, p. 765–775, 2005. ISSN 0256-2499.

MOESLUND, T.B.; HILTON, A.; KRUGER, V. A survey of advances in vision-based human motion capture and analysis. *Computer vision and image understanding*, Elsevier, v. 104, n. 2-3, p. 90–126, 2006. ISSN 1077-3142.

OTSU, N. A threshold selection method from gray-level histograms. *Automatica*, v. 11, p. 285–296, 1975.

PAVLIDIS, T. **Algorithms for graphics and image processing. 1982.**

PEI, S.C.; LIN, C.N. The detection of dominant points on digital curves by scale-space filtering. *Pattern recognition*, Elsevier, v. 25, n. 11, p. 1307–1314, 1992. ISSN 0031-3203.

RATTARANGSI, A.; CHIN, RT. Scale-based detection of corners of planar curves. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, IEEE, v. 14, n. 4, p. 430–449, 2002. ISSN 0162-8828.

ROSENFELD, A.; JOHNSTON, E. Angle detection on digital curves. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 9, p. 875–878, 2009. ISSN 0018-9340.

ROSENFELD, A.; WESZKA, JS. An improved method of angle detection on digital curves. *Computers, IEEE Transactions on*, IEEE, v. 100, n. 9, p. 940–941, 2006. ISSN 0018-9340.

SARFRAZ, M.; ASIM, MR; MASOOD, A. Piecewise polygonal approximation of digital curves. In: IEEE. *Information Visualisation, 2004. IV 2004. Proceedings. Eighth International Conference on*. [S.l.], 2004. p. 991–996. ISBN 0769521770. ISSN 1093-9547.

SARFRAZ, M.; MASOOD, A.; ASIM, MR. A new approach to corner detection. *Computer Vision and Graphics*, Springer, p. 528–533, 2006.

SAYKOL, E.; GUDUKBAY, U.; ULUSOY, O. A histogram-based approach for object-based query-by-shape-and-color in image and video databases. *Image and Vision Computing*, Elsevier, v. 23, n. 13, p. 1170–1180, 2005. ISSN 0262-8856.

SMITH, P. et al. Effective corner matching. In: CITESEER. *British machine vision conference*. [S.l.], 1998. p. 545–556.

SRINIVASAN, P.; LIANG, P.; HACKWOOD, S. Computational geometric methods in volumetric intersection for 3d reconstruction. *Pattern Recognition*, Elsevier, v. 23, n. 8, p. 843–857, 1990. ISSN 0031-3203.

TANG, K. T. **Mathematical Methods for Engineers and Scientists 3: Fourier Analysis, Partial Differential Equations and Variational Methods**. [S.l.]: Springer, 2008. ISBN 3540446958.

THACKER, N.A.; AHERNE, F.J.; ROCKETT, P.I. The Bhattacharyya metric as an absolute similarity measure for frequency coded data. *Kybernetika*, Citeseer, v. 34, n. 4, p. 363–368, 1997.

THOMAS, ADH et al. Real-time industrial visual inspection: a review. *Real-Time Imaging*, Elsevier, v. 1, n. 2, p. 139–158, 1995. ISSN 1077-2014.

TRIER, Ř. Due; JAIN, A.K.; TAXT, T. Feature extraction methods for character recognition-a survey. *Pattern recognition*, Elsevier, v. 29, n. 4, p. 641–662, 1996. ISSN 0031-3203.

VINCENT, E.; LAGANIÈRE, R. Detecting and matching feature points. *Journal of Visual Communication and Image Representation*, Elsevier, v. 16, n. 1, p. 38–54, 2005. ISSN 1047-3203.

ZHANG, D.; LU, G. Review of shape representation and description techniques. *Pattern recognition*, Elsevier, v. 37, n. 1, p. 1–19, 2004. ISSN 0031-3203.

Apêndice A

DEFS.H

```
1 #ifndef SOURCE_DEFS_H_
2 #define SOURCE_DEFS_H_
3
4 #include <stdint.h>
5
6 #ifdef __cplusplus
7   #include <vector>
8   #include <list>
9   #include <iomanip>
10  #include <iostream>  // NOLINT(readability/streams) for debugging only
11  #include <cmath>
12
13 #endif
14
15 #include "opencv/cv.h"
16
17 typedef int        INT;
18 typedef unsigned int  UINT;
19 typedef int16_t      SHORT;
20 typedef uint16_t       USHORT;
21 typedef float        FLOAT;
22 typedef double       DOUBLE;
23 typedef char       CHAR;
24 typedef unsigned char   UCHAR;
25 typedef void       VOID;
26 typedef bool       BOOL;
27 #define FALSE      false
28 #define TRUE       true
29
30 typedef INT        Pixel;
31 typedef FLOAT        PointStrength;
32 typedef FLOAT        Tolerance;
33 typedef cv::Point2i    Point;
34 typedef cv::Vec6f      Entropy;
35
36 #define SIZEOF(x)    ((INT) sizeof((x))/sizeof((x[0])))
37
38 #ifdef TRACE
39   #define _TRACE (std::cerr << "[File: " << __FILE__ << " Line: " << \
40                                 __LINE__ << " Func: " << __FUNCTION__ << \
41                                 std::endl)
42 #endif
43 #endif  // SOURCE_DEFS_H_
```

```
1 #include <vector>
2
3 #include "source/Defs.h"
4
5 #include "source/ExpectedEntropies.h"
6 #include "source/ModifiedGaussianWindows.h"
7 #include "source/EntropyStruct.h"
8 #include "source/lme.h"
9
10 #include "opencv/cv.hpp"
11
12 FLOAT Log2(FLOAT n)
13 {
14     // log(n)/log(2) is log2.
15     return log(n) / log(2);
16 }
17
18 namespace lme
19 {
```

```
20 Clme::Clme()
21 : m_ConvexTolerance(0)
22 , m_ConcaveTolerance(0)
23 , m_PixelDistance(0)
24 , m_EntropyList()
25
26 {
27   _TRACE;
28 }
29
30 Clme::~Clme()
31 {
32   _TRACE;
33 }
34
35 VOID Clme::Inicio(const cv::Mat & aImage)
36 {
37   _TRACE;
38   cv::Mat copy = aImage.clone();
39   Tracing(copy);
40   Window3(copy);
41 }
42
43 FLOAT Clme::CalculateEntropy(const FLOAT & aEntropy) const
44 {
45   _TRACE;
46   FLOAT p0 = aEntropy;
47   FLOAT p1 = 1 - p0;
48   return -(p0 * Log2(p0) + p1 * Log2(p1));
49 }
50
51 VOID Clme::CalculatePointStrength()
52 {
53   _TRACE;
54   std::vector<lme::CEntropyStruct*>::iterator it = m_EntropyList.begin();
55   std::vector<lme::CEntropyStruct*>::const_iterator it_end = m_EntropyList.end();
56
57   /* For every candidate: */
58   for(;it != it_end;++it)
59   {
60     lme::CEntropyStruct *entropy = *it;
61     cv::Vec6f vec = entropy->GetEntropy();
62     // Calculae average speed
63     FLOAT mv = vec.dot(cv::Vec6f::all(1.0)) / 6.0;
64     // Calculate average acceleration
65     FLOAT ma = (vec[1] - vec[5]) / 6.0;
66     // Set point strength as avg speed times avg acceleration
67     entropy->SetPointStrength(mv * ma);
68   }
69 }
70
71 VOID Clme::CalculateDominatPoint()
72 {
73   _TRACE;
74   Pixel pixelDistance = GetPixelDistance();
75
76   std::vector<lme::CEntropyStruct*>::iterator it = m_EntropyList.begin();
77   std::vector<lme::CEntropyStruct*>::const_iterator it_end =
78       m_EntropyList.end();
79
80   std::vector<lme::CEntropyStruct*> newEntropyList;
81
82   // i2t points to past element, it points to current element
83   std::vector<lme::CEntropyStruct*>::iterator i2t = it++;
84
85   /* For every candidate: */
86   for(;it != it_end;++it)
87   {
88     Point p1 = (*it)->Coordinate();
89     Point p2 = (*i2t)->Coordinate();
90
91     Pixel distance = ManhattanDistance(p1, p2);
92
93     if(distance < pixelDistance)
94     {
95       // Calculate dominant candidate
96       if((*it)->GetPointStrength() < (*i2t)->GetPointStrength())
97         newEntropyList.push_back((*i2t));
98       else
99         newEntropyList.push_back((*it));
```

```
100       }
101     }
102     // Substitute old list with new list
103     m_EntropyList = newEntropyList;
104 }
105
106 Pixel Clme::ManhattanDistance(Point aPt1, Point aPt2) const
107 {
108   _TRACE;
109   // Horizontal distance
110   INT dx = aPt1.x - aPt2.x;
111
112   // Vertical distance
113   INT dy = aPt1.y - aPt2.y;
114
115   return std::abs(dx + dy);
116 }
117
118     VOID Clme::AdjustEntropies(Tolerance aTolx, Tolerance aToly)
119     {
120         _TRACE;
121         std::vector<lme::CEntropyStruct*>::iterator it = m_EntropyList.begin();
122         std::vector<lme::CEntropyStruct*>::const_iterator it_end = m_EntropyList.end();
123
124         /* For every candidate: */
125         for(;it != it_end;++it)
126       AdjustEntropy(*it);
127     }
128
129 VOID Clme::AdjustEntropy(lme::CEntropyStruct * aEntropyStruct)
130 {
131   _TRACE;
132   const Entropy e = aEntropyStruct->GetEntropy();
133
134   for(INT i = 0; i < (INT) SIZEOF(e); i++)
135   {
136     FLOAT entropy = e[i];
137
138     // Check if candidate is convex
139     if(entropy > ee[i][4])
140     {
141       if(entropy < (ee[i][4] + ee[i][3])/2.0)
142         aEntropyStruct->SetEntropy(i, ee[i][4]);
143       else if (entropy < (ee[i][3] + ee[i][2])/2.0)
144         aEntropyStruct->SetEntropy(i, ee[i][3]);
145       else if (entropy < (ee[i][2] + ee[i][1])/2.0)
146         aEntropyStruct->SetEntropy(i, ee[i][2]);
147       else if (entropy < (ee[i][1] + ee[i][0])/2.0)
148         aEntropyStruct->SetEntropy(i, ee[i][1]);
149       else
150         aEntropyStruct->SetEntropy(i, ee[i][0]);
151     }
152     // Check if candidate is concave
153     else if(entropy < ee[i][5])
154     {
155       if(entropy > (ee[i][5] + ee[i][6])/2.0)
156         aEntropyStruct->SetEntropy(i, ee[i][5]);
157       else if (entropy > (ee[i][6] + ee[i][7])/2.0)
158         aEntropyStruct->SetEntropy(i, ee[i][3]);
159       else if (entropy < (ee[i][2] + ee[i][1])/2.0)
160         aEntropyStruct->SetEntropy(i, ee[i][2]);
161       else if (entropy < (ee[i][1] + ee[i][0])/2.0)
162         aEntropyStruct->SetEntropy(i, ee[i][1]);
163       else
164         aEntropyStruct->SetEntropy(i, ee[i][0]);
165     }
166     // Check if candidate is diagonal or horizontal/vertical segment
167     else
168     {
169       if (entropy > (ee[i][4] + ee[i][5])/2.0)
170         aEntropyStruct->SetEntropy(i, ee[i][4]);
171       else
172         aEntropyStruct->SetEntropy(i, ee[i][5]);
173     }
174   }
175 }
176
177 VOID Clme::SetPixelDistance(Pixel aPixel)
178 {
179   _TRACE;
```

```
180   m_PixelDistance = aPixel;
181 }
182
183 Pixel Clme::GetPixelDistance() const
184 {
185   _TRACE;
186   return m_PixelDistance;
187 }
188
189 VOID Clme::Tracing(cv::Mat & aImg_rgb)
190 {
191   _TRACE;
192   cv::Mat im_gray;
193   cv::Mat im_smooth;
194   cv::Mat im_bw;
195   cv::Mat im_er;
196
197   // turn the input image in binary (object=1,back=0)
198   cv::cvtColor(aImg_rgb, im_gray, CV_RGB2GRAY);
199
200   cv::medianBlur(im_gray, im_smooth, 5);
201
202   cv::morphologyEx(im_smooth, im_er, cv::MORPH_OPEN, cv::Mat(), cv::Point(-1, -1), 2, cv::BORDER_REPLICATE);
203
204   cv::threshold(im_er, im_bw, 0.0, 255.0, cv::THRESH_BINARY_INV | cv::THRESH_OTSU);
205
206   aImg_rgb = im_bw;
207 }
208
209 VOID Clme::Windows(cv::Mat & aBinaryImg)
210 {
211   _TRACE;
212
213   cv::Mat temp;
214
215   std::vector<lme::CEntropyStruct*>::iterator it = m_EntropyList.begin();
216   std::vector<lme::CEntropyStruct*>::const_iterator it_end = m_EntropyList.end();
217
218   FLOAT entropy = 0.0;
219
220   /* Add possible candidate to list */
221   for(;it != it_end;++it)
222   {
223     // Get Point coordinates of corner candidate
224     const cv::Point pt = (*it)->Coordinate();
225
226     // Build a ROI of 1x1 around point location
227     cv::Mat point(aBinaryImg, cv::Rect(pt.x, pt.y, 1, 1));
228
229     // Calculate candidate entropies on all windows sizes
230     for(INT i = 0;i < EntropiesCount;i++)
231     {
232       // Make diffusion happen...
233       cv::filter2D(point, temp, -1, window[i], cv::Point(-1, -1), 0.0, cv::BORDER_CONSTANT);
234
235       entropy = CalculateEntropy(temp.at<FLOAT>(pt));
236
237       (*it)->SetEntropy(i, entropy);
238     }
239   }
240 }
241
242 VOID Clme::Window3(cv::Mat & aBinaryImg)
243 {
244   _TRACE;
245
246   cv::Mat temp;
247
248   // Filter with 3x3 window...
249   cv::filter2D(aBinaryImg, temp, -1, window[mg3], cv::Point(-1, -1), 0.0, cv::BORDER_CONSTANT);
250
251   cv::MatIterator_<FLOAT> it = temp.begin<FLOAT>();
252   cv::MatConstIterator_<FLOAT> it_end = temp.end<FLOAT>();
253
254   FLOAT entropy = 0.0;
255   for(;it != it_end;++it)
256   {
257     entropy = CalculateEntropy(*it);
258
259     // Check if entropy corresponds to noise on image
```

```
260       if((entropy > lme::Clme::MinimumEntropy) &&
261           (entropy < lme::Clme::MaximumEntropy))
262       {
263         m_EntropyList.push_back(new CEntropyStruct(it.pos(), entropy));
264       }
265     }
266 }
267
268 VOID Clme::SetConcaveTolerance(const Tolerance & aConcaveTolerance)
269 {
270   _TRACE;
271   m_ConcaveTolerance = aConcaveTolerance;
272 }
273
274 VOID Clme::SetConvexTolerance(const Tolerance & aConvexTolerance)
275 {
276   _TRACE;
277   m_ConvexTolerance = aConvexTolerance;
278 }
279
280 Tolerance Clme::GetConcaveTolerance()
281 {
282   _TRACE;
283   return m_ConcaveTolerance;
284 }
285
286 Tolerance Clme::GetConvexTolerance()
287 {
288   _TRACE;
289   return m_ConvexTolerance;
290 }
291
292 VOID Clme::CalculateHistogram()
293 {
294   _TRACE;
295 }
296 }
```

```
1  #include "source/EntropyStruct.h"
2
3  namespace lme
4  {
5      CEntropyStruct::CEntropyStruct(INT row, INT col, FLOAT W3E, FLOAT W5E,
6          FLOAT W9E, FLOAT W17E, FLOAT W25E, FLOAT W35E)
7      : m_Coordinates(row, col)
8      , m_Entropies(W3E, W5E, W9E, W17E, W25E, W35E)
9      , m_PointStrength(0)
10     {
11         _TRACE;
12     }
13
14     CEntropyStruct::CEntropyStruct(const Point & aCoordinate,
15         const Entropy & aEntropy)
16     : m_Coordinates(aCoordinate)
17     , m_Entropies(aEntropy)
18     , m_PointStrength(0)
19     {
20         _TRACE;
21     }
22
23     CEntropyStruct::CEntropyStruct(const Point & aCoordinate,
24         const DOUBLE & aEntropy)
25     : m_Coordinates(aCoordinate)
26     , m_Entropies()
27     , m_PointStrength(0)
28     {
29         _TRACE;
30         SetEntropy(0, (FLOAT)(aEntropy));
31     }
32
33     CEntropyStruct::~CEntropyStruct()
34     {
35         _TRACE;
36     }
37
38     VOID CEntropyStruct::SetEntropy(INT aWindow, FLOAT aEntropy)
39     {
40         _TRACE;
41         m_Entropies[aWindow] = aEntropy;
42     }
```

```
43
44      FLOAT CEntropyStruct::GetEntropy(INT aWindow) const
45      {
46          _TRACE;
47          return m_Entropies[aWindow];
48      }
49
50      PointStrength CEntropyStruct::GetPointStrength() const
51      {
52        _TRACE;
53          return m_PointStrength;
54      }
55
56      VOID CEntropyStruct::SetPointStrength(const PointStrength & aPointStrength)
57    {
58        _TRACE;
59      this->m_PointStrength = aPointStrength;
60    }
61
62
63      Entropy CEntropyStruct::GetEntropy() const
64      {
65        _TRACE;
66          return m_Entropies;
67      }
68
69      Point CEntropyStruct::Coordinate() const
70      {
71          _TRACE;
72          return m_Coordinates;
73      }
74  }
```

```
1  #ifndef SOURCE_LME_H_
2  #define SOURCE_LME_H_
3
4  #include <vector>
5
6  #include "opencv/cv.h"
7  #include "opencv/highgui.h"
8
9  namespace lme
10 {
11 class CEntropyStruct;
12
13 class Clme
14 {
15  public:
16
17   Clme();
18
19   ~Clme();
20
21   VOID Inicio(const cv::Mat & aImage);
22
23   VOID Tracing(cv::Mat & aImage);
24
25   VOID SetPixelDistance(Pixel aPixel);
26   Pixel GetPixelDistance() const;
27
28   VOID SetConcaveTolerance(const Tolerance & aConcaveTolerance);
29   VOID SetConvexTolerance(const Tolerance & aConvexTolerance);
30
31   Tolerance GetConcaveTolerance();
32   Tolerance GetConvexTolerance();
33
34  private:
35
36   VOID AdjustEntropies(Tolerance aTolx, Tolerance aToly);
37   VOID AdjustEntropy(lme::CEntropyStruct * aEntropyStruct);
38
39   VOID Window3(cv::Mat & aBinaryImg);
40   VOID Windows(cv::Mat & aBinaryImg);
41
42   FLOAT CalculateEntropy(const FLOAT & aEntropy) const;
43   VOID CalculatePointStrength();
44
45   VOID CalculateDominatPoint();
46
47   VOID CalculateHistogram();
```

```
48
49    /** Calculates Manhattan distance between two points */
50    /**
51     * @param aPt1 reference point
52     * @param aPt2 second point
53     */
54    Pixel ManhattanDistance(Point aPt1, Point aPt2) const;
55
56    static const Tolerance MinimumEntropy = 0.5;
57    static const Tolerance MaximumEntropy = 0.955;
58
59    // Store convex tolerance parameter
60    Tolerance m_ConvexTolerance;
61
62    // Store concave tolerance parameter
63    Tolerance m_ConcaveTolerance;
64
65    Pixel m_PixelDistance;
66
67    std::vector<CEntropyStruct *> m_EntropyList;
68 };
69 }
70 #endif  // SOURCE_LME_H_
```

```
1  #ifndef SOURCE_ENTROPYSTRUCT_H_
2  #define SOURCE_ENTROPYSTRUCT_H_
3
4  #include "source/Defs.h"
5
6  namespace lme
7  {
8  class CEntropyStruct
9  {
10  public:
11
12    // Default constructor
13    explicit CEntropyStruct(const INT aRow = 0, const INT aCol = 0,
14        const FLOAT aW3E = 0, const FLOAT aW5E = 0, const FLOAT aW9E = 0,
15        const FLOAT aW17E = 0, const FLOAT aW25E = 0,
16        const FLOAT aW35E = 0);
17
18    CEntropyStruct(const Point & aCoordinate, const Entropy & aEntropy);
19
20    CEntropyStruct(const Point & aCoordinate, const DOUBLE & aEntropy);
21
22    virtual ~CEntropyStruct();
23
24    //  Set Point Coordinate
25    VOID SetCoordinate(const INT aRow, const INT aCol);
26    VOID SetCoordinate(const Point & aCoordinate);
27
28    // Get Point Coordinate
29    Point Coordinate() const;
30
31    // Set Window entropy
32    VOID SetEntropy(INT aWindow, const FLOAT aEntropy);
33
34    // Get Window entropy
35    FLOAT GetEntropy(INT aWindow) const;
36    Entropy GetEntropy() const;
37
38    VOID SetPointStrength(const PointStrength & aStrength);
39    PointStrength GetPointStrength() const;
40
41  private:
42    // Point coordinates
43    Point m_Coordinates;
44
45    // Point entropies
46    Entropy m_Entropies;
47
48    // Value used to compare point strengths
49    PointStrength m_PointStrength;
50 };
51 }
52 #endif  // SOURCE_ENTROPYSTRUCT_H_
```