Universidade Federal de Minas Gerais
Centro de Pesquisa e Desenvolvimento em Engenharia Elétrica
Programa de Pós-Graduação em Engenharia Elétrica

Guilherme Damasceno Silva

Desenvolvimento de um Sistema de Visão Computacional para o Estudo do Comportamento de Animais Experimentais

Belo Horizonte 2008

Universidade Federal de Minas Gerais
Centro de Pesquisa e Desenvolvimento em Engenharia Elétrica
Programa de Pós-Graduação em Engenharia Elétrica

Desenvolvimento de um Sistema de Visão Computacional para o Estudo do Comportamento de Animais Experimentais

Guilherme Damasceno Silva

Dissertação de Mestrado submetida à Banca Examinadora designada pelo Colegiado do Programa de Pós-Graduação em Engenharia Elétrica da Universidade Federal de Minas Gerais como parte dos requisitos necessários à obtenção do grau de Mestre em Engenharia Elétrica.

Orientador: Homero Nogueira Guimarães

Belo Horizonte 2008

(Folha de Aprovação a ser anexada)

“ Muitas vezes as pessoas são egocêntricas, ilógicas e insensatas.
Perdoe-as assim mesmo.

Se você é gentil, as pessoas podem acusá-lo de egoísta, interesseiro.
Seja gentil assim mesmo.

Se você é vencedor, terá alguns falsos amigos e alguns inimigos verdadeiros.
Vença assim mesmo.

Se você é honesto e franco, as pessoas podem enganá-lo.
Seja honesto e franco assim mesmo.

O que você levou anos para construir, alguém pode destruir de uma hora para outra.
Construa assim mesmo.

O bem que você faz hoje pode ser esquecido amanhã.
Faça o bem assim mesmo.

Dê ao mundo o melhor de você, mas isso pode nunca ser o bastante.
Dê o melhor de você assim mesmo.

Veja você que, no final das contas, é entre você e Deus.
Nunca foi entre você e as outras pessoas.”

(Madre Tereza de Calcutá)

Dedicatória

Aos meus pais **Dayse e Antonio**,

Que momento de alegria...
Pensei que jamais este dia chegaria...
Finalmente... O dia da minha defesa de mestrado...
Foram anos de muito esforço e dedicação,
em que fui amparado pela ternura de meus pais tão queridos,
cujos esforços jamais serão medidos, sempre me dando amparo e carinho...
Gostaria de dizer MUITO OBRIGADO, por este esforço ora recompensado...
E gostaria que aceitassem este beijo que lhes estou a dar, pelo amor que sempre me dedicaram, e pelo apoio que sempre me propiciaram...
Meu amor e reconhecimento eterno...

Agradecimentos

Chegou a hora de agradecer àquelas pessoas que, de alguma forma, contribuíram para a realização deste trabalho. Pessoas que me incentivaram de diferentes maneiras. A todos vocês o meu sincero “MUITO OBRIGADO”!

Agradeço em primeiro lugar a **Deus** que iluminou o meu caminho durante esta caminhada;

Aos professores do **PPGEE** por terem sido pessoas essenciais em minha formação, em especial, ao meu orientador **Homero Nogueira Guimarães**;

À minha **irmã** por ter sido uma pessoa fundamental em minha criação;

À minha namorada **Jakelyne Machado Lima**, que de forma especial e carinhosa me deu força e coragem me apoiando nos momentos de dificuldade;

Ao meu amigo **André Paim Lemos** por ter se mostrado uma pessoa prestativa e por ter me ajudado em tudo que precisei para o desenvolvimento deste trabalho;

Ao meu amigo **Marcos Flávio Silveira Vasconcelos D’Angelo**, por ter me auxiliado em tudo que precisei com boa vontade e carinho;

Ao meu padrinho **Eduardo Jorge Carneiro Soares** por ter me recebido de braços abertos e por ter me tratado de forma especial nesse período em que estive em Belo Horizonte;

À minha tia **Ana Cleide Polles Freire**, ao **Carlyle do Carmo Junior**, ao **Lucas Freire do Carmo** e ao **Carlyle do Carmo**, por terem sido pessoas fundamentais, amigas e se mostrado pessoas prestativas me ajudando no que precisei, jamais esquecerei;

Ao meu grande amigo e irmão **Davidson Lopes de Figueiredo**, por ter sido uma pessoa especial em minha vida nesta etapa que passei em Belo Horizonte;

Ao meu grande amigo **Marco Antonio Peroni**, pela amizade sincera e pelos momentos de alegria que passamos;

Ao casal **Henrique Resende Martins** e **Flávia Alcântara**, pelos momentos de alegria e felicidade que passamos juntos e pela contribuição dada no desenvolvimento deste trabalho.

Resumo

Visão Computacional é a área da ciência que se dedica a desenvolver teorias e métodos voltados à extração automática de informações contidas em imagens.

Nesta pesquisa, foi desenvolvido um sistema de monitoramento de animais experimentais utilizando alguns desses métodos. Esse sistema tem como objetivo automatizar os experimentos utilizados pela fisiologia do comportamento e psicofarmacologia, os quais buscam caracterizar o comportamento dos animais experimentais (geralmente roedores) na exploração de labirintos ou campo aberto, situações normais, patológicas e sob tratamento com drogas padrão e experimentais.

Para isto, diversas etapas foram desenvolvidas, desde a parte de análise de requisitos, estudo do comportamento animal, implementação do *software*, realização de experimentos, até a análise dos resultados.

Descrevemos as etapas do desenvolvimento deste sistema de monitoramento, começando com o estudo das técnicas de processamento de sinais candidatas, passando pelas técnicas de caracterização de comportamento animal, a especificação do sistema, a escolha da câmera digital, calibração, implementação do sistema propriamente dito, até os experimentos piloto realizados com camundongos.

Os resultados encontrados nos mostraram que no atual estágio da implementação já é possível fazer uma análise da ocupação do campo de observação pelo animal experimental de forma totalmente automática, fazer uma estatística dessa ocupação, o que é de grande relevância para estudos de psicofarmacologia.

Palavras – Chaves: Animais experimentais, Sistema de Monitoramento, Visão Computacional, Comportamento Animal.

Abstract

Computer vision is the area of science that is dedicated to developing theories and methods focused on the automatic extraction of information contained in images.

In this research, a tracking system for experimental animals using some of these methods was developed. This system aims to automate the experiments used by the physiology of behavior, psychopharmacology, with the purpose of characterize the behavior of experimental animals (usually rodents) in the exploration of mazes or open field, normal situations, pathological and under treatment with standard and experimental drugs.

For this, various steps have been taken from the part of requirement analysis, study of animal behavior, implementation of the software, carrying out experiments, to analyze the results.

We describe the stages of development of this monitoring system, starting with the study of the candidates techniques of signal processing, through the techniques of characterization of animal behavior, the specification of the system, the choice of digital camera, calibration, implementation of the system itself, until the pilot experiments conducted on mice.

The results showed that in the current stage of implementation it is possible to do a fully automated analysis of the experimental animal observation field occupation, make a statistic of that occupation, which is of great importance for studies of psychopharmacology.

Keywords: Experimental Animals, System Monitoring, Computer Vision, Animal Behavior.

Sumário

Lista de Abreviaturas e Siglas

SIGLA	DESCRIÇÃO
UML	Unified Modeling Language
CV	Pacotes de processamento de imagens da Biblioteca OpenCV
CXCORE	Pacotes de Manipulação de arrays da Biblioteca OpenCV
HighGUI	Pacotes gráficos da Biblioteca OpenCV
ML	Pacotes de máquinas de aprendizagem da Biblioteca OpenCV
I/O	Entrada e Saída
Fx	Posição na Coordenada X
Fy	Posição na Coordenada Y
Pin Hole	Câmera sem Lente
3D	Três Dimensões
ER	Engenharia de Requisitos

Capítulo 1 – Introdução

A visão é um dos sentidos mais importantes para os seres humanos. O ser humano é um animal que, apesar de ter todos os sentidos (visão, olfato, paladar, audição, tato e a propriocepção), depende de sua visão por ser o sentido que lhe fornece uma quantidade significativa de informações, facilitando assim, a sua percepção do mundo exterior.

Hoje em dia, as tecnologias que simulam a visão humana estão em constante desenvolvimento. Essas possuem como foco os problemas que estão relacionados às questões onde os seres humanos possuem deficiências, como nas tarefas de inspeção, classificação e monitoramento em geral, podem ser prejudiciais e cansativas à saúde humana, e/ou nas tarefas de medição *óptica* de precisão, por não estarem dentro das condições estabelecidas pela visão humana.

Um dos desafios para a visão computacional está relacionado com a automatização das observações do comportamento de animais, necessária em estudos científicos de psicofisiologia e psicofarmacologia.

Em várias pesquisas, o estudo do comportamento animal é feito de forma manual. Porém, deve se considerar que o observador possui a necessidade de presenciar todo o experimento para obter as informações relevantes nas pesquisas. Dessa forma, o registro manual exige um trabalho exaustivo dos pesquisadores, devido ao tempo de observação, o que compromete o registro de comportamentos de interesse.

O monitoramento do comportamento dos animais experimentais por meio de câmeras facilita a investigação dos pesquisadores nesta área, fornecendo assim informações confiáveis e consistentes durante um longo período de pesquisa.

Uma das principais contribuições deste trabalho é apresentar um software multiplataforma, e de código-fonte aberto, aos sistemas atualmente disponíveis no mercado, na maioria das vezes com alto custo e pouco flexíveis. Além disso, o pesquisador pode utilizar diversos dispositivos de captura de imagens digitais de baixo custo.

Os pesquisadores estão interessados em analisar padrões de comportamento que serão modificados pela ação psicotrópica de fármacos, dentro de um contexto de testes, desenvolvimento de novas drogas e predicamentos.

Outro ponto relevante do monitoramento por vídeo é a sua utilização para caracterização do comportamento locomotor dos animais, como por exemplo, a distância percorrida, a velocidade, a aceleração e outros comportamentos que podem acontecer ao longo do dia.

Esta dissertação tem como objetivo geral desenvolver um *software* com algumas funcionalidades para o monitoramento de animais experimentais utilizando visão computacional para estudos científicos de psicofisiologia e psicofarmacologia, e através da automatização do processo de observações, fornecer dados para auxiliar pesquisadores na análise dos padrões de comportamento. As informações obtidas deverão permitir um pós-processamento e apresentação em formatos exportáveis para programas estatísticos.

Para atender ao objetivo geral foram delineados os seguintes objetivos específicos: realizar estudos relacionados às ferramentas atuais de visão computacional, mais especificamente, as de processamento de imagens, realizar estudos referentes às técnicas de análise de comportamento de animais experimentais em labirintos e em campo aberto, especificar o sistema, desenvolver o sistema, validar o sistema através de experimento piloto, realizar estudos de técnicas de análise de dados com o objetivo de identificar alguns padrões de comportamento, através da análise dos eventos gerados pela movimentação do animal.

No capítulo 2 trataremos da visão computacional, suas propriedades e suas etapas de processamento fazendo um breve histórico do processamento de imagens e como estes são utilizados.

No capitulo 3 trataremos do comportamento animal e da importância de seu estudo para a psicofarmacologia.

No capitulo 4 discutiremos a especificação do *software* de monitoramento de animais experimentais.

No capítulo 5 discutiremos e detalharemos a implementação do *software* de monitoramento.

No capítulo 6 apresentaremos os resultados obtidos com os animais experimentais (camundongos) detalhando o funcionamento do *software*.

Por fim, no capítulo 7, apresentaremos nossas considerações finais sobre o sistema desenvolvido.

Capítulo 2 - Visão Computacional e Processamento de Imagem

2.1 Visão Computacional

Visão Computacional é a área da ciência que estuda e procura desenvolver teorias e métodos voltados à extração automática de informações úteis contidas em imagens. Tais imagens são capturadas por dispositivos como câmera de vídeo e scanner (Crowley e Christensen, 1995).

Por este motivo, a recente evolução tecnológica dos computadores e dispositivos de imagem, a preços acessíveis ao consumidor, tem possibilitado a crescente aplicação de visão computacional nas mais diversas áreas. Como por exemplo, podemos citar: análise automática de sêmen humano, medição computadorizada do dimensional de peças, monitoramento de animais experimentais, análise morfológica de células, reconhecimento e sintetização de faces humanas, entre outras (Faugeras, 1993).

A criação de sistemas de visão computacional, seja para aplicações em inspeção industrial ou para navegação de robôs em movimento, envolve, quase sempre, a execução de um determinado conjunto de transformações em dados obtidos de sensores como câmeras e sonares. Para esta classe de problemas é possível a determinação de uma arquitetura de *software* que suporte as principais etapas do desenvolvimento deste tipo de sistema, enfatizando o reaproveitamento de código e minimizando o esforço despendido em atividades repetitivas (Hartley e Zisserman, 2003).

O objetivo da construção de máquinas inteligentes[1] é expandir os sentidos através dos quais o computador pode se comunicar com o mundo exterior. A utilização da visão da máquina aumenta as aplicações em computadores, como por exemplo, navegação móvel por robô, tarefas complexas de manufatura, análise de imagens de satélites e processamento de imagens médicas.

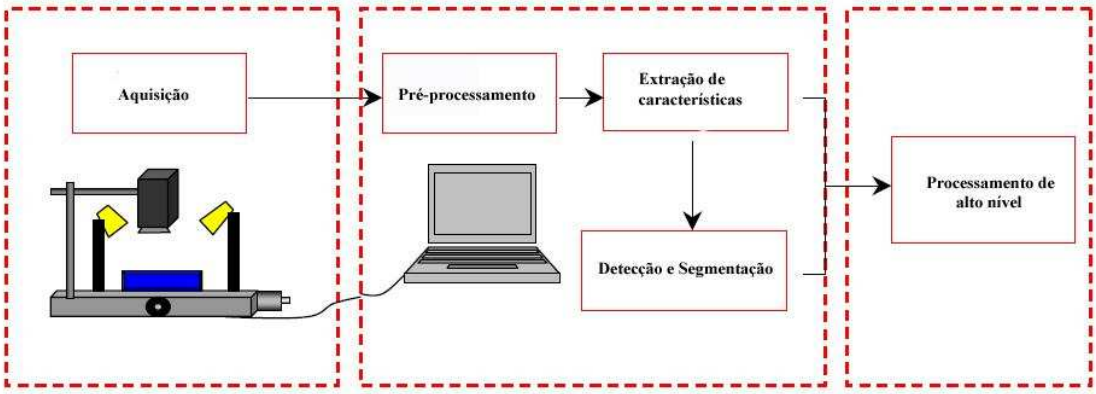
A visão computacional procura oferecer, de modo mais eficiente possível, uma vasta quantidade de informações ao sistema computacional. O reconhecimento de padrões está engajado no campo da visão computacional com atuações e perspectivas importantes para alcançar e realizar a máquina inteligente (Jähne, 2002).

[1] Neste trabalho, máquinas que utilizam visão computacional.

2.1.1 Sistemas de visão computacional

Para todo sistema de visão computacional mostrado na figura 2.1 são necessárias algumas etapas de processamento que variam de acordo com a aplicação. Na maioria dos casos, ocorrem as seguintes etapas:

- **Aquisição de imagem:** todas as imagens digitais são criadas por um ou mais sensores. De acordo com cada sensor pode-se ter vários tipos de imagens, como bidimencional, tridimencional ou ainda uma sequência de imagens. Cada *pixel* tem um valor agregado e cada valor é responsavel por uma intensidade de luz em uma ou em várias faixas de cor. (Morris, 2004).
- **Pré-processamento:** antes da aplicação de qualquer método de visão computacional é necessário verificar se a imagem satisfaz as condições do método a ser utilizado. Para isso podemos citar alguns exemplos: remapeamento (para assegurar o sistema de coordenadas), redução de ruídos (para assegurar que as informações são verdadeiras) e aumento de contraste (para assegurar que as informações relevantes serão detectadas). No monitormaneto de animais experimentais o aumento de contraste foi muito importante, com o intuito de detectar o objeto (Morris, 2004).
- **Extração de características:** existem várias características matemáticas que se pode obter em uma imagem em diversos níveis de complexidade. Como exemplo básico inclui-se detecção de bordas, cantos ou pontos. Exemplos mais complexos incluem detecção de textura, formato e movimento (Morris, 2004).
- **Detecção e segmentação**: durante uma parte processo sempre se toma uma determinada decisão em relação às regiões de interesse que serão processadas no futuro. Como exemplo citamos algumas: seleção de regiões de interesse específicos e segmentação de uma ou mais regiões que contém um objeto de interesse (Morris, 2004).
- **Processamento de alto nível:** nesta etapa a entrada é definida como um conjunto pequeno de dados, ou seja, no caso de monitoramento de animais experimentais, o dado é apenas o objeto de interesse já detectado. O processo seguinte caracteriza-se pela verificação da satisfação dos dados, pela estimativa de parâmetros sobre a imagem e pela classificação dos objetos detectados em diferentes categorias (Morris, 2004).



2.1. Esquema de um sistema de visão computacional

2.2 Breve Histórico de Processamento de Imagens

A melhoria da informação visual e o processamento de dados de cenas para a leitura através de máquinas são considerados as duas áreas principais que vêm despertando interesse em métodos de processamento de imagens digitais.

A utilização do processamento de imagens tem como objetivo principal a melhoria do aspecto visual de feições estruturais para que a análise humana seja beneficiada, além do que, com a utilização de suas técnicas, é possível fornecer informações para a manipulação da imagem e a geração de produtos que possam sofrer, posteriormente, outros tipos de processamento. Com o investimento maciço nas tecnologias de computação digital e no desenvolvimento de novos algoritmos para manipular sinais bidimensionais, a área de processamento digital de imagens vem ganhando destaque nas últimas décadas permitindo explorar um número considerável de novas aplicações.

Dessa forma, a tecnologia de processamento digital de imagens vem conquistando domínios nas mais diversas áreas, como por exemplo: transmissão digital de sinais de televisão, análise de imagens biomédicas, técnicas de tomografia computadorizada, dentre outras.

Umas das principais utilizações das técnicas de processamento digital de imagens foi melhorar ilustrações de jornais que eram enviadas por cabo submarino entre Londres e Nova York por volta de 1920 (Gonzalez and Woods, 1992), evidenciando assim o principal interesse da utilização dessa técnica: a necessidade de melhorar a qualidade da informação para interpretação humana.

Em meados dos anos 60, as técnicas de processamento digital de imagens evoluíram bastante com o advento de computadores digitais e com o programa espacial americano. As imagens da lua transmitidas por sondas foram processadas e corrigidas com as técnicas, pois a má qualidade dos equipamentos utilizados na época causou distorções nas imagens. Depois de comprovada a eficiência da utilização das técnicas, essas serviram como base para reestruturar imagens de outros programas espaciais.

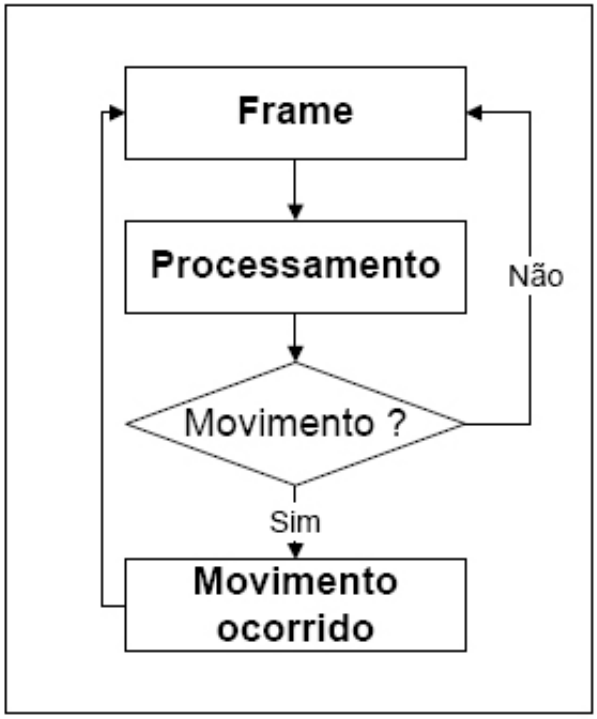
2.3 Técnicas de Processamento de Imagens

2.3.1 Detecção de Movimento

A detecção de movimento mostrado no fluxograma da figura 2.2 visa a diferenciar, em uma seqüência de vídeo, as imagens dos objetos dinâmicos dos estáticos. Essa técnica é um passo essencial em diversos problemas de visão computacional, especialmente no campo de segurança e vigilância eletrônica. Quanto maior a precisão com que os objetos em movimento forem extraídos, menor serão os ruídos e falsos positivos, ou seja, detectar somente o objeto de interesse e descartar os falsos positivos (ração do animal e fezes por exemplo), levando a um menor custo computacional nas etapas seguintes.

Técnicas de subtração ou segmentação de fundo e detecção de textura podem ser utilizadas para resolver alguns dos problemas de detecção de movimento. Essas técnicas vêm sendo estudadas há mais de 25 anos e englobam diversas áreas de interesse humano como: sistemas de vigilância (Haritaoglu *et al*, 2000), captura de movimento 3D, arte digital (Levin, 2004), reconhecimento de gestos, estimativa de pose humana e monitoramento em geral (McFarlane and Schopfield, 1995).

Atualmente, não existe uma abordagem definitiva para resolver o problema de detecção de movimento de forma genérica. As soluções existentes são para resolver esse problema em condições específicas relacionadas com a aplicação que se deseja criar (Sminchiescu and Telea, 2002). Toda esta dificuldade é devido a mudanças na iluminação (posição e intensidade), sombras, camuflagem, superfícies espelhadas, mudanças na movimentação (oscilação da câmera e objetos de alta freqüência), mudanças na geometria do fundo e, o mais importante de todos, ser em tempo real o que não acontece se por exemplo o programa tem como parâmetro de entrada um vídeo.



2.2. Fluxograma da detecção do movimento

2.3.2 Segmentação

Em quase todos os casos, o primeiro passo a ser dado em rastreamento de animais é fazer a separação de área de interesse do plano de fundo, ou fazer a detecção de algum tipo de movimento. Isso significa detectar regiões que contém os objetos de interesse.

O objetivo das técnicas de segmentação é dividir a imagem em suas diversas partes constituintes ou segmentos (objetos e regiões). O nível ou quantidade de divisões aplicadas na imagem varia conforme a aplicação, e em geral é realizada até atingir um nível de separação suficiente entre os objetos de interesse da cena analisada (Gonzalez and Woods, 1992).

Normalmente, para imagens monocromáticas as propriedades de descontinuidade e similaridade dos *pixels* são utilizadas para realizar a segmentação. A descontinuidade permite identificar e separar pontos isolados e bordas da imagem que possuem uma mudança abrupta dos níveis de cinza. Já a similaridade é a propriedade na qual se baseiam as técnicas de crescimento de regiões, limiarização, divisão e fusão de

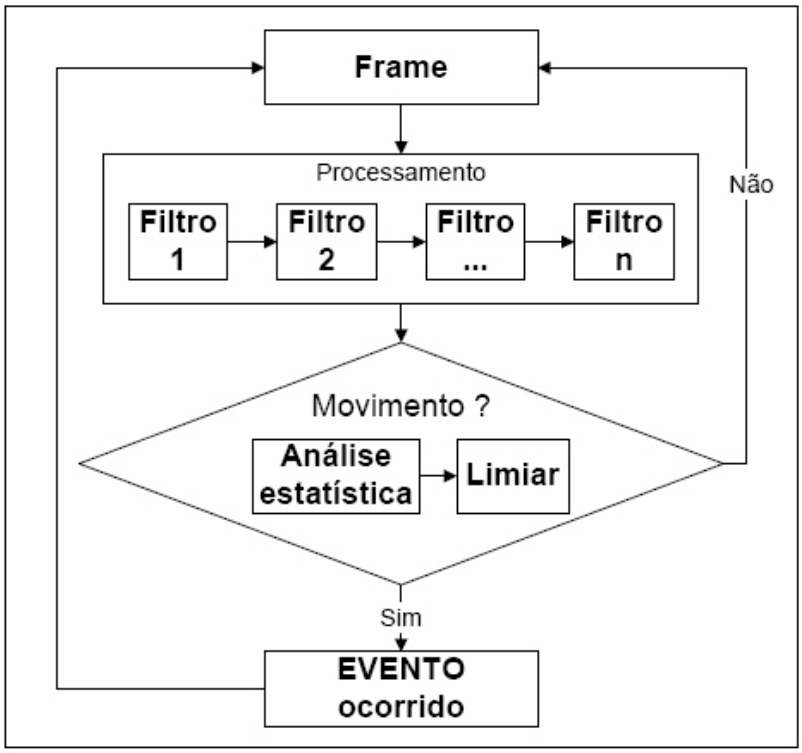
regiões. Em imagens coloridas costuma-se utilizar a similaridade entre os histogramas em cores para efetuar a segmentação (Pérez *et al*. , 2002).

2.3.2.1 Segmentação por subtração de imagens

O movimento é o fator principal para que os seres humanos e os animais possam distinguir um objeto de interesse em relação ao seu plano de fundo. Considerando-se um objeto estacionário e uma seqüência de imagens obtidas sobre ele, ao comparar, por exemplo, duas imagens distintas e subseqüentes *pixel* a *pixel*, e fazendo a subtração dos seus valores, é possível chegar à conclusão se houve movimento ou não, já que os componentes estacionários são cancelados na subtração. Aplicando-se um limiar[2] "L" na imagem resultante da subtração como mostrado na figura 2.3, torna-se possível separar os *pixels* que representam o movimento realizado por algum componente da imagem (Gonzalez and Woods, 1992).

É importante ressaltar que um movimento pode ocorrer tanto no plano de fundo, se o mesmo possuir algo que possa se movimentar no decorrer do tempo, como também no objeto de estudo. A escolha do limiar favorece a segmentação do objeto em estudo, isso se o limiar escolhido for capaz de diferenciar o movimento feito por um objeto de plano de fundo ou o movimento feito pelo objeto de interesse.

[2] Limite, fronteira entre duas áreas.



2.3. Fluxograma que demonstra a segmentação e o limiar.

2.3.2.2 Segmentação por subtração de background

A segmentação por subtração de background foi o tipo de segmentação escolhida para ser utilizada nesse trabalho, que seria capturar uma imagem apenas do plano de fundo, para ser usada como referência. Obtendo as imagens subseqüentes, procede-se à subtração dos valores dos *pixels* da imagem corrente - que contém o seu objeto de interesse a ser rastreado - dos valores dos *pixels* da imagem do plano de fundo (imagem de referência) (Pavim, 2002). Considerando que o dispositivo de captura seja estacionário, e o plano de fundo imóvel e que não sofra variações ao longo do tempo, o resultado é uma imagem que contém apenas o objeto de interesse.

Capítulo 3 – Técnicas de Estudo do Comportamento de Animais Experimentais

Alguns tipos de animais são utilizados para pesquisa com o objetivo de investigar os efeitos dos novos fármacos. É importante observar que, às vezes, um determinado tipo de comportamento em animais experimentais de laboratório pode ser de caráter específico, estando esta relacionada à exploração de um novo ambiente ou a uma atividade específica. Para se ter uma constatação do ocorrido existem alguns testes para a avaliação da atividade motora a partir do uso de novos fármacos.

Atualmente, na literatura, vários fármacos estão sendo estudados e testados pelos pesquisadores, porém foi dada uma ênfase maior em três: drogas ansiolíticas, o álcool e drogas que alteram o processo de apredizagem-memória (Graeff e Guimarães, 1999).

No entanto, a visão computacional nos últimos anos vem tentando ajudar os pesquisadores a solucionarem estes procedimentos de análise comportamental de testes de novos fármacos em animais experimentais de laboratório. Com isso, os pesquisadores não são obrigados a ficar um longo período de tempo observando determinados comportamentos específicos e poderão fornecer diagnósticos precisos a respeito de determinados fármacos em testes.

3.1 Técnicas utilizadas para análise de comportamento

Existem várias técnicas para se analisar o comportamento de animais experimentais de laboratório sob o efeito de cada droga distinta. Nesse item serão mostrados alguns tipos de técnicas utilizadas pelos cientistas, porém o teste *open field,* ou "campo aberto" será explorado com maior ênfase neste trabalho, devido a sua escolha para a pesquisa.

3.1.1 Teste do labirinto em cruz elevado

A figura 3.1 mostra o labirinto em cruz elevado, cujo fundamento baseia-se no conhecimento de que camundongos evitam locais abertos e elevados. Quando eles estão confinados, demonstram certo sinal de medo que pode ser identificado por alguns sintomas como: congelamento, defecação, micção e aumento do nível plasmático do hormônio de estresse (Graeff e Guimarães, 1999). O camundongo irá explorar todas as

partes do labirinto, ou seja, os braços abertos e fechados (são as partes abertas e fechadas do labirinto), porém espera-se que ele permaneça com mais freqüências nos braços fechados. Uma maior intensidade de "ansiedade" equivale à menor preferência por braços abertos (Morato e Brandão, 1997).

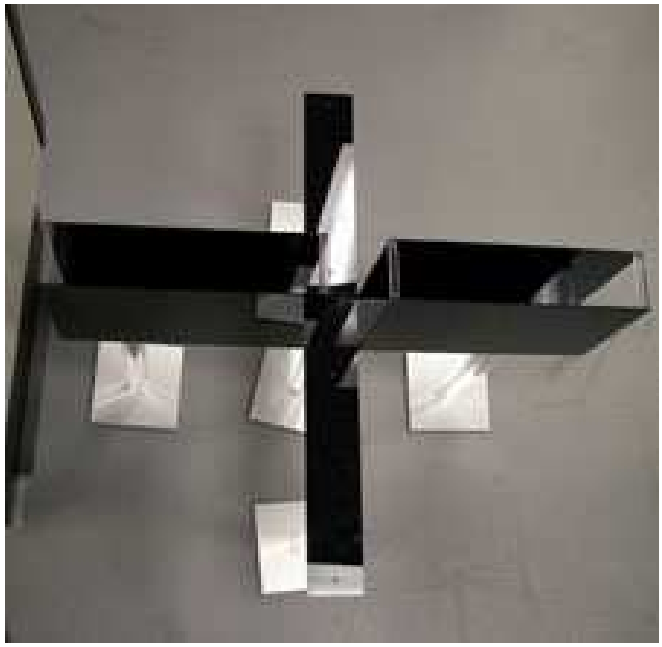


Figura 3.1. Labirinto em cruz elevado. (Figura extraída da página: http://www.insightltda.com.br – Página visitada no dia 02/09/2008).

3.1.2 Teste de esconder as esferas

No teste de esconder as esferas mostrado na figura 3.2 se verificam diversas alterações comportamentais de drogas que possam apresentar um perfil ansiolítico. Isso é caracterizado pelo aumento ou diminuição do comportamento de esconder as esferas de vidro, que são jogadas, ao acaso, na caixa coberta de serragem. A ansiedade é caracterizada pelo aumento das esferas recobertas (Treit *et al*., 1981 ; Broekkamp *et al*., 1986), já que se supõe representar para o animal o desconhecido, ou seja, um objeto estranho cuja a presença pode representar uma ameaça (Nogueira, 1997). A administração de drogas ansiolíticas reduz esse comportamento de recobrir as esferas de vidro.

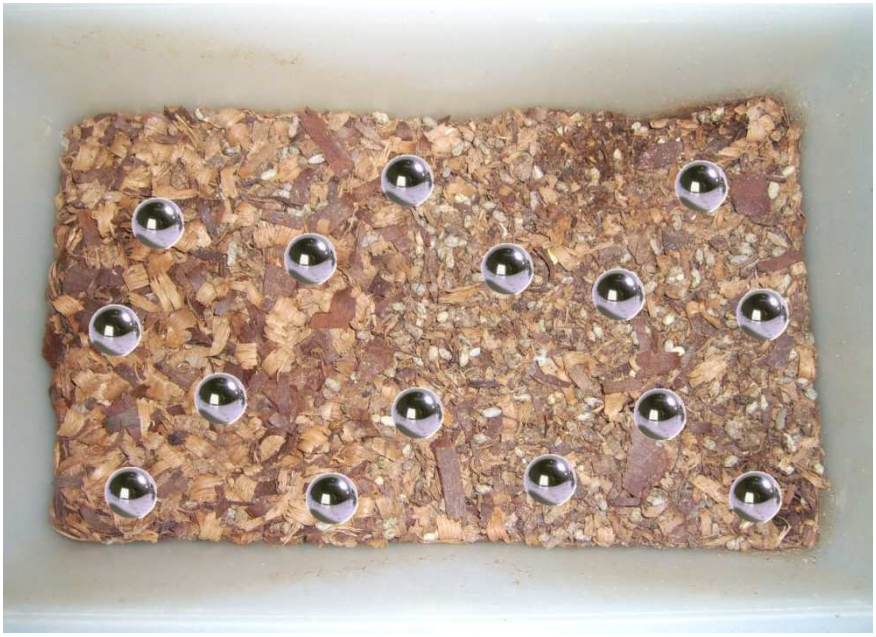


Figura 3.2. Teste de esconder as esferas.

3.1.3 Teste da placa perfurada

O labirinto da placa perfurada mostrado na figura 3.3 tem como objetivo o teste de drogas ansiolíticas e se baseia na observação de que atividades de mergulho dos animais são inversamente proporcionais ao estado de ansiedade dos mesmos. Outro ponto observado é a avaliação da atividade motora e exploratória direcionada através do número de cruzamentos entre os quadros delimitados na placa perfurada, do numero de levantar para o mesmo período de tempo, além do tempo de imobilidade (Boisser, 1976).

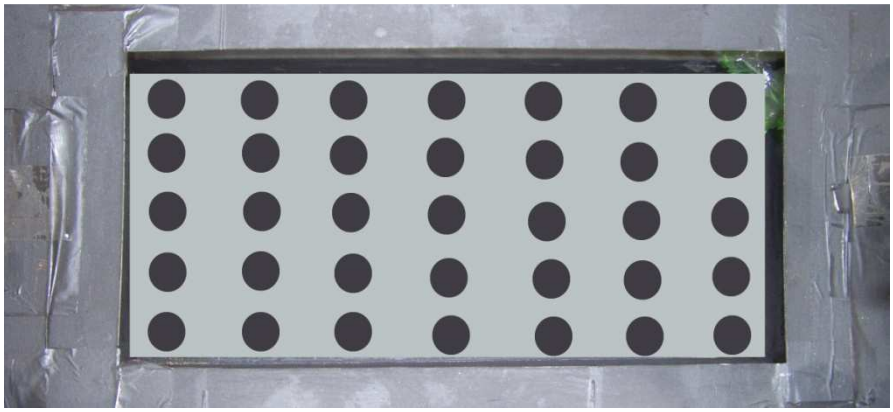


Figura 3.3. Teste da placa perfurada.

3.1.4 Teste do eixo giratório

A fundamentação desse teste está relacionada com a coordenação motora, especialmente nas drogas depressoras como o álcool. Sua avaliação é útil na pesquisa de drogas que possam potencialmente reduzir os efeitos prejudiciais do álcool, por exemplo. Há modelos em que se analisam as baixas velocidades de rotação, velocidades constantes e altas velocidades, mostrado na figura 3.4.

Figura 3.4. Teste do eixo giratório (Figura extraída da página: http://www.insightltda.com.br – Página visitada no dia 02/09/2008)

3.1.5 Teste do campo aberto (Open-Field)

O teste do campo aberto já vem sendo utilizado há muito tempo, com o intuito de avaliar o efeito dos fármacos em animais experimentais de laboratório. Os modelos mais utilizados no teste de campo aberto são as atividades locomotoras e alterações na expressão da emocionalidade.

O teste do campo aberto ou *open Field* mostrado na figura 3.5 é amplamente utilizado para quantificar movimentos locomotores e de exploração dos animais. Os

movimentos locomotores são os deslocamentos de um ponto a outro da arena. Os movimentos de exploração ou não locomotores são aqueles que o animal pode realizar sem a necessidade de deslocamento, como por exemplo, elevação vertical, cheirar o ambiente e autolimpeza.

Em experimentos com roedores, esses comportamentos são essenciais para compreender o efeito de diferentes drogas psicoestimulantes e ansiolíticas (Prut and Belzung, 2003).

Neste trabalho foi escolhido o teste de campo aberto para analisar o comportamento de animais experimentais, com o intuito de fazer um monitoramento e ter como resultado final informações relacionadas a tempo e posição, ou seja, o sistema gera um "arquivo texto" (Evento, Tempo FX e FY), que em seguida é usado em uma análise estatística.

Figura 3.5. Ambiente para campo aberto

Capítulo 4 – Especificações do Sistema

4.1 Introdução

O processo de desenvolvimento de *software* é composto de um conjunto de atividades que envolvem métodos, ferramentas e procedimentos, com o intuito de produzir programas que atendam aos requisitos especificados pelos usuários (clientes) (Mayrhauser, 1990).

Um projeto de *software* que não teve uma boa especificação, certamente irá desapontar o usuário e acarretar problemas aos programadores que irão desenvolvê-lo, que terão de modificá-lo para se adequar às necessidades dos usuários. De acordo com Castro 1995, a especificação de requisitos serve como um padrão para testar se as fases de projeto e implementação do processo de desenvolvimento de *software* estão corretas.

A fase de análise de requisitos é fundamental para o sucesso do processo de desenvolvimento do *software*. Nesta fase, o projetista (engenheiro de requisitos) especifica as funções e desempenho do *software*, indica a interface do *software* com outros sistemas e estabelece as restrições de projeto do *software* (Pressman, 1994).

Inicialmente, será feita uma distinção entre os termos requisito e especificação. Um modo prático de definir requisito é utilizar a definição do glossário de engenharia de *software* (IEEE Std. 610.12, 1990) e do dicionário Aurélio (Aurélio, 1986). O glossário de engenharia de *software* do IEEE (*IEEE Std. 610.12, 1990*) define requisito como:

1. Uma condição ou capacidade necessitada por um usuário para resolver um problema ou alcançar um objetivo.

2. Uma condição ou capacidade que deve ser satisfeita ou possuída por um sistema ou componente do sistema para satisfazer um contrato, um padrão ou uma especificação.

3. Uma representação documentada de uma condição ou capacidade como em (1) ou (2).

Segundo o dicionário Aurélio (Aurélio, 1986), o termo requisito pode ser definido como "condição necessária para a obtenção de certo objetivo, ou para o preenchimento de certo fim". Já o termo especificação é "uma descrição rigorosa e minuciosa das características que um material, uma obra, ou um serviço deverão apresentar".

De acordo com o IEEE (IEEE Std.1984), o processo de aquisição, refinamento e verificação das necessidades do usuário é chamado de engenharia de requisitos (E.R.). O objetivo da E.R. é sistematizar o processo de definição dos requisitos, obtendo uma especificação correta e completa dos requisitos. A compreensão da engenharia de *software* como uma disciplina que procura tornar mais eficaz o *software* e o processo utilizado para produzi-lo é fundamental para entender o papel da E.R.. Boehm, (1989) define a E.R. como uma disciplina cujo objetivo é desenvolver uma especificação completa, consistente e não ambígua, servindo de base para um acordo entre todas as partes envolvidas e descrevendo o quê o produto de *software* irá fazer ou executar, mas não como ele será feito.

4.2 Definição do Problema

Necessidade de criação de um *software* que utilize visão computacional com o objetivo de auxiliar os pesquisadores no monitoramento de animais experimentais e fornecer dados precisos para uma análise.

4.3 Hardware

4.3.1 Computador

É considerado uma peça fundamental do sistema, pois é ele quem coordena todas as operações desempenhadas, desde o acionamento do *hardware* de aquisição de imagem, as operações realizadas pelo *software* de processamento de imagem, o armazenamento dos dados, até a visualização dos resultados em um *display*. Para essa aplicação foi utilizado um computador com as seguintes características: *Processador IV, Centrino Duo 1.73, 1536 MB DDR2, Hd 120 GB*. O sistema operacional utilizado nesta pesquisa foi o *Windows XP Professional*.

4.3.2 WebCam

Para o desenvolvimento do trabalho e para a escolha do melhor equipamento a ser utilizado, foram realizadas diversas pesquisas no que diz respeito aos tipos de câmeras encontradas atualmente no mercado, suas principais características, aplicações, preço e tamanho.

A proposta inicial da utilização das *webcams* estava voltada, principalmente, para ambientes domésticos e de entretenimento, porém essas câmeras já estão alcançando espaços no mercado em diversas outras aplicações, como as indústrias, por exemplo.

Trata-se de câmeras que permitem a transmissão de voz e vídeo tendo como meio principal a *internet*, são câmeras que possuem boas taxas de transferência de dados, já existindo modelos com alta resolução de imagens (*megapixel*). As *webcams,* de modo geral, possuem tamanho pequeno, baixo custo e pouco peso.

Outro aspecto que precisa ser ressaltado é a importância do sensor, que influencia diretamente na qualidade do vídeo que a *webcam* é capaz de reproduzir.

Neste trabalho foi utilizado uma Webcam Leadership 1.3 MegaPixel, interface USB 2.0, Sensor 300K CMOS, Freqüência de amostragem 30 quadros por segundo e um foco 30mm.

Podemos citar dois tipos de sensores: *CMOS* (*Complementare Metal Oxide Semiconductor*) considerado ideal para obter melhores imagens em situações de luz escassa, e o *CCD* (*Charge-Copled Device*), que oferece como característica principal um aparência mais limpa para a imagem.

As *webcams* são atualmente bem utilizadas em vídeo conferência, sistema de monitoramento, vigilância de ambientes, dentre outros.

4.4 Requisitos

4.4.1 Requisitos Funcionais

Os requisitos funcionais podem ser definidos como sendo um conjunto de serviços que o sistema deve fornecer. Em sistemas de software existem diferentes níveis de abstração em que esses serviços podem ser desenvolvidos. Aqui, serão considerados

apenas os mais básicos. (Bérard, 1999) identifica três serviços que os sistemas de interação usuário-computador baseados em visão computacional devem fornecer, estes são: detecção, identificação e rastreamento.

Detecção: A detecção determina a presença ou ausência de uma determinada classe de objetos na imagem. Tais classes de objetos poderiam ser partes do rato, rabo e cabeça por exemplo. Tendo como referência a imagem inteira, o processo de detecção deve ser capaz de detectar na imagem a classe de objetos que se está procurando. Uma forma de facilitar o processo de detecção é limitar o número de objetos que podem estar presentes na cena em um determinado momento. As técnicas de detecção mais conhecidas são as baseadas em cor e movimento (Bérard, 1999).

Identificação: A identificação determina qual objeto, dentre um conjunto conhecido de objetos, está presente na cena. Perante a presença de objetos compostos, por exemplo, o rato com o rabo, a identificação deve permitir determinar partes desses objetos, tais como o rato. Para o nosso caso, o processo de detecção encontra o centro do rato (Bérard, 1999).

Rastreamento: Os objetos de interesse podem não permanecer no mesmo lugar ao longo do tempo, devido a essa característica o processo de rastreamento utiliza as informações dos dois processos anteriores para manter o foco nos objetos de interesse. Em nosso trabalho o rastreamento se refere à captura das posições do centro do rato (Bérard, 1999).

4.4.2 Requisitos Não Funcionais

É indispensável definir alguns requisitos não funcionais que estabeleçam a qualidade mínima com que os serviços devem ser implementados. Porque muito desses sistemas de interação podem demorar certo tempo (horas) para cumprir com todos os requisitos funcionais. Os requisitos não funcionais considerados são: latência, resolução e estabilidade.

Latência: A latência é o tempo transcorrido entre a ação do usuário e a resposta do sistema. A latência é uma característica inerente a todo sistema, não existem sistemas

sem latência. Essa latência deve possuir um valor aceitável de acordo com a tarefa que o sistema deve desempenhar. A interação deve conseguir uma latência o menos perceptível para o usuário (Bérard, 1999).

Resolução: A resolução espacial (número de *pixels* na imagem) deve permitir uma representação adequada do ambiente capturado. Idealmente, esse número deveria ser igual ao número de *pixels* existentes nos monitores, mas os sistemas de captura que utilizamos no nosso trabalho como *WebCams* não possuem ainda essa resolução e estão limitados a resolução menores (Bérard, 1999).

Estabilidade: A estabilidade dos sistemas refere-se às flutuações significativas nos valores capturados (gestos ou posições do rato). Por exemplo, um sistema pode ser considerado estável se ante um padrão imóvel os valores capturados não mudam significativamente. As possíveis causas de instabilidade são principalmente as flutuações nas fontes de iluminação e o ruído inerente aos dispositivos de captura (Bérard, 1999).

Capitulo 5 – Desenvolvimento do Sistema

Para o desenvolvimento deste sistema foram utilizadas algumas ferramentas de modelagem, ferramentas de desenvolvimento, bibliotecas e *toolbox* que serão discutidas nesse capítulo.

5.1 Modelagem do Sistema (UML)

Para se desenvolver novos sistemas que fazem uso da orientação de objetos nas suas várias fases como análise de requisitos, análise de sistemas e *design*, existe o grande problema da falta de um padrão que englobe quaisquer tipos de aplicações.

Existem diversas simbologias, conceitos próprios, gráficos e terminologias que resultam em desordens, principalmente quando o objetivo é utilizar “orientação a objetos” para se projetar sistemas cada vez mais eficazes.

Com a criação da *UML* (*Unified Modeling Language*), muitos desenvolvedores viram o problema da falta de padronização chegar ao fim com a proposta de unificação da *UML*.

No entanto, a *UML* foi além, tornou-se muito mais que uma padronização, desencadeou o desenvolvimento de novos conceitos normalmente não usados. Devido às suas vantagens, não basta apenas aprender a simbologia da *UML*, é necessário compreender seu significado e aprender a modelar no paradigma orientado a objetos.

Grady Booch, *James Rumbaugh*, e *Ivar Jacobson* conhecidos como "os três amigos", são os mentores da *UML*, eles possuem um vasto conhecimento no que diz respeito à área de modelagem, já que são os desenvolvedores das três mais conceituadas metodologias de modelagem orientadas a objetos, e o que havia de melhor nos três métodos deu origem ao que conhecemos como *UML*, com um adicional de novos conceitos e visões da linguagem.

5.1.1 – Diagrama de Classes do Sistema

O diagrama de classes é responsável por demonstrar a estrutura estática das classes de um sistema. Classes podem se relacionar através de diversas maneiras:

associação (conectadas entre si), dependência (uma classe depende ou usa outra classe), especialização (uma classe é uma especialização de outra classe), ou em pacotes (classes agrupadas por características similares). No sistema de monitoramento atual foi utilizada a relação de dependência, ou seja, onde uma classe depende da outra. Alguns relacionamentos são mostrados na figura 5.1 juntamente com as suas estruturas internas, que são os atributos e operações. O diagrama de classes é considerado estático já que a estrutura descrita é sempre válida em qualquer ponto do ciclo de vida do sistema.

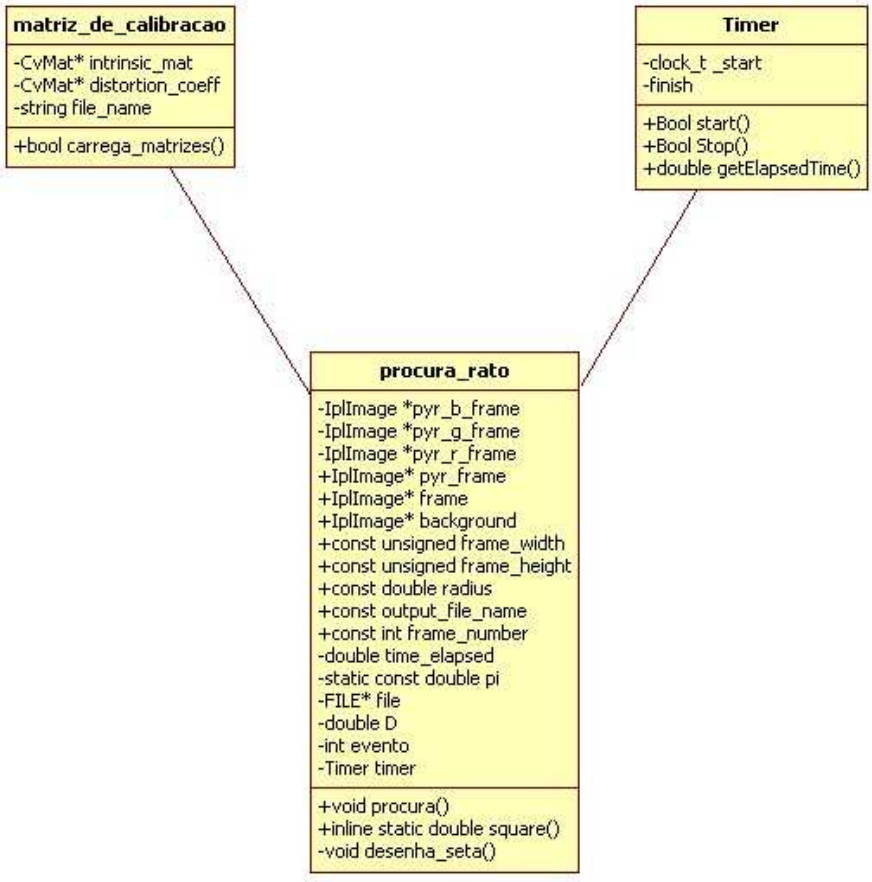


Figura 5.1. Diagrama de Classes do sistema de monitoramento de animais experimentais.

5.1.2 – Diagrama de Seqüência

O diagrama de seqüência tem como finalidade mostrar a colaboração dinâmica entre os vários objetos de um sistema no decorrer do tempo. O aspecto mais importante desse diagrama é que através dele a seqüência das mensagens entre os objetos torna-se

mais visível, facilitando a visualização do andamento do processo, mostrando assim o que ocorrerá em um determinado ponto específico da execução do sistema.

O diagrama de seqüência consiste em um número de objetos mostrados em linhas verticais que representam exatamente o tempo de vida de um objeto, e em linhas horizontais que representam as mensagens trocadas pelos objetos, essas mensagens são simbolizadas por setas horizontais. Podem existir também mensagens enviadas para o mesmo objeto, representando uma iteração.

No eixo horizontal são mostrados os objetos que estão envolvidos na seqüência, onde cada um é representado por um retângulo e uma linha vertical pontilhada chamada linha de vida do objeto, que indica a execução do objeto durante a seqüência. As mensagens podem possuir também números seqüenciais, eles são utilizados para tornar mais explícita a seqüência no diagrama e facilitar sua leitura.

Na figura 5.2 será mostrados quais são os passos utilizados para o monitoramento de animais experimentais.

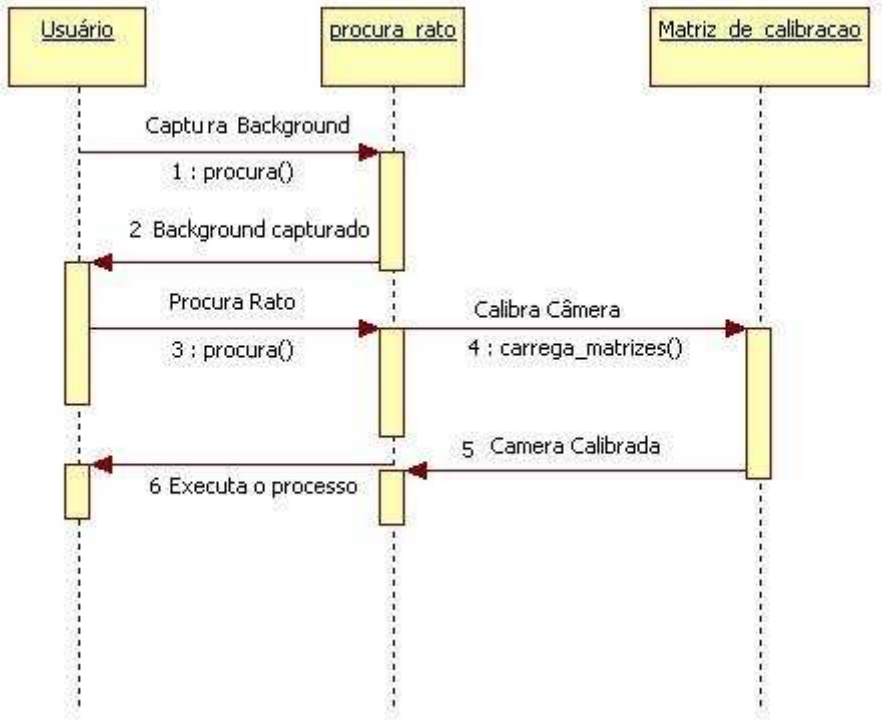


Figura 5.2. Diagrama de seqüência que demonstra o processo dinâmico do monitoramento animal.

5.2 OpenCV

Para o desenvolvimento do algoritmo de processamento de imagens foi escolhida uma biblioteca que atendesse alguns requisitos como: ser *Open Source*, ser desenvolvida na linguagem C e possuir procedimentos e funções relacionados a processamento de imagens.

A biblioteca *OpenCV* (*Intel Open Source Computer Vision Library*) é uma coleção de funções C e classes C++ que implementam algoritmos importantes relacionados ao processamento de imagens.

Essa biblioteca está estruturada na linguagem C e foi desenvolvida inicialmente pela *Intel*, que liberou o código para sua utilização. Posteriormente essa decisão foi alterada, já que foram feitas várias modificações na biblioteca, como a criação de vários módulos novos, que tinham como principal intuito a otimização de sua utilização.

O *OpenCv* está dividido em vários pacotes com o intuito de fornecer condições para que programadores possam de uma maneira rápida e efetiva operar com o mundo do vídeo e do processamento de imagem. A Biblioteca divide-se nas seguintes áreas:

A primeira área é o pacote núcleo da biblioteca (CV) e está dividida nos seguintes itens:

a) Processamento de imagem;
b) Análise de movimento;
c) Análise estrutural;
d) Reconhecimento de padrões;
e) Calibração de câmara e reconhecimento 3D;

Outra área é o pacote com estruturas de dados importantes (CXCORE) para o desenvolvimento de programas (fornece um conjunto de métodos de desenho), e está dividido em dois itens.

a) Operações em *arrays*
b) Estruturas dinâmicas

Em seguida o pacote com funcionalidades de *interface (HighGUI)*. Ferramentas para criação de janelas, *captura* de imagem, como mostrado nos itens seguintes.

a) GUI simples

b) Vídeo I/O
c) Imagens I/O

5.3 *Toolbox* do *Matlab* para calibração de câmera[3]

Existem vários modelos matemáticos para fazer a calibração da câmera, o toolbox do Matlab foi escolhido por ser de fácil manipulação e ter uma boa precisão como saída.

5.3.1 Introdução

Este item tem como objetivo descrever as principais características do *Toolbox* de calibração de câmeras do *Matlab*, com o intuito de facilitar sua utilização. Serão descritos os métodos utilizados, a estimação de parâmetros de calibração e algumas funções que foram implementadas no *Toolbox* com seus respectivos parâmetros de entrada e saída.

O *Toolbox* de calibração de câmeras do *Matlab*, está disponível na *web* (Prista, 2003). Com esta ferramenta é possível ler imagens de calibração, extrair os principais pontos de calibração (vértices dos quadros existentes na imagem), calibrar a câmara, mostrar os resultados obtidos, controlar precisões, adicionar ou remover imagens, corrigir imagens que possuem algum tipo de distorção e alterar o modelo intrínseco da câmara escolhendo quais parâmetros otimizar.

5.3.2 Classificação do Método

O método de calibração implementado por esta ferramenta é considerado extremamente preciso e utiliza quatro passos para calibrar uma câmera:

a) Determina uma solução analítica (*closed-form solution*), ou seja, uma solução que permita obter uma aproximação inicial dos parâmetros intrínseco-extrínsecos da

[3] Ver http://www.vision.caltech.edu/bouguetj/calib_doc/ (Visitado no dia 05/05/2008)

câmera utilizando o modelo de *Pin Hole* (câmera sem lente) e pontos de calibração não co-planares.

b) Utiliza estimação não-linear dos parâmetros através do método dos "Mínimos Quadrados" com objetivo de diminuir os resíduos entre o modelo e as *N* observações (erro de re-projeção), como distorção radial e tangencial da lente.

c) Corrige os pontos extraídos na calibração da imagem devido a distorção do padrão de calibração, causados pela projeção perspectiva.

d) Corrige as coordenadas de imagem distorcida a partir do modelo inverso empírico que compensa a distorção radial e tangencial da lente.

Os passos utilizados pelo método de calibração estão explicitados na figura 5.3.

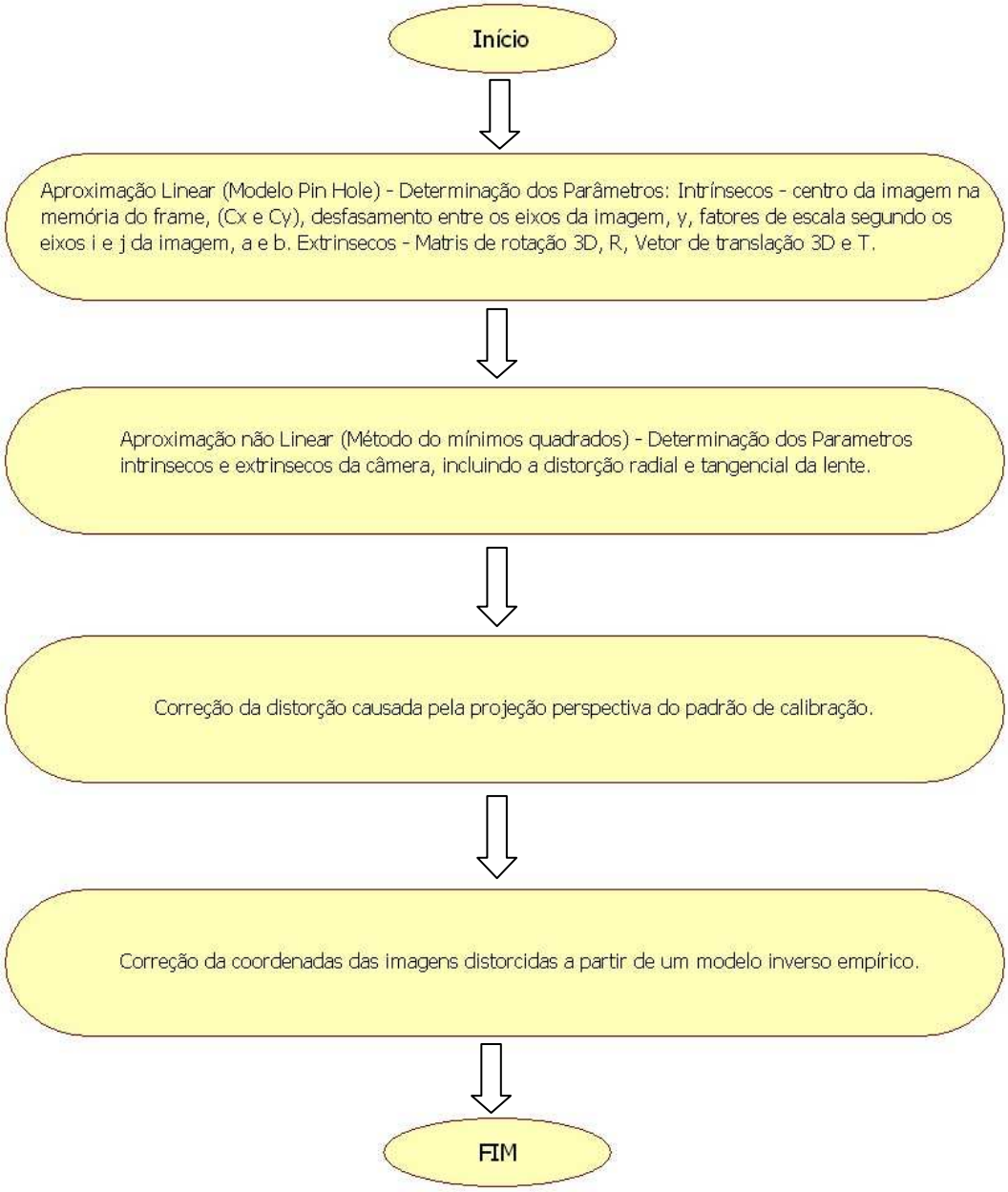


Figura 5.3. Esquema do método de calibração de câmeras para Matlab utilizado no Toolbox.

O método de calibração de câmeras do *Matlab* utilizado pelo *Toolbox* faz uso de um modelo de câmera que possui dois parâmetros a estimar:

Parâmetros Extrínsecos: esses parâmetros são utilizados para transformar coordenadas 3D do "sistema mundo" em coordenadas 3D do "sistema câmera" para cada posição e/ou orientação da câmera de calibração. Assim podemos dizer que existem os seguintes parâmetros extrínsecos: a matriz de rotação 3D Rc (3x3) e as três componentes do vetor de rotação 3D omc (3x1) (Coelho e Tavares, 2003).

Parâmetros Intrínsecos: são utilizados, segundo Coelho e Tavares (2003), para transformar coordenas 3D do sistema câmera em coordenadas 2D do sistema de memória *frame*. No modelo são apresentados 5 (cinco) parâmetros intrínsecos:

a) **Distância Focal**: é a distância em *pixels* entre o centro de projeção e o plano de imagem (segundo dois eixos, *i* e *f*) e é guardada no vetor *fc* (2x1) (fc(1) = fc_x e fc(2) = fc_y). A distância focal efetiva (*f*) pode ser obtida de: $fc_x = f.s_x/d_x$ ou $fc = f/d_y$, onde d_x e d_y são as distâncias entre centros dos elementos sensores vizinhos de acordo com as direções X e Y, e s_x é um fator de incerteza horizontal devido a erros de sincronização.
b) **Centro óptico da imagem**: são as coordenadas em *pixels*, do centro ótico da imagem na memória *frame* e são armazenadas no vetor **cc(2x1) (cc(1) = ccx e cc(2) = ccy);**
c) **Coeficiente de Desfasamento:** é a representação do ângulo entre os eixos X e Y da imagem na memória *frame* armazenado na variável **alpha_c;**
d) **Distorções:** são os coeficientes de distorção da imagem (radial e tangencial) e são armazenados no vetor (5x1) kc(kc(1),kc(3) e kc(5)), que são os coeficientes de distorção radial de 2ª, 4ª e 6ª ordem respectivamente, e kc(2) e kc(4) são os coeficientes de distorção tangencial de 1ª e 2ª ordem respectivamente.

5.3.3 Software de calibração de câmera para Matlab

O *software* de calibração é compatível com as versões do *Matlab* 5x e 6x (somente até a versão 6.5), o mesmo foi desenvolvido para "rodar" nos sistemas operacionais *Microsoft Windows*, *Unix e Linux*, levando sempre em consideração a versão do *Matlab*. A implementação em C desta ferramenta também pode ser encontrada disponível no *OPENCV* que é distribuído pela *Intel*. Nas seções subseqüentes serão apresentados os principais parâmetros de entrada e saída de cada uma das funções deste *Toolbox*.

5.3.3.1 Sistema com uma única câmera

Nesta seção são apresentadas as funções que o *Toolbox* de calibração de câmeras possui para *Matlab*. Essas funções permitem a calibração completa de uma câmera, para isso é utilizado como plano de calibração, um padrão constituído por quadros. Assim, serão apresentadas as funções que vão permitir ler imagens de calibração, extrair os vértices dos quadros da imagem (que vão ser os pontos de calibração), calibrar a câmera, alterar o modelo intrínseco da câmera escolhendo os parâmetros que serão otimizados, apresentar os resultados, controlar precisões e corrigir imagens distorcidas (Brown 1966).

5.3.3.1.1 Leitura de Imagens de Calibração

Nesta seção é apresentada a ferramenta utilizada pelo *Toolbox* que permite fazer a leitura das imagens de calibração.

Para que as imagens sejam processadas é necessário informar um parâmetro de entrada, a fim de se obter um parâmetro de saída.

5.3.3.1.2 Parâmetro de entrada

ImageName1, Extension; Imagename2, Extension - Este parâmetro possui um padrão no seu formato, onde deve ser informado o nome das imagens de calibração seguido do respectivo formato (formatos possíveis: *ras, bmp, tif, pmg, jpg e ppms*) como mostrado na figura 5.4.

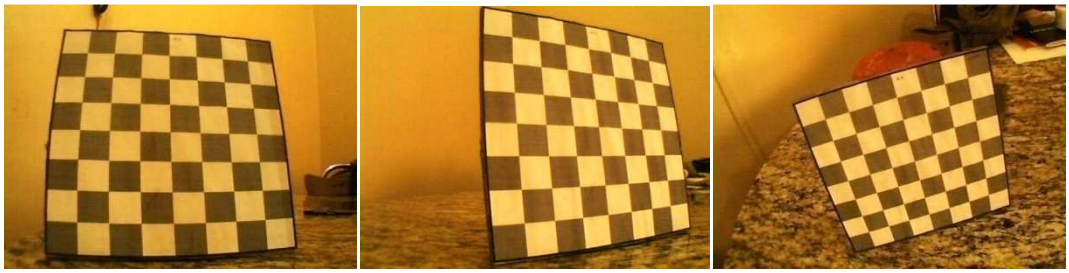


Figura 5.4. Exemplo de imagens de calibração lidas pelo Toolbox

5.3.3.1.3 Parâmetro de Saída

Complet Set of Images - É exibido como saída um conjunto completo das imagens lidas pelo *Toolbox* em formato de mosaico, como mostra a figura 5.5.

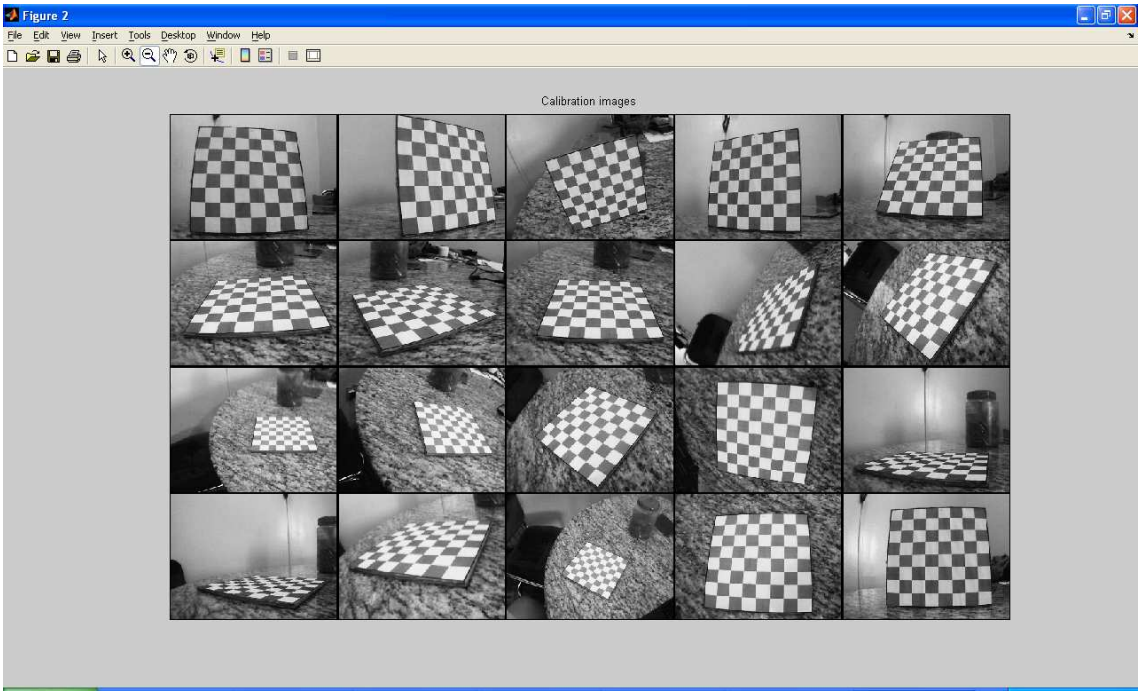


Figura 5.5. Conjunto de imagens exibidas em formato de mosaico

5.3.4 Extração dos Vértices

Nesta seção são apresentados os parâmetros de entrada e saída, e os procedimentos utilizados para extrair os pontos referentes aos vértices dos quadrados de uma grelha de calibração (Coelho e Tavares, 2003).

5.3.4.1 Parâmetros de Entrada

Etapa I: é necessário primeiro fornecer ao programa o número de imagens de calibração que serão processadas, o tamanho da janela de busca dos vértices dos quadros e o número de quadrados nas imagens de calibração.

a) ***Number of Images to Process -*** é o número de imagens que serão processadas para extração dos pontos de calibração. Se o parâmetro de entrada for nulo "[]", todas as imagens lidas serão processadas, do contrário, é preciso informar uma lista do subconjunto de índices das imagens que se pretende extrair os vértices (ex. [2 5 8 10 12].

b) ***Wintx e Winty -*** é o tamanho da janela de busca dos vértices nas direções X e Y, respectivamente. Quando os parâmetros de entrada são nulos "[]" para *Wintx e Winty*, o valor considerado para ambos é 5 (cinco), pois corresponde ao tamanho efetivo da janela de 11x11*pixels*, como mostrado na figura 5.6.

c) ***Number of Squares*** **–** permite utilizar o mecanismo automático de contagem dos quadrados presentes nas imagens de calibração (parâmetro de entrada nulo "[]"), ou informar manualmente o número.

```
Extraction of the grid corners on the images
Number(s) of image(s) to process ([] = all images) =
Window size for corner finder (wintx and winty):
wintx ([] = 5) =
winty ([] = 5) =
Window size = 11x11
Do you want to use the automatic square counting mechanism (0=[]=default)
  or do you always want to enter the number of squares manually (1,other)?
```

Figura 5.6. Parâmetros de entrada para extração dos vértices dos quadrados presentes na imagem de calibração (Figura retirada do Manual do toolbox de calibração de câmera do Matlab).

Etapa II: Escolha dos limites da grelha de calibração de onde vão se extrair os vértices dos quadrados do padrão retangular.

Após informar os parâmetros de entrada de acordo com a Etapa I, a primeira imagem de calibração é mostrada, e a partir dela são escolhidos quatro vértices extremos do padrão retangular de acordo com o seguinte procedimento: é necessário primeiro, pressionar sobre a imagem o local do primeiro vértice que corresponde ao ponto de origem da janela de referência associada à grelha; em seguida o próximo passo é fazer a escolha dos outros três pontos da grelha retangular, seguindo uma ordem circular independente do sentido, como mostrado na figura 5.7.

Etapa III: Informar ao programa o tamanho de cada quadrado presente na grelha de calibração. Onde dX e dY são os tamanhos de cada quadrado da grelha de calibração

nas direções X e Y, respectivamente. Se os parâmetros de entrada forem nulos "[]", o valor para ambos é de 30 (trinta) milímetros.

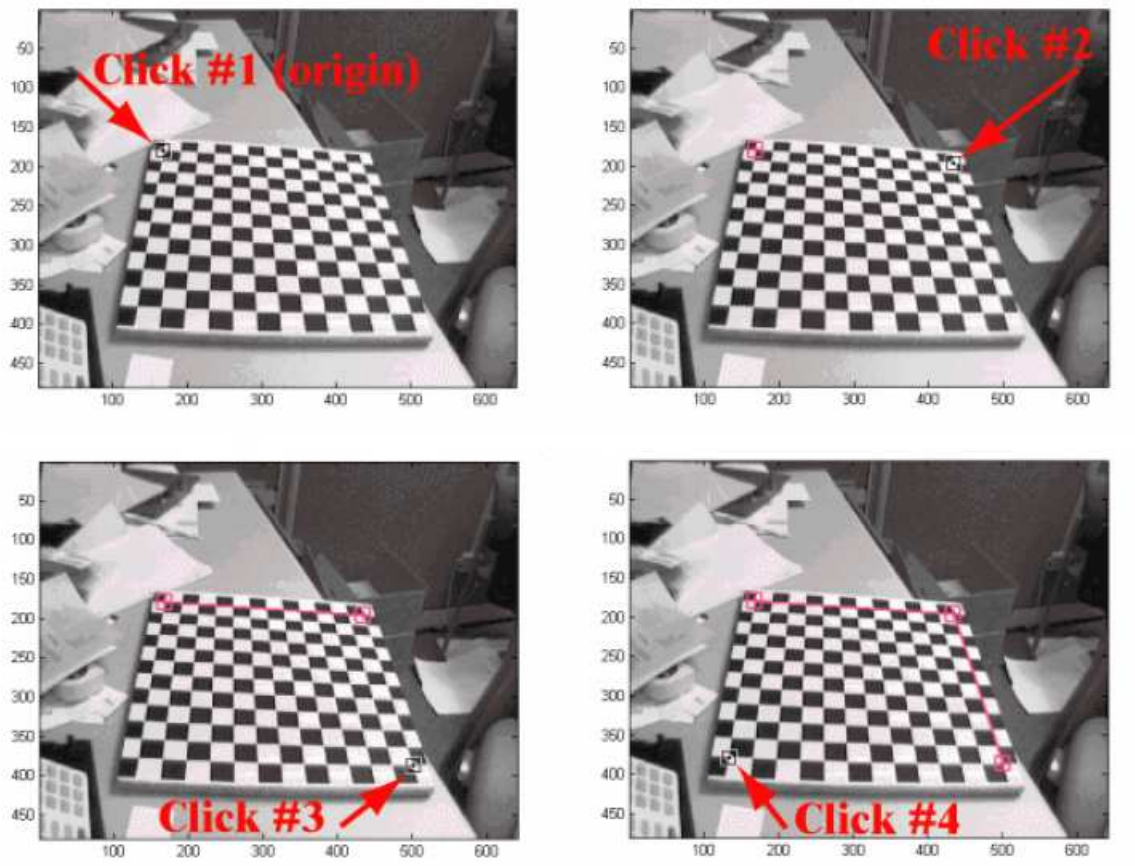


Figura 5.7. Procedimento para determinar os cantos da grelha de onde vão se extrair os vértices do padrão retangular (Figura retirada do Manual do toolbox de calibração de câmera do Matlab).

5.3.4.2 Parâmetros de Saída

Image with Predicted Grid Corners - o *software* mostra os vértices estimados dos quadrados da grelha de calibração, quando não há distorção (figura 5.8). Quando a imagem tem distorção elevada, a estimação destes vértices poderá ser fraca, ou seja, podem estar afastados dos vértices reais dos quadrados da imagem. Portanto, é possível introduzir um valor inicial para o coeficiente de distorção radial da lente, para "ajudar" o *software* a dispor dos vértices dos quadrados na imagem.

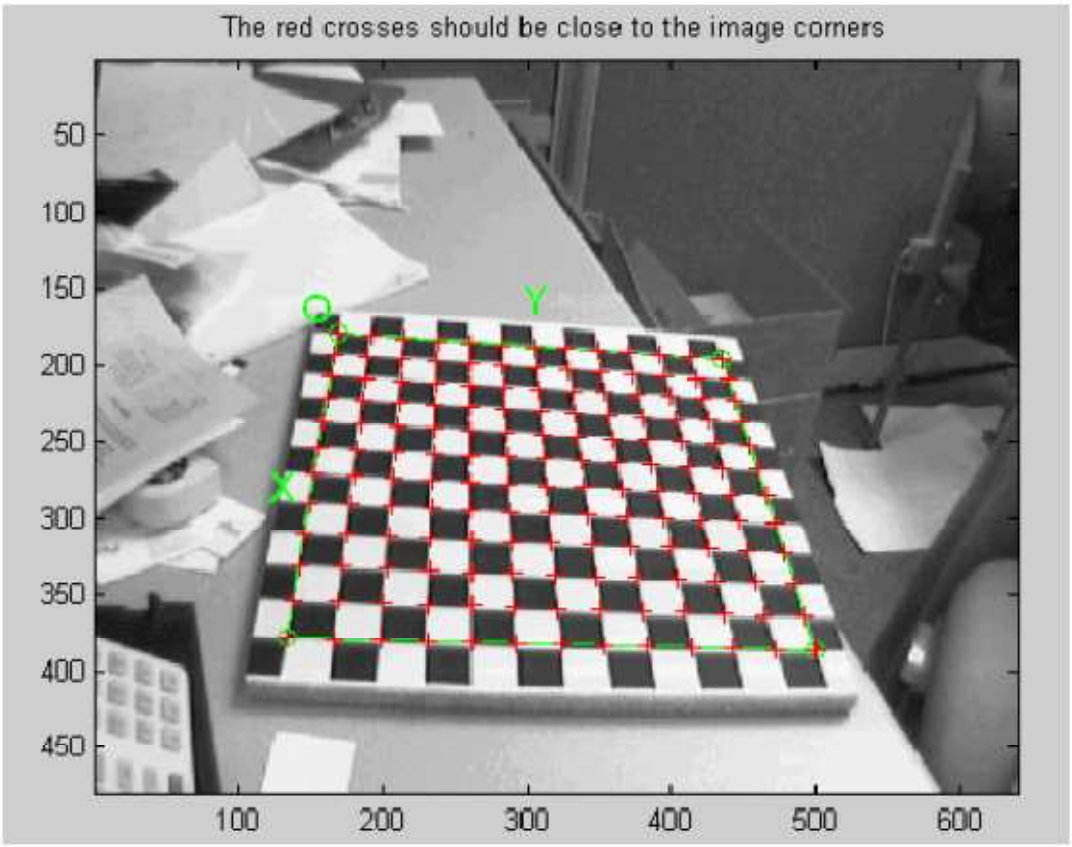


Figura 5.8. Exemplo de um conjunto de vértices estimados pelo Toolbox (vértices em vermelho) (Figura retirada do Manual do toolbox de calibração de câmera do Matlab).

5.3.5 Calibração de uma Câmera

Para realizar a calibração de uma câmera é necessário realizar duas fases: aproximação linear e otimização não-linear. Na fase de aproximação linear é calculada uma solução inicial (*closed-form solution*) para os parâmetros de calibração, não incluindo a distorção da lente. A fase de otimização não-linear tem a função de diminuir os erros de re-projeção total (método de mínimos quadros) de todos os parâmetros de calibração.

5.3.5.1 Parâmetros de Entrada

a) ***Image with Predicted Grid Corners*** – é a imagem que possui marcados os vértices estimados nos quadrados da grelha de calibração;
b) ***Aspect RatioOptmized*** **(parâmetro de entrada utilizado no fazer de otimização)–** opção para otimizar o parâmetro do modelo de calibração que indica

se o *pixel*, numa linha discretizada pelo *CCD,* é quadrado ou não. Esta opção esta ativa por padrão (est_aspect_ratio =1);

c) ***Principal Point Optimized*** **(parâmetro de entrada utilizado no fazer de otimização)** – opção para otimizar o centro da imagem na memória *frame*. Por *default* esta opção esta ativa (Center_optim = 1);

d) ***Skew not Optimized*** **(parâmetro de entrada utilizado no fazer de otimização) –** opção para otimizar o parâmetro do modelo de calibração que descreve o desfasamento entre os dois eixos da imagem. Esta opção não vem ativa por padrão (est_alpha = 0).

5.3.5.2 Parâmetros de Saída

a) ***Calibration Parameters after Initialization*** **–** parâmetros intrínsecos da câmera estimados, considerando o modelo *Pin Hole* para a câmera (aproximação inicial);

b) ***Calibration Parameters after Optimization*** **-** parâmetros intrínsecos da câmera estimados, considerando um modelo não-linear para a câmera (método de Mínimos Quadros). Os desvios padrão de cada um dos parâmetros estimados também são apresentados na figura 5.9.

```
Aspect ratio optimized (est_aspect_ratio = 1) -> both components of fc are estimated (DEFAULT).
Principal point optimized (center_optim=1) - (DEFAULT). To reject principal point, set center_optim=0
Skew not optimized (est_alpha=0) - (DEFAULT)
Distortion not fully estimated (defined by the variable est_dist):
     Sixth order distortion not estimated (est_dist(5)=0) - (DEFAULT) .
Initialization of the principal point at the center of the image.
Initialization of the image distortion to zero.
Initialization of the intrinsic parameters using the vanishing points of planar patterns.

Initialization of the intrinsic parameters - Number of images: 20


Calibration parameters after initialization:

Focal Length:          fc = [ 671.13759   680.77186 ]
Principal point:       cc = [ 319.50000   239.50000 ]
Skew:             alpha_c = [ 0.00000 ]   => angle of pixel = 90.00000 degrees
Distortion:            kc = [ 0.00000   0.00000   0.00000   0.00000   0.00000 ]

Main calibration optimization procedure - Number of images: 20
Gradient descent iterations: 1...2...3...4...5...6...7...8...9...10...11...done
Estimation of uncertainties...done


Calibration results after optimization (with uncertainties):

Focal Length:          fc = [ 661.67001   662.82858 ] ± [ 1.17913   1.26567 ]
Principal point:       cc = [ 306.09598   240.78987 ] ± [ 2.38443   2.17481 ]
Skew:             alpha_c = [ 0.00000 ] ± [ 0.00000 ]   => angle of pixel axes = 90.00000 ± 0.00000 degrees
Distortion:            kc = [ -0.25425   0.22645   0.00020   0.00023   0.00000 ] ± [ 0.00934   0.03826   0.00052   0.00053   0.00000 ]
Pixel error:          err = [ 0.45330   0.38916 ]

Note: The numerical errors are approximately three times the standard deviations (for reference).


Recommendation: Some distortion coefficients are found equal to zero (within their uncertainties).
               To reject them from the optimization set est_dist=[1;1;0;0;0] and run Calibration
```

Figura 5.9. Exemplos dos parâmetros de entrada e saída da implementação desenvolvida para calibração de uma câmera (Figura retirada do Manual do toolbox de calibração de câmera do Matlab).

Após a primeira calibração da câmera, uma grande quantidade de procedimentos podem ser realizados para analisar e estimar com precisão os parâmetros da câmara, como por exemplo: utilizar a ferramenta de análise de erros que mostra o gráfico da correlação cruzada do erro de re-projeção; visualização 3D gráfica dos parâmetros extrínsecos da câmera; adicionar e suprimir imagens de calibração; realizar novas calibrações escolhendo parâmetros a otimizar ou optando por não os estimar; visualização gráfica do efeito da distorção ao longo da área de imagem, dentre outros. Após realizar cada calibração as informações de resultados são guardadas no arquivo ***Calib_Results.mat***.

5.3.6 Correção de imagens distorcidas

Esta função tem como característica criar uma nova versão não distorcida de uma ou mais imagens, dados os parâmetros intrínsecos da câmera, que foram previamente estimados.

5.3.6.1 Parâmetros de Entrada

Undistort All the Calibration Images or a New Image? – este parâmetro permite escolher as imagens distorcidas que se pretende corrigir. Quando o parâmetro de entrada é nulo "[]", todas as imagens que foram utilizadas na fase de calibração serão corrigidas. Do contrário é realizada a correção da imagem escolhida pelo utilizador, e são requeridos dois novos parâmetros de entrada:

a) ***ImageName:*** nome da imagem distorcida que se pretende corrigir;
b) ***Extension:*** formato da imagem a corrigir.

5.3.6.2 Parâmetro de Saída

Undistorted Image – A figura 5.10 mostra a função *Undistorted* onde a imagem é guardada e mostrada sem distorções. A imagem A apresenta distorções e a imagem B mostra a aplicação da função e a imagem sem distorções.

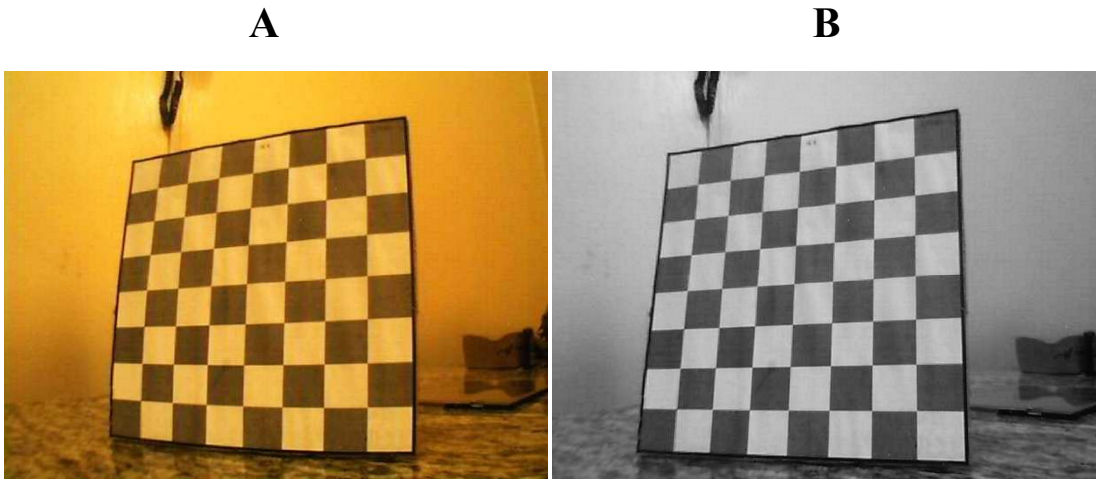


Figura 5.10. Exemplo da aplicação da função de correção de distorção de imagens

5.4 Implemetação

Na implementação do código foram desenvolvidas três classes: matriz de calibração, procura_rato e *Timer*.

5.4.1 Classe procura_rato

A classe procura_rato é a principal classe do trabalho, pois nela é feita a procura propriamente dita do objeto de interesse usando a técnica de subtração de *background*. Dentro desta classe existe uma função chamada "procura()", ela é responsável por todo o processamento de imagem, desde a captura do vídeo, até a impressão final do arquivo. Para que isso seja feito é necessário seguir alguns passos que estão descritos abaixo.

No primeiro momento da execução do programa é feita a captura do *background*, e o mesmo é salvo como uma foto comum, em seguida inicia-se o vídeo e a subtração de quadro a quadro através da função "cvAbsDiff", com o intuito de ter como objetivo final a subtração dos quadros, ou seja, o objeto de interesse.

Para que o objeto de interesse seja achado com perfeição é necessário realizar alguns passos de processamentos como mostrado a seguir.

Passo1: gera-se um *frame* com metade da dimensão do *frame* original utilizando-se a técnica de multiresolução, esta tem por finalidade reduzir o custo computacional (reduzir o número de operações matemática na imagem), como mostrado no código a seguir.

```
cvPyrDown(frame_copy, pyr_frame_copy);
```

Passo2: o *background* é subtraído do *frame*, com o objetivo de ressaltar o rato do *frame*, como podemos ver no código a seguir.

```
cvAbsDiff(pyr_frame, background, pyr_frame);
```

Passo3: extrai-se o canal azul "b" do *frame*. Isso foi feito porque em testes experimentais, após ser testado os três canais, esse canal apresentou mais eficiencia na detecção do rato. Os canais do *frame* original estão na ordem BGR (azul, verde e vermelho), como mostrados a seguir.

```
cvSplit(pyr_frame, pyr_b_frame, NULL, NULL, NULL);
```

Passo4: aplica-se um *threshold* bilateral no canal "b" para encontrar o rato. Os valores de *threshold* foram medidos experimentalmente, como mostrados a seguir.

```
cvInRangeS(pyr_b_frame,cvScalar(125), cvScalar(255), pyr_Aux1_frame);
```

Passo5: foi aplicado um dilatação e uma erosão no *frame* resultante com objetivo de reduzir os ruídos da imagem.

```
cvDilate(pyr_Aux1_frame, pyr_Aux1_frame);
cvErode(pyr_Aux1_frame, pyr_Aux1_frame);
```

Passo6: encontra-se os contornos dos objetos da imagem binária resultante utilizando-se a função *cvFindContours*, com os seguintes parâmetros:

CV_RETR_EXTERNAL – Retorna apenas contornos externos

CV_CHAN_APPROX_SIMPLE – Método de aproximação

```
cvFindContours( pyr_aux1_frame, storage, &contours, sizeof(CvContour),
CV_RETR_EXTERNAL, CV_CHAIN_APPROX_SIMPLE );
```

Passo7: encontra-se o contorno do objeto com a maior área. Para cada contorno encontrado, calcula-se sua área através da função *cvContourArea*. Caso a área calculada seja maior que max_area, calcula-se o retângulo ao redor do contorno correspondente utilizando a função *cvBoundingRect* e max_area é definido como o valor da área encontrada, como mostrado a seguir.

```
double max_area = 0, area;
CvRect rect;
while(contours!=NULL)

 {
        // Retorna a área da região delimitada pelo contorno e o contorno
        area =  fabs(cvContourArea(contours));
        if (area > max_area)
 {
        // Retorna um retângulo em torno do contorno
        rect = cvBoundingRect(contours);
        max_area = area;
 }
        contours = contours->h_next; //  próximo contorno
}
```

Passo8: calcula-se o centro do retângulo do objeto de maior área. O valor resultante será assumido como o centro do objeto de interesse no *frame* (Rato).

```
if (max_area > 0)

 {

        CvRect r_max = rect;
```

```
            u_max = 2*static_cast<int>(r_max.x + r_max.width/2) ;
            v_max = 2*static_cast<int>(r_max.y + r_max.height/2) ;

        }
```

Passo9: desenha-se no *frame* de tamanho original um vetor indicando a posição do rato com relação ao sistema de coordenadas com origem no canto superior esquerdo do *frame*. Para desenhar os vetores o programa faz o uso de uma rotina retirada da internet, que dado dois pontos p e q de uma imagem desenha uma seta de p para q (desenha_seta).

```
        // Desenha os eixos das coordenadas x,y da imagem
        CvFont font;
        cvInitFont(&font, CV_FONT_HERSHEY_DUPLEX,0.5,0.5,0.0,1);
        desenha_seta(frame, cvPoint(0,0), cvPoint(200,0));
        cvPutText(frame,"x ", cvPoint(200,10), &font, CV_RGB(255,0,0));
        cvPutText(frame,"y ", cvPoint(10,200), &font, CV_RGB(255,0,0));
        desenha_seta(frame, cvPoint(0,0), cvPoint(0,200));
        desenha_seta(frame, cvPoint(0,0), cvPoint(u_max,v_max));
```

Ainda dentro da função “procura()”, existe um modulo destinado a imprimir dados em um arquivo texto. Esse arquivo texto é composto de quatro colunas na seguinte ordem: a primeira coluna é o evento, a segunda coluna é o tempo, a terceira é posição “Fx” e a ultima coluna é a posição “Fy” do eixo cartesiano, como podemos ver a seguir.

```
            if( dist > D )

{
        evento++;
        fprintf(file,"\n%i", evento);
        timer.stop();
        fprintf(file, "\t%.3f", timer.getElapsedTime());
        fprintf(file, "\t%i \t%i", (int)u_max, (int)v_max);
```

```
std::cout << "Evento #" << evento << " - Time: " << timer.getElapsedTime() <<
" - Pos: " << "(" << u_max << "," << v_max << ")" << std::endl;
        }
```

Outro ponto de grande relevância foi a criação de uma variável “D” onde a mesma é dada pela raiz quadrada de um número, que tem como objetivo principal definir um determinado raio a partir de um ponto da imagem, ou seja, se é definido um D de três centímetros, por exemplo, assim que o objeto de interesse sair do ponto atual e se mover para outro ponto em uma distancia acima de três centímetro é gerado um novo evento (um novo registro com todas as informações Evento, Tempo, Fx e Fy).

5.4.2 Classe Matriz de Calibração

A classe matriz de calibração tem como objetivo principal fazer a calibração da câmera. Para isso, a função “carrega_matrizes” tem como objetivo principal ler uma linha de um arquivo texto que contém os seguintes dados de calibração: parâmetros intrínsecos, parâmetros extrínsecos, coeficiente de distorção e distância focal. Após a leitura desse arquivo a função carrega esses parâmetros e faz a correção de todos os *frames* do vídeo. Uma vez carregado esses parâmetros, utiliza-se a função *cvUndistort2* para corrigir cada *frame* lido do vídeo, como mostrado a seguir.

```
cvUndistort2( aux, frame_copy, matriz_calibracao.get_intrinsic_mat(),
matriz_calibracao.get_distortion_coeff() );
```

5.4.3 Classe Timer

A classe *Timer* tem como objetivo principal a contagem do tempo. Dentro da classe *timer* existem três funções: a primeira é a *“Bool Start()”,* que tem como finalidade iniciar a contagem do tempo, em seguida a “*Bool Stop*”, que tem como finalidade parar o tempo e por fim a “*double getElapsedTime()”* que retorna o tempo corrente.

Esta classe é de grande relevância para o desenvolvimento do processo de monitoramento de camundongo. Um dos parâmetros de saída do monitoramento de

camundongo é o tempo, ou seja, em que instante o mesmo se encontrava naquela coordenada X e Y, por exemplo. Como mostrado a seguir.

```
class Timer
{
public:
bool start(){
        this->_start = (clock_t)0;
        this->_start = clock();
        if( (int)_start != -1 )
                return true;
        else
                return false;
}

bool stop(){
        this->_finish = (clock_t)0;
                this->_finish = clock();
        if( (int)_finish != -1 )
                return true;
        else
                return false;
}

double getElapsedTime(){
        return (double)(this->_finish - this->_start) / CLOCKS_PER_SEC;
   }

private:
clock_t _start, _finish;
};
```

5.4.4 Funções do OpenCv

Algumas funções da biblioteca *OpenCv* foram utilizadas no código e estão listadas nas tabelas 5.1[4].

Tabela5.1. Funções da biblioteca OpenCv utilizadas no sistema de monitoramento

CXCORE	***HighGUI***	***CV***
cvAbsDiff	*cvShowImage*	*cvPyrDown*
cvSplit	*cvSaveImage*	*cvDilate*
cvInRangeS	*cvNamedWindow*	*cvErode*
cvCreateMemStorage	*cvDestroyWindow*	*cvFindContours*
cvReleaseMemStorage	*cvLoadImage*	*cvContourArea*
cvInitFont		*cvBoundingRect*
cvPutText		*cvcamSelectCamera*
cvCreateImage		*cvcamSetProperty*
cvFlip		*cvcamInit*
cvCopy		*cvcamStart*
cvGetTickCount		*cvcamStop*
cvReleaseImage		*cvcamExit*

[4] Maiores detalhes relacionados às funções, ver apêndice.

Capítulo 6 – Experimentos e Resultados

Neste capítulo será mostrado como foram realizados os experimentos utilizando o sistema de monitoramento desenvolvido.

6.1 Camundongo

O camundongo utilizado em laboratório originou-se do selvagem (*Mus musculus*) que vivia nos campos e celeiros. As colaborações dos cientistas levaram ao desenvolvimento de linhagens isogênicas e heterogênicas disponíveis hoje em grande número.

O camundongo é um animal de laboratório muito comum devido a facilidade de manuseio e ao baixo custo de manutenção. A facilidade com que grande número de camundongos pode ser mantido em ambientes de laboratório e a quantidade de mutações diferentes que foram encontradas, tornando esse pequeno roedor um modelo favorito para os cientistas como mostrado na figura 6.1.

Graças ao seu tempo curto de vida (De 6 a 8 meses) quando comparado a coelhos, cães, gatos ou primatas não humanos, o camundongo tornou-se um modelo ideal para estudos relacionados a idade, toxicologia crônica, comportamento, microbiologia e teratologia, e para a avaliação e eficácia dos fármacos.

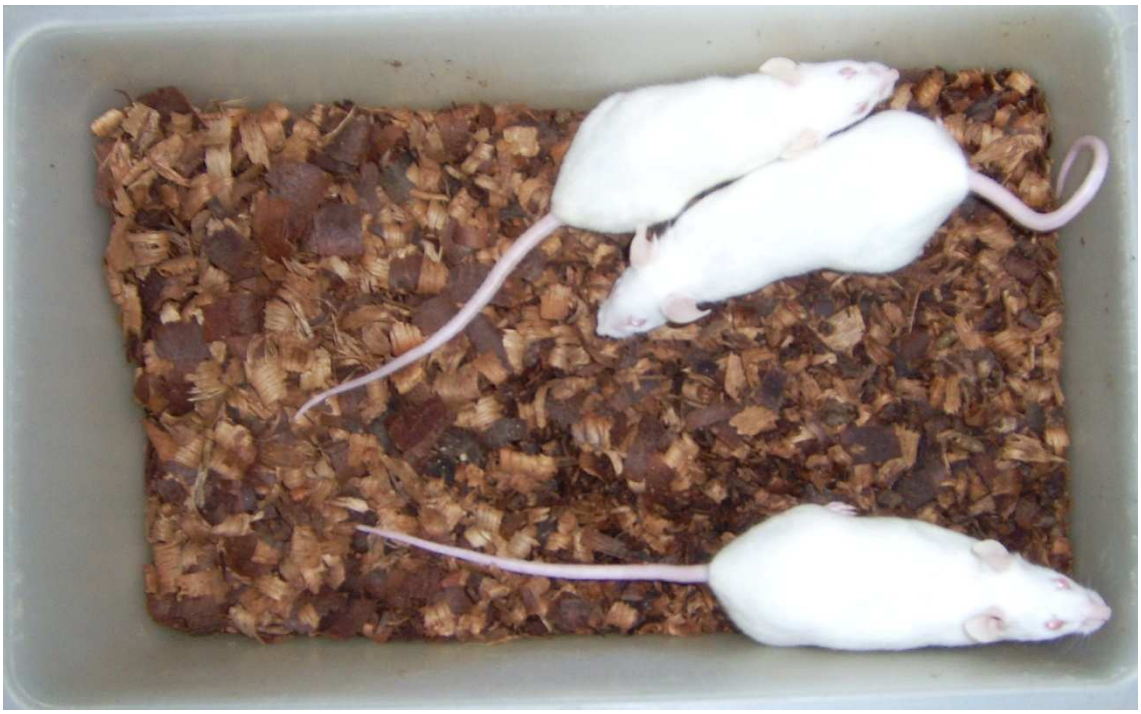


Figura 6.1. Camundongos utilizados na pesquisa

6.2 Materiais e Métodos

Para que este procedimento fosse realizado utilizou-se uma caixa retangular de madeira (55 X 24 centímetros) com o fundo pintado de preto para ajudar a dar melhor realce aos animais de estudo, três camundongos, uma *WebCam* e um computador.

Cada camundongo foi colocado na caixa individualmente e foi monitorado até um número determinado de deslocamentos dentro da caixa, ou seja, um número relacionado a quantas vezes o mesmo gerou um novo evento.

6.2.1 Interfaces do Sistema

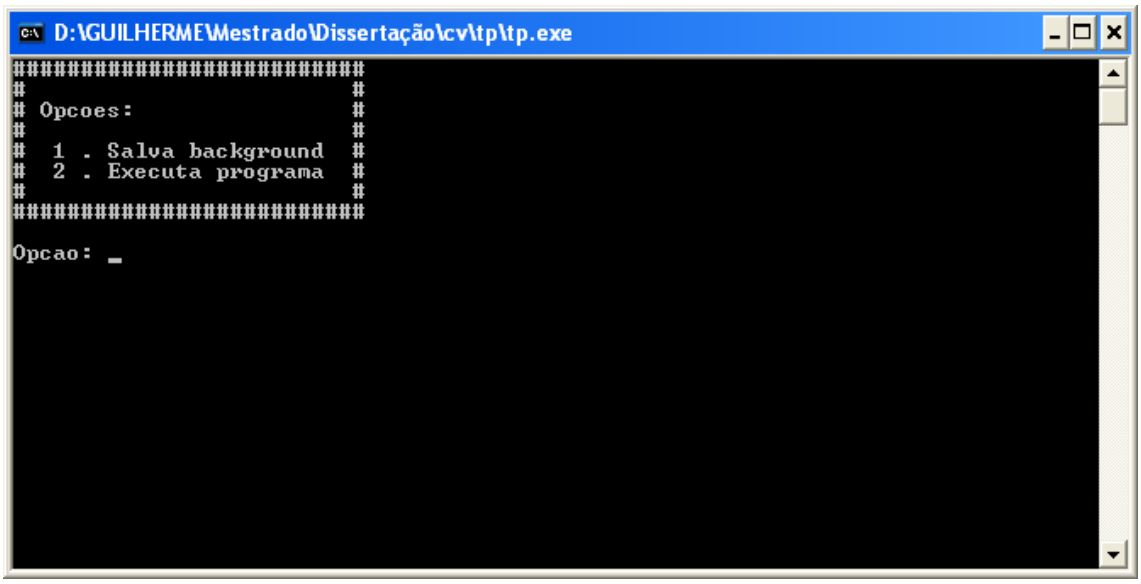


Figura 6.2. Tela inicial do sistema

A figura 6.2 mostra a tela principal do sistema, dando início ao processo de monitoramento. Primeiramente é escolhido o número "1", em que é feita a captura do *background*, e em seguida se escolhe o número "2", onde começa todo o processo de monitoramento.

Após o início do programa será mostrado quatro janelas distintas, tela de resultado, tela de *threshold*, tela de subtração de *background* e tela de impressão dos parâmetros.

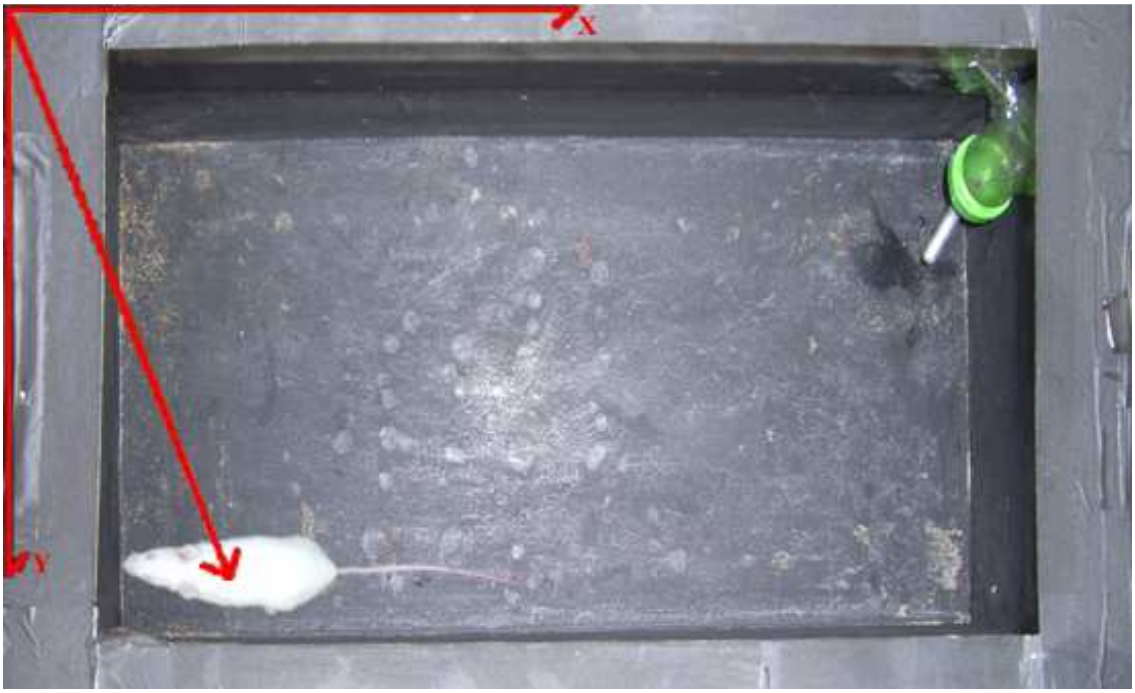


Figura 6.3. Tela de Resultado

A figura 6.3 mostra a detecção do objeto de interesse (o camundongo) através da subtração de *background*, e em seguida pode se observar a seta vermelha automaticamente seguindo o animal e tem como objetivo detectar a posição em que o objeto de interesse se encontra.

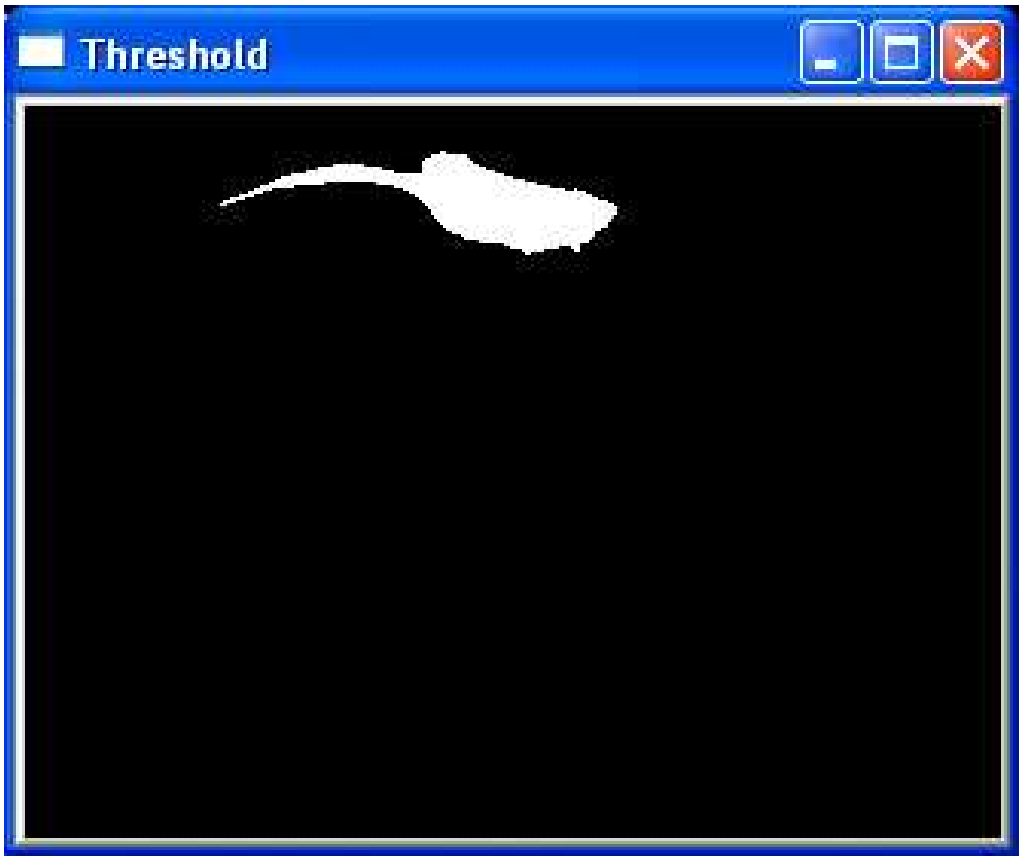


Figura 6.4. Tela de Threshold

A figura 6.4 mostra os ajustes dos parâmetros relacionados ao *threshold*, entre o objeto de interesse e o resto do ambiente. Através desse ajuste fica mais fácil a detecção do objeto.

Figura 6.5. Tela de Subtração de background

A figura 6.5 mostra a subtração de quadro a quadro do vídeo, ou seja, o que sobra da subtração é o movimento que é detectado pelo *software*.

6.3 Análise estatística dos dados

O ambiente *Matlab* foi utilizado como ferramenta principal para a análise dos dados devido ao seu grande número de funções, no que diz respeito a análise estatística.

Dando início à análise é importante visualizar o modelo de dados gerados no arquivo analisado como mostra a tabela 6.1.

Tabela 6.1. Exemplo dos dados gerados pelo sistema de monitoramento de animais experimentais.

Evento	Tempo (Segundos)	Posição Fx (Pixel)	Posição Fy(Pixel)
1	6.031	398	302
2	10.796	280	286
3	10.937	230	276
4	15.640	124	258
5	17.109	246	176
6	17.359	296	142
7	17.484	262	148
8	17.687	334	130
9	22.578	390	126
10	24.312	458	154
11	25.109	504	174
12	25.359	544	196
13	34.671	508	296
14	35.093	522	276
15	40.750	452	178
16	42.234	434	188
17	42.875	238	234
18	43.921	92	272
19	49.421	208	146
20	49.812	272	126

"Na tabela 6.1, encontramos as quatro colunas que representam Evento, Tempo, FX e FY respectivamente."

O sinal da figura 6.6, foi *plotado* com as dimensões do labirinto usado no experimento.

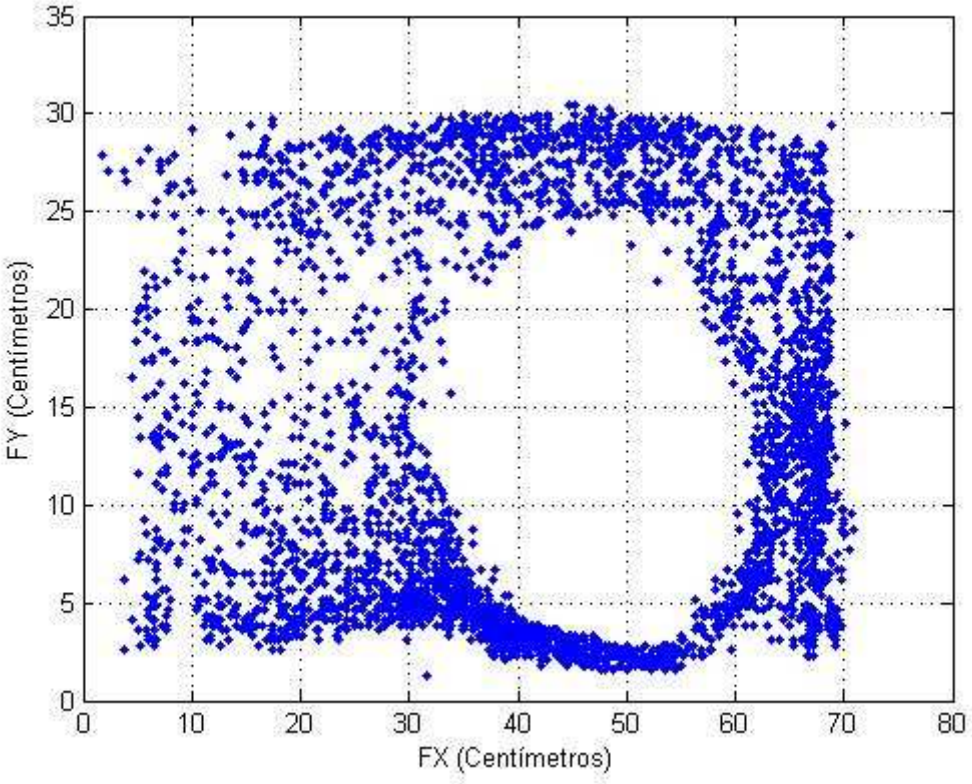


Figura 6.6. Gráfico que mostra o sinal nas posições FX e FY.

6.3.1 Primeira Análise

Com base nos dados gerados pelo arquivo, a primeira função desenvolvida está relacionada aos ajustes gerais, que têm por objetivo capturar a imagem como entrada e separar apenas a arena onde o camundongo se movimentou como saída, ou seja, o fundo da caixa.

Em seguida a função realiza a transformação de medidas de *pixel* para centímetro com o objetivo de facilitar o entendimento (Transformação das escalas de Pixel para Centímetro).

6.3.2 Segunda Análise

A função desenvolvida na segunda análise caracteriza-se em dividir o fundo da caixa de madeira em uma matriz e, em seguida, fazer a soma de todos os pontos de cada região, ou seja, quantas vezes o camundongo ficou ou passou em cada região, com o objetivo de analisar os locais de maior interesse do animal em um período de tempo, como mostrado na figura 6.7.

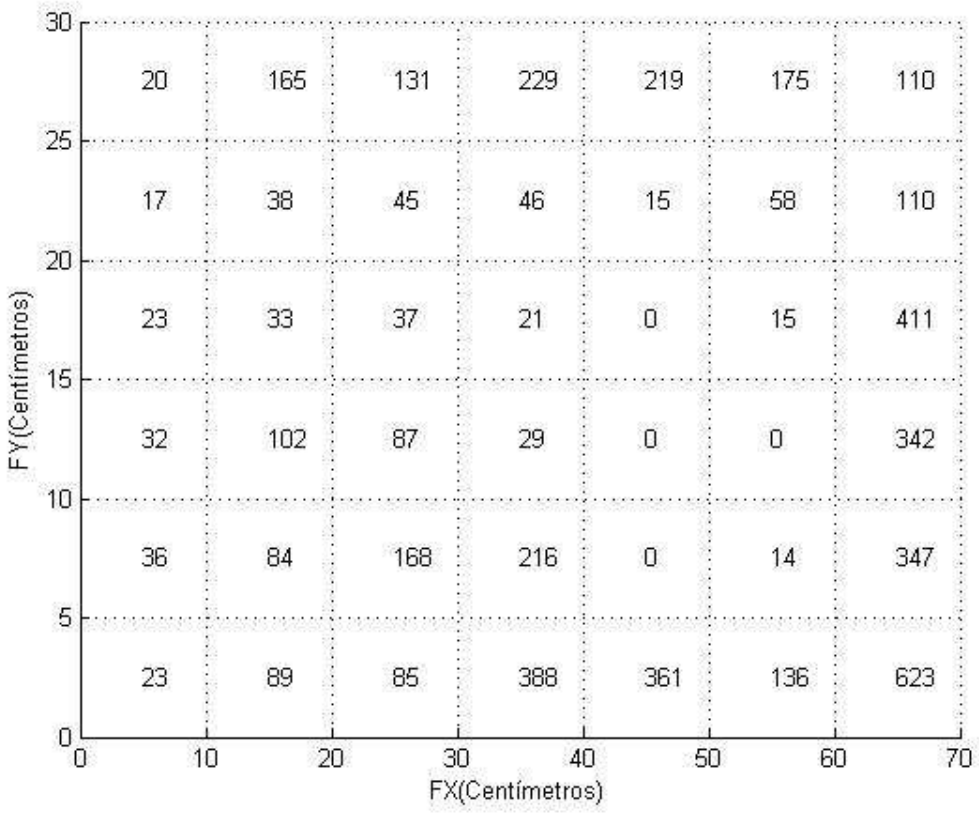


Figura 6.7. Gráfico da matriz da soma dos pontos em cada célula.

6.3.3 Terceira Análise

A função desenvolvida na terceira análise caracteriza-se por realizar a soma de todos os pontos de cada célula como mostrado na figura 6.9. Isso é importante para abrir uma discussão onde se podem fazer algumas perguntas, como por exemplo:

- Por onde ele passou mais vezes foi onde ele ficou por mais tempo?
- Ele ficou muito ou pouco tempo perto da comida?
- Ele ficou muito ou pouco tempo perto do bebedouro?
- Em que local da caixa ele dormiu?

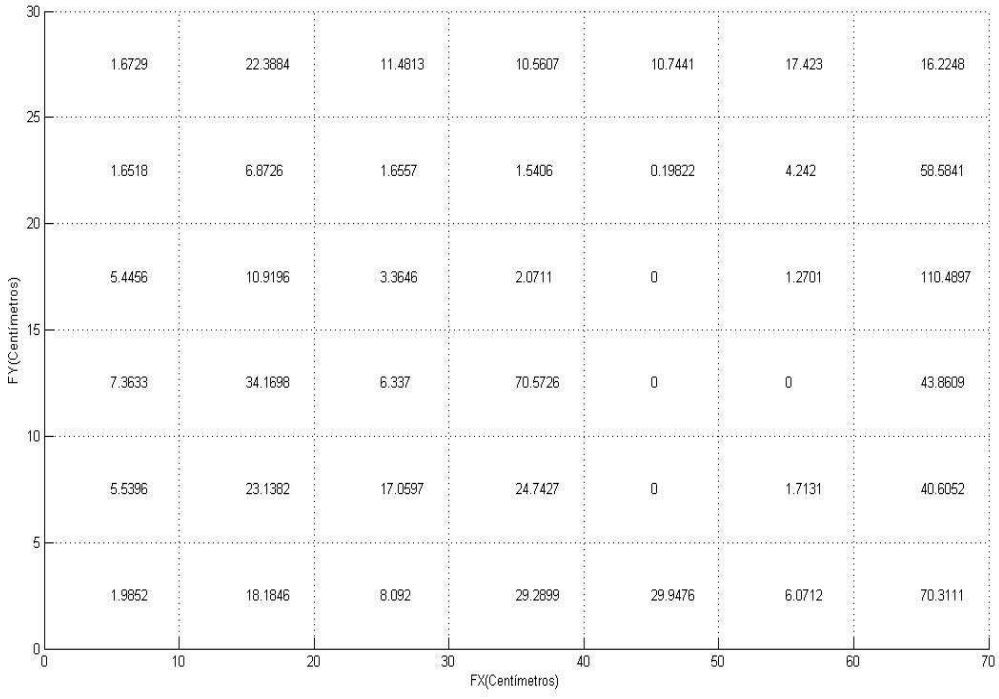


Figura 6.8. Gráfico da matriz da soma do tempo dos pontos em cada célula.

Discussão

Para este trabalho foram realizadas pesquisas sobre algoritmos de segmentação de imagem e subtração de *background,* além da biblioteca de visão computacional *OpenCV*. Foi proposto um sistema em tempo real para a detecção e o rastreamento de animais experimentais em labirintos. Esse sistema foi construído utilizando-se algoritmos de subtração de *background* e detecção de contornos.

Outro propósito do trabalho foi mostrar que sistemas de interação baseados em visão computacional podem ser implementados em computadores convencionais utilizando dispositivos de *captura* acessíveis (baixo custo).

O uso da segmentação da imagem justifica-se pela necessidade de separar o camundongo do labirinto e pelas vantagens deste método que permite localizar o camundongo mesmo que ele mude de forma de uma cena para outra, por exemplo, quando o animal se levanta, ficando apoiado apenas nas patas traseiras. Apesar do processo de segmentação do camundongo apresentar algumas falhas de classificação, tais como, cor e movimento, estas foram superadas através da utilização de filtros de suavização e de morfologia matemática. Alguns desses erros de segmentação muitas vezes são ocasionados pelo brilho e reflexão de alguns objetos e foram atenuados nas etapas posteriores à segmentação.

No que se refere ao labirinto utilizado, as condições de iluminação e a presença e intensidade de sombras nas imagens definem os parâmetros da segmentação. Uma dificuldade foi estabelecer mecanismos de ajuste automáticos, ou semi-automáticos, que considerassem essas características.

A utilização da *WebCam* como ferramenta para a aquisição de vídeo nos experimentos com camundongos mostrou ser adequada para o monitoramento desses animais.

O método de subtração de *background* deve ser usado com moderação. É imprescindível que as fotos e os filmes utilizados possuam a mesma condição de posição e tenham as mesmas características de edição de imagem (como contraste e brilho), e ainda, que a *WebCam* permaneça fixa na posição inicial ao longo da filmagem. Com isso, garantimos que a cena de fundo seja fiel à imagem de fundo do filme, pois qualquer diferença acrescentaria ruído à imagem segmentada.

É importante observar que o uso da *WebCam* e do método de segmentação escolhido apresentaram total compatibilidade, pois foi possível se ter uma cena inicial e o filme com a mesma *WebCam*, na mesma posição, com os mesmos fatores de edição.

Com o *software* desenvolvido foi possível automatizar o teste de campo aberto. Para o *software* ter um bom funcionamento foram estabelecidas algumas normas de utilização. Quanto ao objeto a ser monitorado, deverá apresentar uma cor que estabeleça um contraste em relação à cor de fundo do labirinto, ou seja, se o fundo for preto o ideal é que o animal seja de cor branca ou vice-versa. Em relação à captura do vídeo, o equipamento utilizado deverá ser posicionado na parte superior do labirinto, tendo como ângulo de filmagem a parte inferior do mesmo (o fundo do labirinto), devendo esse estar livre de grades e outros objetos transparentes, tais como, vidros e acrílicos. Em relação ao número de animais a serem monitorados a cada filmagem, o *software* desenvolvido, só é capaz de monitorar um animal por filmagem.

Os experimentos realizados duraram em média 24 horas e os animais monitorados tiveram acesso a água e comida à vontade, e encontravam-se sem o efeito de nenhuma droga.

Em relação ao ambiente de realização do experimento, nesta pesquisa foi utilizada uma arena retangular, seguindo a mesma idéia do *Open Field*, que é definido fisicamente como uma arena de campo aberto em forma de círculo confeccionada em madeira ou material plástico, com a base dividida em quadrantes, circundada por uma parede de mesmo material (Broekkamp *et al.,* 1986).

A função desenvolvida no *software* permite a delimitação de um raio através de um ponto, ou seja, a partir de um ponto é gerado um evento e um raio. Quando o animal se movimenta e ultrapassa esse raio, é gerado um novo evento e um novo raio para o novo ponto. É importante salientar que os pesquisadores utilizam esta arena para uma análise relacionada à atividade exploratória dos animais. A tendência natural do animal em um ambiente novo é a de explorá-lo, apesar do estresse e do conflito provocado pelo ambiente novo (Montgomery, 1955). Neste teste observa-se o número de comportamentos de autolimpeza, o tempo em que o animal permanece parado, o tempo em que permanece se movimentando, o tempo consumido em cada região e a defecação, como índice de emocionalidade. O *software* desenvolvido é capaz de analisar alguns desses padrões como o tempo em que o animal permanece parado e o tempo em que permanece se movimentando.

Neste trabalho foram analisados alguns parâmetros de monitoramento como tempo, deslocamento e posição nas coordenadas X e Y. Analisando esses parâmetros, podemos obter informações a respeito de locais onde o camundongo costuma ficar ou apenas passar. No sinal que foi extraído pelo *software* que estamos usando neste trabalho, podemos observar nas figuras 7.8 e 7.9, o movimento que o animal teve durante um período de tempo levando em consideração os momentos em que apenas passou pelos pontos e os períodos em que permaneceu nos mesmos, ou seja, dormiu ou ficou mais tempo naquele ponto por outros motivos. Com estas informações podemos analisar diversos tipos de fármacos, como por exemplo, os ansiolíticos. Alguns estudos têm demonstrado que alterações nesses parâmetros têm correlação com a ansiedade no homem (Masur, 1971).

O *software* de visão computacional implementado para este trabalho foi desenvolvido considerando a possibilidade de que novos classificadores possam ser adicionados, ou seja, foi desenvolvido seguindo as normas de programação orientada a objetos. Com isso é possível que outros alunos e pesquisadores que desejarem trabalhar com monitoramento de animais possam utilizar seus métodos sem ter que implementar um sistema inteiro de visão computacional. É interessante ressaltar que o *software* desenvolvido pode ser utilizado em outras aplicações similares envolvendo visão computacional.

O monitoramento de animais experimentais visa a suprir as necessidades dos pesquisadores na compreensão do funcionamento dos padrões de comportamento animal, tais como ansiedade, excitação, dentre outros (Metris, 2005).

Pode-se afirmar que os objetivos propostos foram alcançados de modo satisfatório. Foi desenvolvido um sistema seguindo as metodologias de projeto, onde foram aplicados os conhecimentos de visão computacional, técnicas de processamento de imagem, calibração de câmeras, especificação do sistema, implementação do *software* e experimentos realizados, culminando nos testes do *software*.

Como trabalhos futuros pretende-se fazer uma adaptação do *software* para outros tipos de labirintos que medem a atividade motora do animal; como exemplos podemos citar o labirinto aquático de *Morris* (Grossmann e Skinner, 1996) e o labirinto em cruz elevado (Boguszewski e Szmagalska, 2002). Pode-se também implementar funções para identificação de sinais específicos como micção, defecação, *writhing* (contorções abdominais), pêlos arrepiados, tremores, convulsões, postura, dentre outros. Outro ponto que poderia ser expandido no *software* é o número de variáveis a serem

analisadas. Finalmente, outra maneira de utilização do sistema, seria a de realizar experimentos com outros tipos de animais em ambientes variados.

Referências Bibliográficas

Bakstein H (1999). Diploma Thesis: A Complete DLT-based Camera Calibration witha Virtual 3D Calibration Object. Faculty of Mathematics and Physics, Charles University, Prague.

Berard F (1999). Computer Vision for the Strongly Coupled Human-Computer Interaction. Doctoral Theses, Université Joseph Fourier.

Boehm BW (1989). Software risk management. IEEE Computer Society Press: Washington.

Boguszewski P e Szmagalska JZ (2002). Emotional changes related to age in rats a behavioral analysis. Behavioural Brain Research, 133:332–332.

Boisser JR e Simon P (1976). Central Nervous system and Behavioural Pharmachology, Oxford: Pergamon Press.

Broekkamp CL, Rijik HW, Geloind J, Lioyd KL (1986). Major tranquilizer can be distinguished from minor tranquilizer on the basis of effects on marble butying and swiminduced grooming in mice. European Journal of Pharmacology.

Brown DC (1971). Photogrammetric Engineering, pages 855-866, Vol. 37, No. 8.

Carvalho THF e Lopes OU (2006). O emprego de camundongo geneticamente modificado como modelo de estudo para doenças cardiovasculares. In X Simpósio Brasileiro de Fisiologia Cardiovascular, volume 39, pages 110–116, Ribeirão Preto,Brasil.

Castro JFB (1995). Introdução à engenharia de requisitos. In: XV Congresso da Sociedade Brasileira de Computação, JAI'95, Canela, RS, Brasil.

Crowley JL e Christensen HI (1995). Vision as Process. Springer-Verlag.

Davies ER (2005). Machine Vision: Theory, Algorithms, Practicalities. Morgan Kaufmann.

Fagundes DJ e Taha MO (2004). Modelo animal de doença: critérios de escolha e espécies de animais de uso corrente. Acta Cirúrgica Brasileira, 19:59–65.

Faugeras O (1993). Three-Dimensional Computer Vision, A Geometric Viewpoint. MIT Press.

Ferreira ABH (1986). Novo Dicionário Aurélio da Língua Portuguesa. Segunda Edição - revista e ampliada). Editora Nova Fronteira.

Fisher R, Dawson-Howe K, Fitzgibbon A, Robertson C e Trucco E (2005). Dictionary of Computer Vision and Image Processing. John Wiley.

Forsyth AD e Ponce J (2003). Computer Vision, A Modern Approach. Prentice Hall.

Fryer JG e Brown DC (1986). Photogrammetric Engineering and Remote Sensing Vol. 52(1) pp 51-58.

Graeff FG e Guimarães FS (1999). Fundamentos da Psicofarmacolofia, São Paulo: Atheneu.

Granlund GH e Knutsson H (1995). Signal Processing for Computer Vision. Kluwer Academic Publisher.

Gonzalez RC e Woods RE (1992). Processamento de Imagens Digitais. Edgard Blucher.

Grossmann M e Skinner M H (1996). A simple computer based system to analyze morris water maze trials on-line. Journal of Neuroscience Methods, 70(2):171–175.

Haritaoglu I, Harwood D e Davis LS (2000). "W4: real-time surveillance of people and their activities". IEEE Transactions on Pattern Analysis and Machine Intelligence. pp. 809-830.

Hartley R e Zisserman A (2003). Multiple View Geometry in computer vision. Cambridge University Press.

Horn BKP (1986). Robot Vision. MIT Press.

IEEE Std. 610.12 IEEE Standard Glossary of Software Engineering Terminology. The Institute of Electrical and Electronics Engineers. New York, 1990.

IEEE Std. 830. IEEE Guide to Software Requirement Specification. The Institute of Electrical and Electronics Engineers. New York, 1984.

Jähne B e Haußecker H (2000). Computer Vision and Applications, A Guide for Students and Practitioners.

Jähne B (2002). Digital Image Processing. Springer.

Klette R, Schluens K e Koschan A (1998). Computer Vision - Three-Dimensional Data from Images. Springer, Singapore.

Levin G (2004). "Computer Vision for Artists and Designer: Pedagogic Tools and Techniques for Novice Programmers". Carnegie Mellon University.

Lindeberg T (1994). Scale-Space Theory in Computer Vision. Springer, 1994.

Marr D (1982). Vision. W. H. Freeman and Company, 1982.

Masur J, Martz RM e Carlini EA (1971). Effects of acute and chronic administration of cannabis sativa and tetrahydrocannabinol on the behavior of rats in on open-field arena. Psychopharmachology.

Mayrhauser AV (1990). Software Engineering: Methods and Management. Academic Press.

McFarlane N e Schopfield C (1995). "Segmentation and tracking of piglets in images". Machine Vision and Applications 8(3). pp. 187-193.

Medioni G e Kang SB (2004). Emerging Topics in Computer Vision. Prentice Hall.

Metris BV (2005). High-quality measurements of rodent behavior tracking and ultrasounds using laboras and sonotrack. Business Briefing, Future Drug Discovery.

Montgomery KC (1955). The relationship between fear induced by novel stimulation an exploration behavior. J Comp Physiol Psychol.

Morris T (2004). Computer Vision and Image Processing. Palgrave Macmillan.

Morrow-Tesch J, Dailey JW, e JIang H (1998). A video data base system for studying animal behavior. Journal of Animal Science, 76(10).

Nogueira L (2003). Tecnologia Farmacêutica, vol. I, 6ª edição, Fundação Calouste Gulbenkian

Nogueira TCML (1997). II curso de validação de plantas medicinais com atividade no sistema nervoso central. Programa Iberoamericano de ciência e Tecnologia para o desenvolvimento / Rede iberoamericana de validação de plantas medicinais, Florianópolis.

Orth A (2001). Desenvolvimento de um Sistema de Visão para medir o Desgaste de Flanco de Ferramentas de Corte. Dissertação de mestrado, Universidade Federal de Santa Catarina - Programa de Pós Graduação em Engenharia Elétrica, Florianópolis - SC - Brasil.

Pavim AX (2002). Seminário em Sistemas de Visão: Aplicações no Controle de Qualidade e Rastreabilidade.

Pérez P, Hue C, Vermaak J, e Gangnet M (2002). Color-based probalistic tracking. European Conference on Computer Vision.

Piccardi M (2004). Background subtraction techniques: a review. In Proc. of IEEE SMC 2004 International Conference on Systems, Man and Cybernetics.

Pressman RS (1994). Software engineering: a practitioner's approach. Euopean Edition.

Prut, L e Belzung C (2003). The open field as a paradigm to measure the effects of drugs on anxiety-like behaviors: a review. European Journal of Pharmacology, pages 3–33.

Rosenfeld A e Kak A (1982). Digital Picture Processing. Academic Press.

Shapiro LG e Stockman GC (2001). Computer Vision. Prentice Hall.

Sonka M, Hlavac V e Boyle R (1999). Image Processing, Analysis, and Machine Vision. PWS Publishing.

Sonka M, lavac V e Boyler R (2007). Image processing, analysis, and machine vision. Thomson-Engineering.

Souza, KP e Pistori H (2005). Implementação de um extrator de características baseado em momentos da imagem. In Proceedings of SIBGRAPI 2005 - XVIII Brazilian Symposium on Computer Graphics and Image Processing, Natal, RN, Brazil. SBC, IEEE Press.

Sminchiescu C e Telea A (2002). "Human pose estimation from silhouettes. A consistent approach using distance level sets". WSCG International Conference on Computer Graphics.

Trucco E e Verri A (1998). Introductory Techniques for 3-D Computer Vision. Prentice Hall.

Wolberg G (1994). Digital image warping. Los Alamitos, CA, USA: IEEE Computer Society Press.

Apêndice I

As funções do OpenCv utilizadas no software de monitoramento de animais experimentais, estão descriminadas e detalhadas nesse apêndice.

- CXCORE

  - **cvAbsDiff**

    - *Descrição*

A função cvAbsDiff calcula a diferença absoluta entre duas matrizes.

$dst(I)_c = abs(src1(I)_c - src2(I)_c)$.

Todas as matrizes devem ter o mesmo tipo de dados e as mesmas dimensões (tamanho ou ROI).

    - *Declaração*

void cvAbsDiff(const CvArr* src1, const CvArr* src2, CvArr* dst);

    - *Parâmetros*

src1

A primeira posição da matriz.

src2

A segunda posição da matriz.

dst

A posição de destino da matriz.

❖ **cvSplit**

- *Descrição*

A função cvSplit divide um array multicanal em vários canais separados. Estão disponíveis dois modos de operação. Se a fonte array tem N canais então se o primeiro N canais não é NULL, todos eles serão extraídos a partir da fonte array, caso contrário, se apenas um único destino o primeiro canal de N não é NULL, este canal específico é extraído, caso contrário, uma erro é levantada. Para o resto de destino dos canais (para além do primeiro N) deve ser sempre NULL. Para IplImage cvCopy conjunto com a COI pode ser também usada para extrair um único canal a partir da imagem.

- *Declaração*

```
void cvSplit( const CvArr* src, CvArr* dst0, CvArr* dst1,
              CvArr* dst2, CvArr* dst3 );

#define cvCvtPixToPlane cvSplit
```

- *Parâmetros*

src

Array de origem.

dst0...dst3

Canais de destino

❖ **cvInRangeS**

- *Descrição*

A função cvInRangeS faz o “check faixa” para cada elemento do array de entrada:

DST (I) = lower0 <src = (I) 0 <upper0

para uma única matriz de canal,

DST (I) = lower0 <src = (I) 0 <upper0 & &
lower1 <src = (I) 1 <upper1

por um período de dois canais array etc.

DST (I) está definido para 0xff (todos os'1'-bits) se src (I) se encontra dentro do intervalo e 0 de outra forma. Todos os arranjos devem ter as mesmas dimensões (tamanho ou ROI).

- *Declaração*

void cvInRangeS(const CvArr* src, CvScalar lower, CvScalar upper, CvArr* dst);

- *Parâmetros*

src

O array de origem.

lower

O limite inferior inclusive.

upper

O limite superior exclusiva.

dst

O array de destino, deve ter 8U ou 8S.

- **cvCreateMemStorage**

  - *Descrição*

A função cvCreateMemStorage cria uma memória de armazenamento e retorna um ponteiro para ele. Inicialmente, o armazenamento está vazio. Todos os campos do cabeçalho, com excesssão do block_size, são fixados em 0.

  - *Declaração*

CvMemStorage* cvCreateMemStorage(int block_size=0);

  - *Parâmetros*

block_size

O tamanho dos blocos de armazenamento em bytes. Se for 0, o tamanho do bloco é definido como o valor padrão - atualmente é ≈ 64 K.

- **cvReleaseMemStorage**

  - *Descrição*

A função cvClearMemStorage redefine o topo (espaço livre fronteira) para o armazenamento do início. Esta função não deloca qualquer memória. Se o armazenamento tem um "pai", a função retorna todos os blocos para os pais.

- *Declaração*

void cvReleaseMemStorage(CvMemStorage** storage);

- *Parâmetros*

storage

Ponteiro para o armazenamento.

❖ **cvInitFont**

- *Descrição*

A função cvInitFont inicializa o tipo de letra e estrutura que pode ser passada para o texto.

- *Declaração*

void cvInitFont(CvFont* font, int font_face, double hscale,
double vscale, double shear=0, int thickness=1, int line_type=8);

- *Parâmetros*

font

Ponteiro para o tipo de estrutura inicializada pela função.

font_face

CV_FONT_HERSHEY_SIMPLEX - tamanho normal sans-serif font
CV_FONT_HERSHEY_PLAIN - tamanho pequeno sans-serif font
CV_FONT_HERSHEY_DUPLEX - tamanho normal sans-serif font (mais complexa do que CV_FONT_HERSHEY_SIMPLEX)
CV_FONT_HERSHEY_COMPLEX - tamanho normal serif font
CV_FONT_HERSHEY_TRIPLEX - tamanho normal serif font (mais complexa do que CV_FONT_HERSHEY_COMPLEX)
CV_FONT_HERSHEY_COMPLEX_SMALL - versão menor do CV_FONT_HERSHEY_COMPLEX
CV_FONT_HERSHEY_SCRIPT_SIMPLEX - escrito à mão estilo font
CV_FONT_HERSHEY_SCRIPT_COMPLEX - variante mais complexa do CV_FONT_HERSHEY_SCRIPT_SIMPLEX

O parâmetro pode ser compostos de um dos valores acima e facultativo CV_FONT_ITALIC, que significa itálico ou oblíqua fonte.

hscale

Horizontal escala. Se igual a 1.0f, os caracteres têm a largura inicial, dependendo do tipo de letra. Se igual a 0.5f, os caracteres são a metade da largura original.

vscale

Vertical escala. Se igual a 1.0f, os caracteres têm a altura original, dependendo do tipo de letra. Se igual a 0.5f, os caracteres são a metade da altura original.

shear

Aproximado tangente do caracter de inclinação em relação à linha vertical. Zero significa um valor não-itálico, 1.0f meio ≈ 45 ° declive, etc.

thickness

Espessura do texto.

line_type

Tipos de linha

- **cvPutText**
  - *Descrição*

A função cvPutText torna o texto da imagem com um determinado tipo de letra e cor.

  - *Declaração*

void cvPutText(CvArr* img, const char* text, CvPoint org, const CvFont* font, CvScalar color);

  - *Parâmetros*

img

Imagem de entrada

text

String para imprimir

org

Coordenadas do canto esquerdo inferior

font

Ponteiro para o tipo da estrutura.

color

Cor do texto

- **cvCreateImage**
  - *Descrição*

A função cvCreateImage cria o cabeçalho e aloca os dados.

  - *Declaração*

IplImage* cvCreateImage(CvSize size, int depth, int channels);

  - *Parâmetros*

size

Imagem com largura e altura.

depth

Bit de profundidade de elementos de uma imagem. Pode ser um de:
IPL_DEPTH_8U - usando 8 bits inteiros
IPL_DEPTH_8S - assinados 8-bit inteiros
IPL_DEPTH_16U - usando inteiros de 16 bits
IPL_DEPTH_16S - assinado inteiros de 16 bits
IPL_DEPTH_32S - assinado inteiros de 32 bits

IPL_DEPTH_32F - único número de ponto flutuante de precisão
IPL_DEPTH_64F - precisão dupla de números de ponto flutuante

channels

Número de canais por cada elemento (pixel). Pode ser 1, 2, 3 ou 4. Os canais são intercalados, por exemplo, a habitual apresentação dos dados é uma cor de imagem: B0 G0 R0 b1 g1 r1 ...

- **cvFlip**

  - *Descrição*

A função cvFlip vira um array de 3 maneiras diferentes (linha e coluna índices são 0-com base):

dst(i,j)=src(rows(src)-i-1,j) if flip_mode = 0

dst(i,j)=src(i,cols(src1)-j-1) if flip_mode > 0

dst(i,j)=src(rows(src)-i-1,cols(src)-j-1) if flip_mode < 0

  - *Declaração*

void cvFlip(const CvArr* src, CvArr* dst=NULL, int flip_mode=0);
#define cvMirror cvFlip

  - *Parâmetros*

src

Array de origem.

dst

Array de destino.

flip_mode

Especifica a forma de flip da matriz. flip_mode = 0 significa lançando em torno do eixo x, flip_mode> 0 (por exemplo: 1), lançando em torno do eixo y e flip_mode <0 (por exemplo, -1), lançando em torno de ambos os eixos.

- **cvCopy**

  - *Descrição*

A função cvCopy cópia elementos selecionados.

  - *Declaração*

void cvCopy(const CvArr* src, CvArr* dst, const CvArr* mask=NULL);

  - *Parâmetros*

src

Array de origem.

dst

Array de destino.

mask

Operação com máscara, 8-bits de canal com um único array; especifica elementos do array de destino a ser mudado.

- **cvGetTickCount**

  - *Descrição*

A função cvGetTickCount retorna o número de tics dependendo da plataforma.

  - *Declaração*

int64 cvGetTickCount(void);

- **cvReleaseImage**

  - *Descrição*

A função cvReleaseImage libera o cabeçalho e os dados da imagem.

  - *Declaração*

void cvReleaseImage(IplImage** image);

- *Parâmetros*

image

Duplo ponteiro para o cabeçalho da imagem

- HighGUI:

  - **cvShowImage**

    - *Descrição*

A função cvShowImage mostra a imagem na janela especificada. Se a janela foi criada com CV_WINDOW_AUTOSIZE então a imagem é mostrada em seu tamanho original, caso contrário, a imagem é redimensionado para se ajustar à janela.

    - *Declaração*

void cvShowImage(const char* name, const CvArr* image);

    - *Parâmetros*

name

Nome da janela.

image

Imagem a ser mostrada.

  - **cvSaveImage**

    - *Descrição*

A função cvSaveImage salva a imagem para o arquivo especificado.

    - *Declaração*

int cvSaveImage(const char* filename, const CvArr* image);

- Parâmetros

filename

Nome do arquivo

image

Imagem a ser salva

❖ **cvNamedWindow**

- *Descrição*

A função cvNamedWindow cria uma janela que pode ser utilizado como um espaço reservado para imagens e vídeos.

- *Declaração*

int cvNamedWindow(const char* name, int flags=CV_WINDOW_AUTOSIZE);

- *Parâmetros*

name

Nome da janela que é utilizado como identificador da mesma e aparece na janela de legenda.

flags

Flag de ajuste da janela.

- **cvDestroyWindow**

  - *Descrição*

A função cvDestroyWindow destrói a janela com um determinado nome.

  - *Declaração*

void cvDestroyWindow(const char* name);

  - *Parâmetros*

name

Nome da janela a ser destruída.

- **cvLoadImage**

  - *Descrição*

A função cvLoadImage carrega uma imagem a partir do arquivo especificado e retorna o ponteiro para a imagem carregada. Atualmente os seguintes formatos são suportados:

  - Windows bitmaps - BMP, DIB;
  - JPEG files - JPEG, JPG, JPE;
  - Portable Network Graphics - PNG;
  - Portable image format - PBM, PGM, PPM;
  - Sun rasters - SR, RAS;
  - TIFF files - TIFF, TIF;

- OpenEXR HDR images - EXR;
- JPEG 2000 images - jp2.

- *Declaração*

IplImage* cvLoadImage(const char* filename, int flags=CV_LOAD_IMAGE_COLOR);

- *Parâmetros*

filename

Nome do arquivo a ser lido.

Flags

Especifica “colorness” e profundidade da imagem carregada:

- CV

❖ **cvPyrDown**

- *Descrição*

A função cvPyrDown utiliza o paradigma de pirâmide.

- *Declaração*

void cvPyrDown(const CvArr* src, CvArr* dst, int filter=CV_GAUSSIAN_5x5);

- *Parâmetros*

src

A imagem de origem.

dst

A imagem de destino, deve ter 2x a largura e a altura menor do que imagem de origem.

filter

Tipo de filtro utilizado para a convolução.

- **cvDilate**

  - *Descrição*

A função cvDilate dilata a imagem de origem usando um elemento estruturante que determina a forma de um pixel bairro durante o qual o máximo é tomado:

dst=dilate(src,element): $dst(x,y)=max_{((x',y') \text{ in element})}src(x+x',y+y')$

A Dilatação pode ser aplicada várias vezes. No caso de imagem de cor cada canal é processado de forma independente.

  - *Declaração*

void cvDilate(const CvArr* src, CvArr* dst, IplConvKernel* element=NULL, int iterations=1);

  - *Parâmetros*

src

Imagem de origem.

dst

imagem de destino.

element

Elemento estruturante utilizado para a erosão. Se for NULL, um elemento estruturante 3x3 é usado.

iterations

Número de vezes que a erosão vai ser aplicada.

- **cvErode**

  - *Descrição*

A função cvErode faz a erosão da imagem especificada usando o elemento estruturante que determina a forma de um pixel bairro mais que o mínimo seja tomada:

dst=erode(src,element): $dst(x,y)=\min_{((x',y') \text{ in element})}src(x+x',y+y')$

A erosão pode ser aplicada várias vezes. No caso de imagem de cor cada canal é processado de forma independente.

  - *Declaração*

void cvErode(const CvArr* src, CvArr* dst, IplConvKernel* element=NULL, int iterations=1);

  - *Parâmetros*

src

Imagem de origem.

dst

imagem de destino.

element

Elemento estruturante utilizado para a erosão. Se for NULL, um elemento estruturante 3x3 é usado.

iterations

Número de vezes que a erosão vai ser aplicada.

- **cvFindContours**

  - *Descrição*

A função cvFindContours recupera os contornos a partir da imagem binária e retorna o número de contornos recuperados. O ponteiro first_contour é preenchido com a função. Ela irá conter a maior parte do ponteiro para o primeiro contorno externo ou NULL se não for detectado nenhum contorno (se a imagem é completamente preta). Outros contornos podem ser acessados a partir de first_contour usando h_next e v_next links. A amostra cvDrawContours em discussão mostra como usar o componente ligado contornos de detecção. Contornos podem ser igualmente utilizados para moldar análise e reconhecimento de objetos.

  - *Declaração*

int cvFindContours(CvArr* image, CvMemStorage* storage, CvSeq** first_contour,

int header_size=sizeof(CvContour), int mode=CV_RETR_LIST,
int method=CV_CHAIN_APPROX_SIMPLE, CvPoint
offset=cvPoint(0,0));

- *Parâmetros*

image

Imagem de origem com 8 bit. Se for diferente de zero pixels são tratados com 1's, zero pixels permanecem 0's - que é tratado como imagem binária. A função modifica a imagem de origem conteúdo.

storage

Contornos recuperados.

first_contour

Parametro de saída, irá conter o ponteiro que irá apontar para o primeiro contorno externo.

header_size

Sequencia e tamanho do cabeçalho, >=sizeof(CvChain) se method=CV_CHAIN_CODE, and >=sizeof(CvContour) outra forma.

mode

Modo de recuperação

- CV_RETR_EXTERNAL - apenas recupera o contorno exterior
- CV_RETR_LIST - recupera todos os contornos e coloca-o na lista
- CV_RETR_CCOMP - recupera todos os contornos e organiza-os em níveis hierárquicos.

- CV_RETR_TREE - recupera todos os contornos e reconstrói a hierarquia dos contornos

method

Aproximação método (para todos os modos, exceto CV_RETR_RUN).

- CV_CHAIN_CODE – contornos de saída na cadeia Freeman. Todos os outros métodos de saída são polígonos (seqüências de vértices).
- CV_CHAIN_APPROX_NONE - traduz todos os pontos da cadeia de código em pontos;
- CV_CHAIN_APPROX_SIMPLE - compress horizontais, verticais e diagonais segmentos, ou seja, a função deixa apenas os seus pontos que termina;
- CV_CHAIN_APPROX_TC89_L1,
- CV_CHAIN_APPROX_TC89_KCOS - aplica um dos modelos de cadeia do Teh-Chin.
- CV_LINK_RUNS - uso completamente diferente do contorno, ligando a recuperação do algoritmo através de segmentos horizontais de 1's. Só CV_RETR_LIST Modo de recuperação possam ser utilizados com este método.

offset

Offset, através do qual cada ponto do contorno seja transferido. Isto é útil se os contornos são extraídos da imagem ROI e, em seguida, eles devem ser analisados no contexto em toda a imagem.

- **cvContourArea**
  - *Descrição*

A função cvContourArea calcula a área de todo o contorno.

  - *Declaração*

double cvContourArea(const CvArr* contour, CvSlice slice=CV_WHOLE_SEQ);

- *Parâmetros*

contour

Contorno

slice

Início e fim dos pontos de contorno da seção de interesse, por padrão toda a área do contorno é calculada.

❖ **cvBoundingRect**

- *Descrição*

A função cvBoundingRect retorna o up-direita, delimita o retângulo para definir ponto 2d.

- *Declaração*

CvRect cvBoundingRect(CvArr* points, int update=0);

- *Parâmetros*

points

Ou um ponto fixado em 2D, representada como uma seqüência (CvSeq *, CvContour *) ou vetor (CvMat *) dos pontos, ou 8 bits de canal único máscara imagem (CvMat *, IplImage *), no qual não são zero pixels considerado.

update

A atualização do flag. Aqui está lista de possíveis combinações de valores do flag e do tipo de contorno:

- pontos é CvContour *, atualização = 0: delimita o retângulo; não está calculado, mas trata-se de ler o contorno rect campo de cabeçalho.
- pontos é CvContour *, atualização = 1: delimita o retângulo; é calculado e escrito para o contorno rect campo de cabeçalho.
- pontos é CvSeq ou CvMat * *: atualização é ignorada, delimita o retângulo; é calculado e devolvido.

❖ **cvcamSelectCamera**

- *Descrição*

Esta função mostra uma caixa de diálogo, para selecionar um ou 2 câmeras. Se não é NULL, o usuário tem alguma câmera selecionada (s), o leque do índice é colocado em * out.

- *Declaração*

int cvcamSelectCamera(int** **out**);

- *Parametros*

int** **out** (out) – um endereço de um ponteiro para int ou NULL.
Se não for NULL irá conter uma série de índices selecionados(câmeras).

- *Valor retornado*

O número de câmeras selecionadas - 0, 1 ou 2.

- **cvcamSetProperty**

  - *Descrição*

Define o valor da propriedade especificada da câmara para o valor especificado.

  - *Declaração*

int cvcamSetProperty(int camera, const char* property, void* value);

  - *Parametros*

int **camera** (in) – um número de 0 ao numero de câmera na base de índice encontrado no sistema.
const char* **property** (in) – nome da propriedade

void* **value** (in) – depended a propriedade e do nome.

  - *Valor retornado*

0 com sucesso, erro negativo código de erro.

- **cvcamInit**

  - *Descrição*

Esta função torna as definições da cvcamSetProperty ativa e faz o último passo da inicialização da camera. Não pode ser confundida - não é a primeira função, é chamada. Assume-se que cvcamGetCamerasCount já foi chamada e, geralmente, que algumas propriedades foram estabelecidas.

- *Declaração*

int cvcamInit();

- *Parametros*

Nenhum.

- *Valor Retornado*

1 com sucesso, 0 com erro.

❖ **cvcamStart**

- *Descrição*

Inicia o vídeo para todas as câmeras ativadas.

- *Declaração*

int cvcamStart();

- *Parametros*

Nenhum.

- *Valor Retornado*

0 com sucesso, -1 com falha.

❖ **cvcamStop**

- *Descrição*

Faz Parar o vídeo.

- *Declaração*

int cvcamStop();

- *Parametros*

Nenhum.

- *Valor Retornado*

Sempre 0.

❖ **cvcamExit**

- *Descrição*

Libera todos os recursos usados pela cvcam.

- *Declaração:*

int cvcamExit();

- *Parametros*

Nenhum.

- *Valor Retornado*

Sempre 0.