UNICEUB – CENTRO UNIVERSITÁRIO DE BRASÍLIA
FAET – FACULDADE DE CIÊNCIAS EXATAS E TECNOLOGIA
CURSO DE ENGENHARIA DA COMPUTAÇÃO

RODRIGO BENIN RIBEIRO

ARMA SENTINELA:

Segurança de ambientes restritos através de arma de fogo com mira e disparo microcontrolados, utilizando processamento e análise de imagens

BRASÍLIA/DF
1º SEMESTRE DE 2007

RODRIGO BENIN RIBEIRO

ARMA SENTINELA:

Segurança de ambientes restritos através de arma de fogo com mira e disparo microcontrolados, utilizando processamento e análise de imagens

Monografia apresentada ao Curso de Engenharia da Computação, como requisito parcial para obtenção do grau de Engenheiro de Computação.

Orientador: Prof. M. C. Claudio Penedo

BRASÍLIA/DF

1º SEMESTRE DE 2007

Resumo

Neste projeto é apresentada a construção de um protótipo eletrônico e software para o direcionamento ao alvo (mira) e acionamento automático do gatilho de uma arma de fogo. É utilizada uma câmera estática que focaliza o ambiente a ser monitorado, sendo efetuada análise das imagens adquiridas para detectar movimento de um possível alvo e movimentar os servomotores ligados à arma, de forma a rastrear o alvo. Quando da presença de um alvo (qualquer objeto em movimento) o mecanismo passa a “perseguir” o alvo e efetua os disparos.

Palavras-Chave: Detecção de Movimento, Monitoração de Ambientes, Arma Sentinela, Servomotor, Alvo, Tiro, Processamento de Imagem, Sistemas de Vigilância

Abstract

In this project is presented the construction of electronic prototype and software for automatic aiming and firing of a gun. It is utilized a static camera that focuses the environment to be monitored, performing analysis of the images acquired for movement detection and to move the servomotors connected to the weapon to track the target. On the occasion of the presence of a target (any moving object) the mechanism begins to "pursue" it and performs the shooting.

Keywords: Movement Detection, Environment Monitoring, Sentry Gun, Servomotor, Target, Shot, Image Processing, Surveillance System.

Agradecimentos

Agradecer é um ato de amor. É o reconhecimento da importância das pessoas que nos cercam e deveria ser um exercício diário. É imbuído desse espírito que gostaria de agradecer a todos que, de alguma forma, contribuíram para o desfecho dessa grande etapa da minha vida. Nomear todos seria uma tarefa gratificante e um bom passeio pela memória desses sete anos de luta. Porém, mesmo correndo o risco de ser injusto, o farei de forma resumida.

Primeiramente a Deus, por me dar a chance de pagar meus pecados e por estar sempre ao meu lado.

Sou muito grato a todos os professores que exerceram seu dever com dedicação e paciência.

Aos colegas de classe e de faculdade, tanto de Sorocaba quanto de Brasília, pelo companheirismo e pelas inspirações: Diogo Curto, Alexandre “Mangaba” Santos, Michel Calheiros, Gustavo “Capitão” de Faria, Nelmar Pássaro, Tafarel, Alex “Lampadinha”, Renatão, Renatinho, Luiz “Trakinas”, José “Zé” Pereira Jr., Thiago “Japa” Suguiama, Thiago “Thiaguera” Volpon, Jeison “Lee” e a todos os demais.

Aos meus sócios Diogo Curto e Luiz Miamoto por aturarem minha ausência durante a execução deste projeto.

A minha esposa Erika por me tirar dos momentos de imersão na hora certa, por ser meu alento e por me apoiar sempre. Amo-te.

Ao meu filho Caio por ter sido meu “alvo” número um durante a fase de testes e por fazer perguntas de toda natureza sobre o projeto, me preparando para a banca (se consigo explicar para uma criança de três anos, explico para qualquer um!). Você é minha luz, “um sorriso de Deus”.

Aos meus pais: por terem me ensinado o valor da educação e do trabalho, por me animarem nos momentos difíceis, por sobejarem tudo que um filho pode esperar de seus pais. Exemplo, fé, porto seguro.

Em especial, agradeço meu orientador Claudio Penedo, por me direcionar de forma correta, por sua competência e disponibilidade. Este trabalho é nosso.

"Nos delitos contra a propriedade, particular, ou pública, não se engravesce ou aligeira o caráter do crime com o ser de maior ou menor grandeza a importância do dano causado, ou de soma subtraída. A malversação não avulta, nem míngua, com a maior ou menor monta dos bens malversados. Destarte se pronuncia a lei escrita; e não me consta que reze de outro modo algum sistema de moral, salvo o contemplado na ironia do provérbio, e segundo o qual quem pouco furta é ladrão, quem muito furta, barão".

Rui Barbosa

Sumário

1 INTRODUÇÃO 1
1.1 Contextualização do Trabalho 1
1.2 Objetivo do Projeto 2
1.3 Motivação 4
1.4 Estrutura do Trabalho 4
2 CONCEITOS BÁSICOS DE PROCESSAMENTO DE IMAGENS 6
2.1 O Sistema de Cores RGB 6
2.2 Escala de Cinza 7
2.3 Processamento de Imagens 7
2.3.1 Definições Básicas 8
2.3.1.1 Imagem 8
2.3.1.2 Imagem Digital 8
2.3.1.3 Pixel 9
2.3.2 Algoritmos de Processamento de Imagem 10
2.3.2.1 Operações Matemáticas 10
2.3.2.2 Operações de Amaciamento (*Smoothing Operations*) 13
2.3.2.3 Operações Baseadas em Morfologia Matemática 20
2.3.3 Técnicas de Segmentação de Imagens 24
2.3.3.1 Limiar (*Thresholding*) 25
2.3.3.2 Bordas (*Edges*) 29
3 DETECÇÃO DE MOVIMENTO POR ANÁLISE DE IMAGENS 35
3.1 Preâmbulo 35
3.2 Método: Diferença entre dois quadros sucessivos 36
3.3 Método: Diferença entre o quadro atual e uma imagem com o cenário de fundo armazenado 37
4 DESCRIÇÃO DO *HARDWARE* 40
4.1 Diagrama de Blocos 40
4.2 A Câmera 41
4.3 Porta Serial 41
4.3.1 Comunicação com RS-232 41
4.4 A Interface (Placa Controladora) 43

4.5 Os Servomotores 44
5 DESCRIÇÃO DO *SOFTWARE* 46
5.1 Tecnologia Utilizada 46
5.2 AForge.Net *Framework* 46
5.3 Módulo de Detecção de Movimento 47
5.3.1 Aquisição de Imagens 49
5.3.1.1 Interface *IVideoSource* 49
5.3.1.2 Classe *CaptureDevice* 49
5.3.1.3 Classe *Camera* 49
5.3.2 Pré-processamento de Imagens 50
5.3.3 Separação do cenário de fundo da imagem 50
5.3.4 Aplicação dos filtros e algoritmos para detecção de movimento, identificação e posicionamento dos objetos em cena 50
5.3.4.1 Interface *IMotionDetector* 51
5.3.4.2 Classe *MotionDetector5* 52
5.3.5 Pós-processamento de imagens 52
5.4 Módulo de Controle da Arma 52
5.4.1 Análise e Monitoração de Alvos 55
5.4.2 Comunicação com a Placa Controladora 55
5.4.3 Movimentação dos Servos 55
5.5 Camada de Apresentação 56
5.5.1 Inicialização da Placa 57
5.5.2 Calibração dos Motores 57
5.5.3 Seleção do Algoritmo de Detecção de Movimento 58
5.5.4 Comandos de Inicialização da Câmera 58
5.5.5 Janela de Monitoração 59
5.5.6 Interface de Geração de Comandos Manuais 60
5.5.7 Comandos de Controle da Arma 61
6 CONSIDERAÇÕES FINAIS 62
6.1 Dificuldades Encontradas 62
6.2 Resultados Obtidos 63
6.3 Conclusões 64
6.4 Sugestões de Trabalhos Futuros 64
REFERÊNCIAS 66

APÊNDICE A – CÓDIGO FONTE DO SOFTWARE CONTROLADOR 68
APÊNDICE B – FOTOS REAIS DO DISPOSITIVO 171

Índice de Figuras

Figura 1.1 – Modelo de arma sentinela 3
Figura 2.1 - Modelo RGB mapeado para um cubo em corte 6
Figura 2.2 - Digitalização de uma imagem contínua. O valor inteiro de brilho do pixel nas coordenadas [m=10, n=3] é 110 9
Figura 2.3 - Exemplo de operações binárias em pontos..... 12
Figura 2.4 - Exemplo de filtro de Diferença 13
Figura 2.5 - Quatro regiões quadradas definidas pelo filtro Kuwahara..... 19
Figura 2.6 - Ilustração de filtros lineares e não-lineares de amaciamento..... 19
Figura 2.7 - Uma imagem binária contendo duas coleções de pontos de objeto A e B 20
Figura 2.8 - Uma imagem binária contendo dois objetos A e B. Os 3 pixels em B estão hachurados assim como seu efeito no resultado..... 22
Figura 2.9 - Elementos de estrutura padrão N_4 e N_8 23
Figura 2.10 - Ilustração de dilatação. Pixels originais de objeto estão em cinza; pixels adicionados por dilatação estão em preto. 23
Figura 2.11 - Exemplo de operações matemáticas de morfologia..... 24
Figura 2.12 - Imagem de entrada e histograma de brilho..... 27
Figura 2.13 - Ilustração do algoritmo do triângulo 29
Figura 2.14 - Bordas baseadas em gradiente, combinado com limiar/isodata 30
Figura 2.15 - Bordas baseado no cruzamento de zeros obtido do laplaciano da imagem..... 31
Figura 2.16 - Filtro do “Chapéu Mexicano” 32
Figura 2.17 - Estratégia geral para "Bordas" baseado em cruzamentos de zeros ... 34
Figura 2.18 - Detecção de bordas usando algoritmos LoG e Soma..... 34
Figura 3.1 - Ilustração do método de diferença entre o quadro atual e uma imagem com o cenário de fundo armazenado 38
Figura 4.1 - Diagrama de blocos do protótipo..... 40
Figura 4.2 – Foto da placa controladora / Conector para porta serial..... 44
Figura 4.3 - Servomotores utilizados no projeto 44
Figura 4.4 – Pinos de conexão dos servomotores com a placa controladora..... 45

Figura 5.1 – Seqüência de passos executados pelo módulo de detecção de movimento 48
Figura 5.2 - Classes principais do módulo de detecção de movimento 48
Figura 5.3 – Seqüência de passos executados pelo módulo de detecção de movimento para detecção de movimento, identificação e posicionamento dos objetos em cena 51
Figura 5.4 – Seqüência de passos executados pelo módulo de controle da arma ... 53
Figura 5.5 – Classes principais do módulo de controle da arma 54
Figura 5.6 - Interface principal da camada de apresentação 56
Figura 5.7 - Botões para controle da câmera 58
Figura 5.8 - Interface para seleção de câmera 58
Figura 5.9 - Interface principal exibindo alvo identificado 59
Figura 5.10 - Botão de acionamento da interface de comandos manuais 60
Figura 5.11 - Interface de comandos manuais - Aba Configurações 60
Figura 5.12 - Interface de comandos manuais - Aba Controle dos Servos 61
Figura B.1 – Foto da arma: vista geral 171
Figura B.2 – Foto da arma: vista lateral 172
Figura B.3 – Foto da arma: vista lateral 172
Figura B.4 – Foto da arma: vista posterior 173
Figura B.5 – Foto da arma: vista frontal 174
Figura B.6 – Foto da arma: vista da base 174
Figura B.7 – Foto da arma: placa controladora 175
Figura B.8 – Foto da arma: servo do eixo X 175

Índice de Tabelas

Tabela 2.1 - Definição das operações binárias 11
Tabela 2.2 - PSF e Função de Transferência para o Caso Retangular 14
Tabela 2.3 - PSF e Função de Transferência para o Caso Circular 14
Tabela 4.1 - Pinos para comunicação serial 42

Lista de Equações

Equação 2.1 – Operação matemática OR..... 11
Equação 2.2 - Operação matemática NOT..... 11
Equação 2.3 - Operação matemática XOR..... 11
Equação 2.4 - Operação matemática AND..... 11
Equação 2.5 - Operação matemática SUB..... 11
Equação 2.6 – Filtro uniforme, caso retangular..... 15
Equação 2.7 – Filtro uniforme, caso circular..... 15
Equação 2.8 - Filtro piramidal (triangular com suporte retangular)..... 16
Equação 2.9 - Filtro cônico (triangular com suporte circular)..... 16
Equação 2.10 – Identificação de objeto por compartilhamento de propriedade comum..... 20
Equação 2.11 – Complemento de um objeto (fundo)..... 21
Equação 2.12 – Soma de Minkowski..... 21
Equação 2.13 - Subtração de Minkowski..... 21
Equação 2.14 – Operação de dilatação..... 22
Equação 2.15 – Operação de erosão..... 22
Equação 2.16 – Operação de abertura..... 23
Equação 2.17 – Operação de fechamento..... 23
Equação 2.18 – Cálculo de valor de limiar por simetria de fundo..... 28
Equação 2.19 – Cruzamentos em por Laplaciano do Gaussiano..... 32
Equação 3.1 – Diferença de imagens por dois quadros sucessivos..... 36
Equação 3.2 – Diferença de imagens por quadro atual e cenário de fundo armazenado..... 38

Glossário de Termos e Abreviaturas

Log – Registro persistido de informações de ação ou evento ocorrido em um sistema computacional.

LoG – Laplaciano do Gaussiano.

OTF – *Optical Transfer Function* – Função de Transferência Óptica

Pixel – *Picture Element* – Elemento da Figura, considerado como cada ponto de uma imagem.

PSF – *Point Spread Function* – Função de espalhamento de ponto

RGB – *Red, Green, Blue* – Vermelho, Verde, Azul.

SNR – *Signal-to-Noise Ratio* – Relação sinal-ruído.

USB – *Universal Serial Bus* - Tipo de conexão que permite a ligação de periféricos ao computador sem a necessidade de desligá-lo.

Lista de Símbolos

θ – Valor de limiar para filtros de amaciamento.

σ - Valor de limiar para filtro de amaciamento Gaussiano

λ - Valor de cor de uma imagem

t – Medida de tempo

L – Valor de um nível da cor cinza em uma imagem

r – Raio de uma circunferência

j – Dimensão das linhas de uma matriz de pontos

k – Dimensão das colunas de uma matriz de pontos

m – Coordenada do eixo X de uma representação discreta de uma imagem

n – Coordenada do eixo Y de uma representação discreta de uma imagem

O – Número de operações realizadas para execução de um algoritmo

$\mathbf{p_{máx}}$ – Valor máximo de pico em um histograma de brilho

1 INTRODUÇÃO

1.1 Contextualização do Trabalho

A segurança e a integridade física do indivíduo, bem como de suas respectivas instalações, são preocupações sociais que influenciam fortemente o dia-a-dia das pessoas e seus comportamentos. Pesquisas nesse campo são potencialmente promissoras, considerando as mazelas sociais causadas pelo alto índice de violência urbana e pelo alto nível tecnológico de armamento e de instrumentação dos criminosos.

A idéia de se ter sistemas de vigilância menos suscetíveis a falhas de natureza humana implica em um campo fértil e promissor para pesquisas. Sistemas autômatos e controlados computacionalmente estão cada vez mais presentes no mercado e em constante desenvolvimento.

A análise da movimentação humana tornou-se muito importante e aplicável na área de comunicação visual e de realidade virtual. Especialmente, a avaliação da postura humana em tempo real é importante para muitas aplicações como jogos eletrônicos, diversão, segurança e avançados sistemas de interface homem-máquina (TAKAHASHI, 1999).

Usualmente, sistemas de captura de movimento que utilizam dispositivos mecânicos, eletromagnéticos ou acústicos apresentam uma série de problemas, tais como a necessidade do uso de equipamentos especiais e a ocorrência de perda de dados de movimento. O presente trabalho utiliza uma alternativa óptica de captura de movimento, constituída das seguintes etapas: (1) processo de captura dos dados de câmeras de vídeo; (2) remoção do fundo da cena e detecção de objetos em movimento e (3) análise da silhueta dos objetos para se ter a exata localização de seu posicionamento.

1.2 Objetivo do Projeto

Neste trabalho é apresentado um projeto acadêmico de mira e acionamento do gatilho de uma arma de fogo, de forma automática, através do computador, realizando análise de imagem e servo-controle. Será utilizada uma câmera estática, que focalizará o ambiente a ser monitorado. Quando da presença de um alvo (qualquer objeto em movimento) o mecanismo passa a rastrear o alvo e efetua os disparos.

Primeiramente, uma câmera captura a imagem. Um computador executando uma aplicação de controle interpreta esta imagem e decide o que mirar. Baseado na calibração dos servos, o computador envia um sinal para o controlador de servos, que então move três servomotores conectados a arma. Estes servomotores estão conectados a arma da seguinte forma:

a) Um servomotor na base da arma realizando os movimentos no eixo x (horizontal);
b) Um servomotor na base da arma realizando os movimentos no eixo y (vertical) e;
c) Um servomotor acoplado ao gatilho da arma realizando os movimentos de pressão (disparos).

São desenvolvidos neste projeto, além do protótipo, o software de reconhecimento de imagem e de movimentação dos servomotores. A Figura 1.1 mostra um esquemático do funcionamento do protótipo.

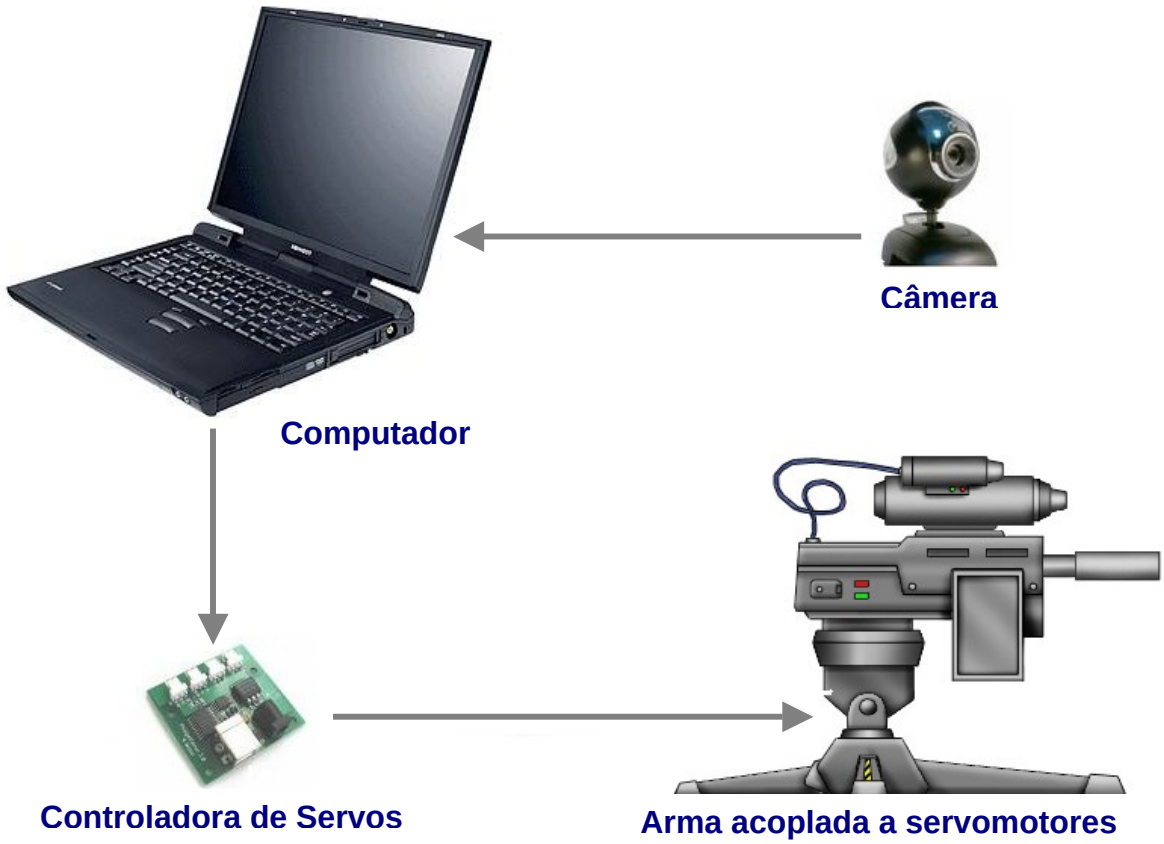


Figura 1.1 – Modelo de arma sentinela

Obviamente um mecanismo como este não seria utilizado sozinho, mas sim como parte de um circuito de segurança maior. Porém, por questões de viabilidade acadêmica e técnica, o projeto ater-se-á ao escopo menor proposto.

Este trabalho não se ocupará de políticas de segurança ou seqüência de eventos que antecedam a identificação do alvo uma vez que este atinja o “campo de visão” do mecanismo. Portanto, avisos, alertas ou sistemas de acesso seguro através do ambiente monitorado não serão considerados como escopo deste trabalho.

1.3 Motivação

As aplicações bélicas de um modelo como o proposto são variadas, como a proteção de ambientes restritos, sem a necessidade de patrulha humana ostensiva. Um exemplo prático seria um depósito de armas bélicas que deve ser monitorado ostensivamente, impedindo o acesso de pessoas não autorizadas. Neste caso, o mecanismo pode efetuar disparos letais ou apenas atingir regiões que não causem a morte do alvo (pernas, por exemplo), conforme a política de segurança adotada.

Em casos onde a corruptibilidade dos agentes humanos mereça consideração, pode haver ganho de confiabilidade com este tipo de sistema, uma vez que o ambiente computacional possui vários níveis de proteção e rastreamento de ações (*logs*).

Exemplos como os citados anteriormente e as possibilidades de aplicação vislumbradas são a principal motivação para o desenvolvimento deste projeto. Além disso, após alguma pesquisa de mercado, percebe-se que ainda há poucos sistemas desta natureza desenvolvidos.

1.4 Estrutura do Trabalho

Além deste capítulo introdutório, este trabalho está estruturado em seis capítulos assim distribuídos:

No **Capítulo 2** é apresentada uma introdução teórica sobre processamento de imagens, onde são abordados os conceitos essenciais para o entendimento do funcionamento do protótipo e, principalmente, do software desenvolvido.

No **Capítulo 3** são apresentados dois métodos de detecção de movimento realizando processamento de imagens, de forma comparativa, ressaltando a abordagem utilizada e os motivos de sua escolha.

No **Capítulo 4** é detalhada a construção do protótipo, descrevendo as etapas e justificativas de montagem. São também caracterizados os principais componentes que compõem o protótipo e o papel de cada um no sistema como um todo.

No **Capítulo 5** é apresentada a construção do Software Controlador, responsável por detectar o movimento do possível alvo através das imagens e movimentar os servomotores ligados à arma. São especificados as tecnologias utilizadas e os motivos de escolha de cada uma. Este capítulo contém ainda a descrição das funcionalidades principais dos módulos da aplicação.

Por fim, no **Capítulo 6** são apresentadas as considerações finais sobre o trabalho, contendo as principais conclusões, os resultados obtidos, as dificuldades encontradas e as sugestões para trabalhos futuros.

2 CONCEITOS BÁSICOS DE PROCESSAMENTO DE IMAGENS

Neste capítulo são abordados conceitos teóricos sobre processamento de imagens que são pertinentes ao entendimento do projeto desenvolvido. Embora alguns dos conceitos citados envolvam desenvolvimentos matemáticos aprofundados, demandando um detalhamento mais extenso do tema para sua completa compreensão, apenas as características principais serão apresentadas, visando manter o foco de interesse do presente trabalho que, embora sirva-se destes conceitos, não tem seu foco principal centrado neles.

2.1 O Sistema de Cores RGB

RGB (*red, green, blue*) refere-se a um sistema para representar as cores usadas numa exposição de imagem via computador. Vermelho, verde, e azul podem ser combinados em várias proporções para se obter qualquer cor no espectro visível. Os níveis de R, G, e B podem variar de 0 a 100 por cento de plena intensidade. Cada nível é representado por um número no intervalo de números decimais de 0 a 255 (256 níveis para cada cor), equivalente ao intervalo de números binários de 00000000 a 11111111, ou para hexadecimal 00 a FF. O número total de cores disponíveis é 256 x 256 x 256, ou 16.777.216 cores possíveis. (WHATIS, 2007). A Figura 2.1 ilustra um mapeamento da combinação das três cores do sistema RGB (vermelho, verde e azul).

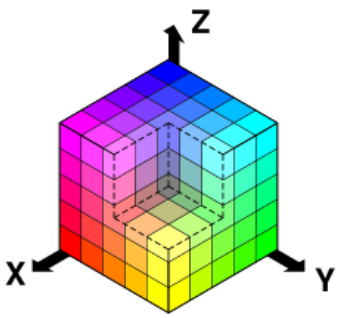


Figura 2.1 - Modelo RGB mapeado para um cubo em corte

2.2 Escala de Cinza

Em computação, imagens exibidas em escala de cinza ou *grayscale*, são imagens onde o valor de cada pixel é uma única amostra. No sistema RGB isto quer dizer que a intensidade de cada uma das três cores (*red, green e blue*) é a mesma.

As imagens geradas nesta escala são compostas por máscaras de cinza que variam do preto (a intensidade mais fraca) ao branco (a intensidade mais forte).

As imagens da escala de cinza são diferentes das imagens em preto-e-branco, que no contexto da imagem latente do computador são imagens com somente duas cores, o preto e o branco; já as imagens em *grayscale* possuem muitas variações de cores entre o preto e o branco.

2.3 Processamento de Imagens

A tecnologia digital moderna tornou possível a manipulação de sinais multidimensionais através de sistemas que variam desde simples circuitos digitais a avançados computadores paralelos. Os objetivos dessa manipulação podem ser divididos em três categorias (YOUNG, 2004):

a) Processamento de Imagem (entrada: imagem, saída: imagem);
b) Análise de Imagem (entrada: imagem, saída: medidas da imagem) e;
c) Compreensão de Imagem (entrada: imagem, saída: descrição de alto nível da imagem).

Esta seção concentra-se nos conceitos fundamentais de processamento de imagem, restritos a imagens bidimensionais (2D), embora a maioria dos conceitos e técnicas que serão descritas possam ser adaptadas para três ou mais dimensões sem prejuízo de conceito (YOUNG, 2004).

2.3.1 Definições Básicas

2.3.1.1 Imagem

Uma imagem pode ser definida como uma função de duas variáveis reais *a(x,y)* onde *a* é a amplitude (de brilho, por exemplo) da imagem nas coordenadas *x,y*. Uma imagem pode conter sub-imagens que normalmente são chamadas de *região de interesse*, ou simplesmente *região*. Este conceito reflete o fato de que imagens freqüentemente contêm coleções de objetos os quais podem ser individualmente a base para uma região. Em sistemas sofisticados de processamento podem-se aplicar operações de processamento de imagens em regiões específicas (YOUNG, 2004).

A amplitude de uma imagem é geralmente representada por números reais ou inteiros, embora em casos como representação por imagem de ressonância magnética, a medida física direta forneça um número complexo na forma de uma magnitude real e uma fase real. Neste trabalho, só são considerados casos de amplitudes reais ou inteiras (YOUNG, 2004).

2.3.1.2 Imagem Digital

Uma imagem digital *a[m,n]* descrita em um espaço discreto 2D é derivada de uma imagem analógica *a(x,y)* em espaço contínuo 2D, através de uma processo matemático de amostragem, quantificação e ou codificação comumente chamado de *digitalização* (YOUNG, 2004).

2.3.1.3 Pixel

A imagem contínua em 2D *a(x,y)* é dividida em *N* linhas e *M* colunas. A intersecção de uma linha e uma coluna é chamada de *pixel*. O valor atribuído às coordenadas inteiras *[m,n]* com *{m=0,1,2,...,M-1} e {n=0,1,2,...,n-1}* é representado pela função *a[m,n]*. De fato, na maioria dos casos, *a(x,y)* - que deve ser considerada como o sinal físico na face de um sensor 2D – é uma função de muitas variáveis, como profundidade *(z),* cor *(λ)* e tempo(*t)*. Em todo este capítulo são consideradas apenas imagens 2D, monocromáticas e estáticas (YOUNG, 2004). A Figura 2.2 ilustra o processo de digitalização de uma imagem contínua.

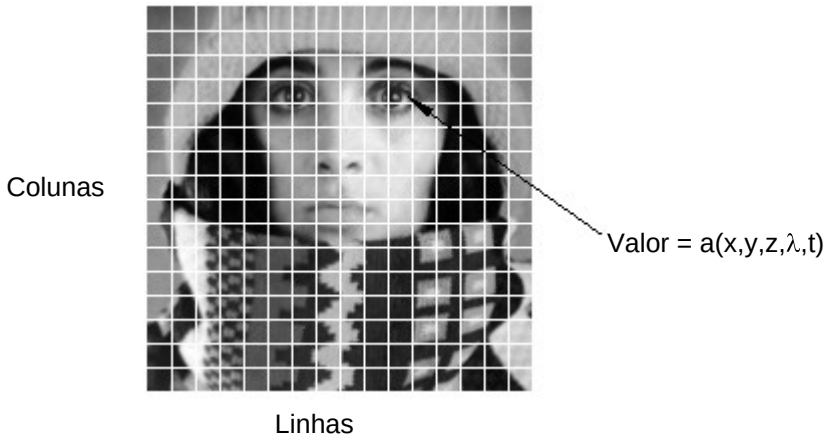


Figura 2.2 - Digitalização de uma imagem contínua. O valor inteiro de brilho do pixel nas coordenadas [m=10, n=3] é 110

A imagem mostrada na Figura 2.2 foi dividida em 16 linhas e 16 colunas. O valor atribuído a cada pixel é a média do valor de brilho do pixel, arredondado para o inteiro mais próximo. O processo de se representar a amplitude de um sinal 2D em uma dada coordenada através de um inteiro, que no caso representa um valor de *L* diferentes níveis de cinza, é usualmente chamado de *quantização de amplitude*, ou simplesmente *quantização* (YOUNG, 2004).

2.3.2 Algoritmos de Processamento de Imagem

As operações fundamentais para processamento de imagens digitais são normalmente divididas em quatro categorias (YOUNG, 2004):

a) Operações baseadas no histograma da imagem;
b) Operações baseadas em matemática simples;
c) Operações baseadas em convolução e;
d) Operações baseadas em morfologia matemática.

Estas operações podem ainda ser classificadas quanto à natureza de sua implementação como sendo: operações aplicadas ponto-a-ponto, operações locais, ou operações globais (YOUNG, 2004).

Neste trabalho serão abordados apenas os algoritmos e/ou operações destes grupos que foram utilizados ou os que se fazem necessários para o entendimento e contextualização teórica de seu desenvolvimento, sendo os mesmos: operação matemática de diferença, operações de amaciamento (que conforme as implementações categorizam-se em grupos de operações diferentes) e operações morfológicas de erosão, dilatação, abertura e fechamento.

2.3.2.1 Operações Matemáticas

As operações matemáticas baseadas em aritmética binária (Booleanos) constituem a base de uma série de ferramentas para processamento de imagens. As operações descritas a seguir (Equações 2.1, 2.2, 2.3, 2.4 e 2.5) são operações aplicadas para cada ponto da imagem (pixel a pixel) e admitem uma variedade de implementações eficientes, incluindo tabelas de pesquisa simples (YOUNG, 2004).

$$OR \quad c = a + b \tag{2.1}$$

$$NOT \quad c = \overline{a} \tag{2.2}$$

$$XOR \quad c = a \oplus b = a \bullet \overline{b} + \overline{a} \bullet b \tag{2.3}$$

$$AND \quad c = a \bullet b \tag{2.4}$$

$$SUB \quad c = a \setminus b = a - b = a \bullet \overline{b} \tag{2.5}$$

As Tabelas 2.1a, 2.1b, 2.1c, 2.1d e 2.1e contêm as definições das operações binárias aplicadas ponto a ponto.

Tabela 2.1 - Definição das operações binárias

NOT a	
0	1
1	0

↑ input ↑ output

(a)

OR a \ b	0	1
0	0	1
1	1	1

(b)

AND a \ b	0	1
0	0	0
1	0	1

(c)

XOR a \ b	0	1
0	0	1
1	1	0

(d)

SUB a \ b	0	1
0	0	0
1	1	0

(e)

A Figura 2.3 demonstra a aplicação das operações binárias aplicadas ponto a ponto nas imagens.

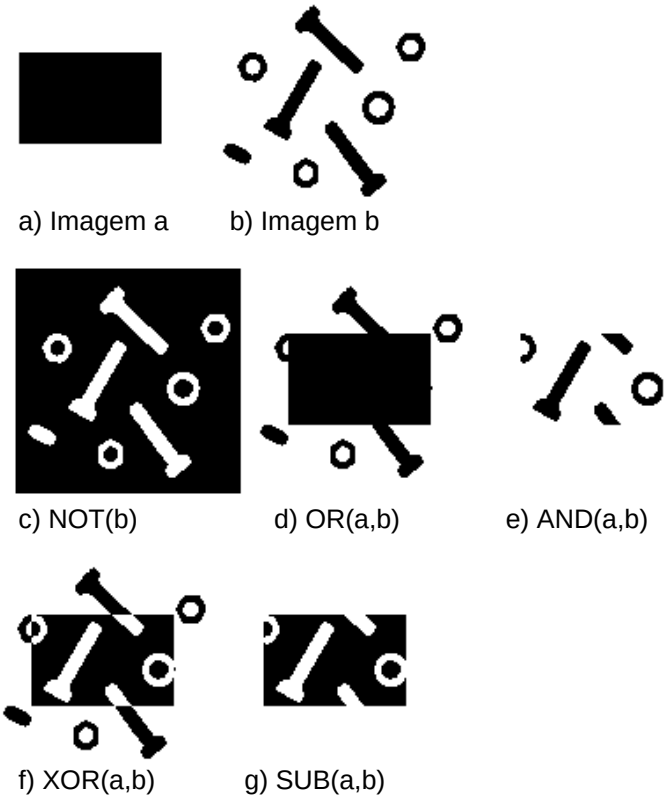
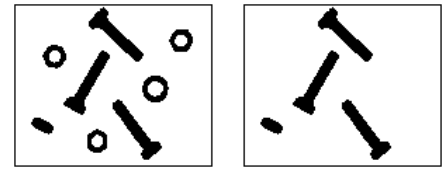


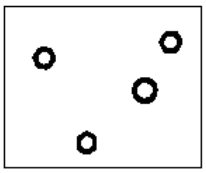
Figura 2.3 - Exemplo de operações binárias em pontos

2.3.2.1.1 Diferença

O filtro de diferença é composto da combinação das operações binárias simples NOT e AND, da forma *NOT(AND(a,b))*, que comparam duas imagens ponto-a-ponto e retornam como imagem de saída apenas os pixels que não coincidem em *a* e *b* (YOUNG, 2004). A Figura 2.4 exemplifica a aplicação de um filtro de diferença entre duas imagens a e b.



a) Imagem a b) Imagem b



c) Diferença(a,b)

Figura 2.4 - Exemplo de filtro de Diferença

Nota-se na Figura 2.4 que a operação compara ponto-a-ponto as imagens *a* e *b* retornando para cada ponto o resultado da operação *NOT(AND(a,b))*, gerando como imagem de saída apenas os pixels não coincidentes nas imagens de entrada.

2.3.2.2 Operações de Amaciamento (*Smoothing Operations*)

Estes algoritmos são aplicados para reduzir ruído ou para preparar imagens para processamento como, por exemplo, segmentação. São classificados em algoritmos lineares e não lineares. Também se distingue entre implementações baseadas num suporte retangular para o filtro e implementações baseadas num suporte circular para o filtro (YOUNG, 2004).

As Tabelas 2.2 e 2.3 mostram dois exemplos de sinais utilizados em filtros e suas respectivas transformadas de Fourier 2D. Por convenção, o termo no domínio do espaço é chamado de PSF (*Point Spread Function* ou Função de Espalhamento de Ponto) ou *resposta ao impulso 2D*. Suas respectivas transformadas de Fourier são denominadas OTF (*Optical Transfer Function* ou Função de Transferência Óptica). Dois sinais padrão utilizados nessas tabelas são a função degrau unitário *u(*)* e a função de Bessel do primeiro tipo $J_1(*)$. Na Tabela 2.2 a Imagem 2

corresponde à transformada de Fourier da Imagem 1, assim como na Tabela 2.3 a Imagem 4 corresponde à transformada de Fourier da Imagem 3.

Tabela 2.2 - PSF e Função de Transferência para o Caso Retangular

Retangular	$R_{a,b}(x,y) = \frac{1}{4ab}u\left(a^2 - x^2\right)u\left(b^2 - y^2\right)$	**F** $\Leftrightarrow$	$\left(\frac{sin\left(2\pi af_x\right)}{\pi af_x}\right)\left(\frac{sin\left(2\pi bf_y\right)}{\pi bf_y}\right)$
	Imagem 1		**Imagem 2**

Tabela 2.3 - PSF e Função de Transferência para o Caso Circular

Circular (Pill Box)	$P_a(r) = \frac{u\left(a^2 - r^2\right)}{\pi a^2}$	**F** $\Leftrightarrow$	$2\frac{\left(J_1 2\pi af\right)}{\pi af}$
	Imagem 3		**Imagem 4**

2.3.2.2.1 Filtros Lineares

Os filtros lineares distinguem-se dos não lineares por serem passíveis de análise no domínio de Fourier, enquanto os não lineares requerem estudos específicos para cada caso.

2.3.2.2.1.1 Filtro Uniforme

A imagem de saída de um filtro uniforme é baseada em uma média local do filtro de entrada, onde todos os valores dentro do suporte do filtro têm o mesmo

peso. Dois exemplos de filtros de entrada para filtragem uniforme são dados, no domínio espacial contínuo *(x,y)*, nas Tabelas 2.3 e 2.3, que contém a PSF e a função de transferência para o caso retangular e para o caso circular (*pill box*). Para o domínio espacial discreto *[m,n]* os valores de filtro são as amostras do caso do domínio contínuo. Os exemplos de filtros uniformes (funções de transferência) para o caso retangular (*J=K=5*) e para o caso circular (*R=2,5*) são mostrados nas Equações 2.6 e 2.7, respectivamente (YOUNG, 2004).

$$h_{ret}[j,k] = \frac{1}{25}\begin{bmatrix} 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \end{bmatrix} \tag{2.6}$$

A Equação 2.6 define um filtro uniforme retangular para amaciamento de imagem, com *j=k=5.*

$$h_{circ}[j,k] = \frac{1}{21}\begin{bmatrix} 0 & 1 & 1 & 1 & 0 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 1 & 1 & 1 & 1 & 1 \\ 0 & 1 & 1 & 1 & 0 \end{bmatrix} \tag{2.7}$$

A Equação 2.7 define um filtro uniforme circular para amaciamento de imagem, com *r=2,5.*

2.3.2.2.1.2 Filtro Triangular

A imagem de saída do filtro triangular é baseada numa média local do filtro de entrada, onde todos os valores dentro do suporte de filtro têm pesos divergentes. Em geral, o filtro pode ser visto como uma convolução de dois filtros uniformes idênticos, retangular ou circular (YOUNG, 2004).

Os exemplos para o caso de suporte retangular e para o caso de suporte circular são mostrados nas Equações 2.8 e 2.9, respectivamente (YOUNG, 2004).

$$h_{ret}[j,k] = \frac{1}{81}\begin{bmatrix} 1 & 2 & 3 & 2 & 1 \\ 2 & 4 & 6 & 4 & 2 \\ 3 & 6 & 9 & 6 & 3 \\ 2 & 4 & 6 & 4 & 2 \\ 1 & 2 & 3 & 2 & 1 \end{bmatrix} \tag{2.8}$$

A Equação 2.8 mostra um filtro piramidal (triangular com suporte retangular) com *j=k=5,* utilizado para amaciamento de imagem*.*

$$h_{circ}[j,k] = \frac{1}{21}\begin{bmatrix} 0 & 0 & 1 & 0 & 0 \\ 0 & 2 & 2 & 2 & 0 \\ 1 & 2 & 5 & 2 & 1 \\ 0 & 2 & 2 & 2 & 0 \\ 0 & 0 & 1 & 0 & 0 \end{bmatrix} \tag{2.9}$$

A Equação 2.9 mostra um filtro cônico (triangular com suporte circular) com *r=2,5,* utilizado para amaciamento de imagem*.*

2.3.2.2.1.3 Filtro Gaussiano

O uso do núcleo (*kernel*) Gaussiano para amaciamento tornou-se extremamente popular devido a suas propriedades, não só pelos teoremas como “limite central” e “produto mínimo de largura de banda de espaço”, mas também pela variedade de suas aplicações como, por exemplo, determinação de borda e análise na escala espacial (YOUNG, 2004).

Há quatro maneiras distintas de se implementar o Gaussiano, a saber (YOUNG, 2004):

a) Convolução usando um número finito de amostras do Gaussiano como sendo o núcleo de convolução.
b) Convolução repetitiva usando um filtro uniforme como sendo o núcleo de convolução.
c) Multiplicação no domínio de freqüência
d) Uso de uma implementação de filtro recursiva.

2.3.2.2.1.4 Outros Filtros

O domínio de Fourier possibilita a implementação de uma variedade de algoritmos de análise. Os filtros de amaciamento são filtros passa-baixo. Em geral, é desejável usar um filtro passa-baixo com fase zero para não se produzir deformidade de fase quando se aplicar o filtro a imagem (YOUNG, 2004).

2.3.2.2.2 Filtros Não Lineares

Existe uma grande variedade de filtros de amaciamento não lineares desenvolvidos. Como estes tipos de filtro não podem, em geral, ser submetidos à

análise de Fourier para a caracterização de suas propriedades e domínios de aplicação, são necessários estudos ostensivos para cada caso (YOUNG, 2004).

2.3.2.2.2.1 Filtro Mediano

Um filtro mediano baseia-se em mover uma janela sobre uma imagem (como numa convolução) e computar o pixel de saída como sendo o valor mediano dos brilhos dentro da janela de entrada. Se a janela é de tamanho *JxK* pode-se ordenar os pixels *J*K* por valor de brilho, de forma crescente. Se *J*K* é ímpar então a mediana será a entrada de ordem *(J*K+1)/2* na lista de brilhos. O valor selecionado será exatamente igual a um dos brilhos existentes, de modo que não ocorrem erros de arredondamento no caso de se trabalhar exclusivamente com valores inteiros de brilho. O algoritmo descrito anteriormente tem uma complexidade genérica por pixel de *O(J*K*log(J*K))*, onde *O* é o número de operações realizadas. Porém, existem algoritmos suficientemente rápidos que reduzem a complexidade a *O(K)* supondo *J>=K* (YOUNG, 2004).

Uma variação útil no tema do filtro mediano é o filtro de percentual. Neste filtro, o pixel do centro na janela não é substituído por 50% do valor de brilho (mediana), mas sim por *p%* do valor de brilho, onde *p%* varia de 0% (o filtro mínimo) a 100% (o filtro máximo). Os valores que não estão em (p=50)% em geral não correspondem a filtros de amaciamento (YOUNG, 2004).

2.3.2.2.2.2 Filtro de Kuwahara

A capacidade de se detectar bordas de objetos exerce um papel importante na percepção humana de imagens, assim como na análise computacional de imagens. Portanto, é importante que um filtro tenha a capacidade de alisar imagens sem perturbar a sua agudeza e, se possível, a posição das bordas. Um filtro que cumpre esta meta é denominado *filtro conservador de borda*. Um exemplo particular é o filtro de Kuwahara. Embora este filtro possa ser implementado para uma

variedade de formas diferentes de janela, o algoritmo será descrito para uma janela quadrada de tamanho *J=K=4L+1* onde *L* é um número inteiro. A janela é particionada em quatro regiões como demonstra a Figura 2.5 (YOUNG, 2004).

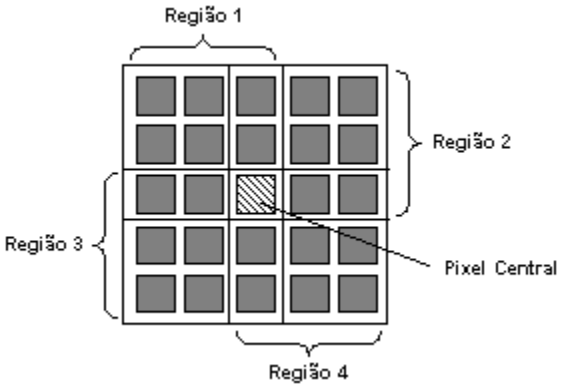
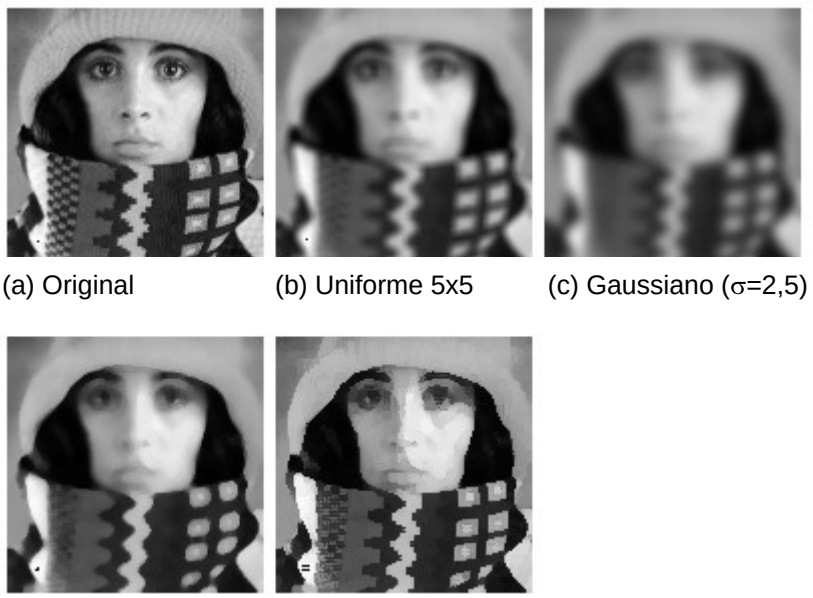


Figura 2.5 - Quatro regiões quadradas definidas pelo filtro Kuwahara

Neste exemplo L=1 .:. J=K=5. Cada região é [(J+1)/2] x [(K+1)/2].

A Figura 2.6 ilustra a aplicação dos tipos de filtros de amaciamento mencionados anteriormente e os seus efeitos sobre uma imagem.



(a) Original (b) Uniforme 5x5 (c) Gaussiano (σ=2,5)

(d) Mediano 5X5 (e) Kuwahara 5x5

Figura 2.6 - Ilustração de filtros lineares e não-lineares de amaciamento

2.3.2.3 Operações Baseadas em Morfologia Matemática

Geralmente se define uma imagem como uma função (amplitude) de duas coordenadas reais variáveis *a(x,y)* ou duas variáveis discretas *a[m,n].* Uma definição alternativa pode ser baseada no conceito de que uma imagem consiste de uma coleção de quaisquer coordenadas contínuas ou discretas. A coleção corresponde aos pontos (ou pixels) que pertencem aos objetos na imagem. Isto é ilustrado na Figura 2.7 que contém dois objetos ou coleções *A* e *B*, sob uma perspectiva binária (YOUNG, 2004).

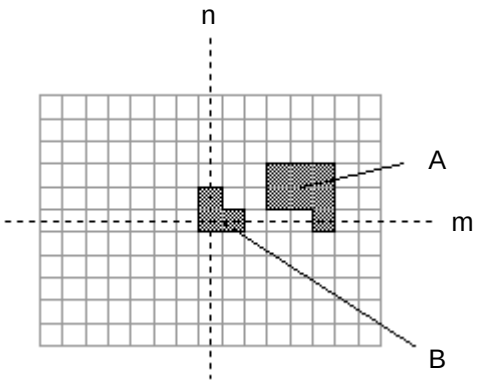


Figura 2.7 - Uma imagem binária contendo duas coleções de pontos de objeto A e B

Na Figura 2.7, o objeto *A* consiste dos pixels *a* que compartilham alguma propriedade comum, conforme Equação 2.10:

$$A = \{a \mid propriedade(a) = verdade\} \qquad (2.10)$$

Utilizando o exemplo anterior, o objeto *B* na Figura 2.7 consiste de {[0,0],[1,0],[0,1]}. O fundo de *A* é dado por A^C (o complemento de *A*) que é definido como os elementos que não estão em *A,* conforme Equação 2.11:

$$A^C = \{a \mid a \notin A\} \tag{2.11}$$

As operações fundamentais associadas a um objeto são as operações *união*, *intersecção*, e *complemento* {$\cap$,$\cup$,^{C}}, além da operação de *translação* (YOUNG, 2004).

A partir desses conceitos, pode-se definir as operações básicas de adição e subtração de Minkowski. Primeiramente, cabe notar que os elementos individuais que compõem *B* não são somente pixels, mas também vetores, já que possuem coordenadas de posição relativas a origem [0,0].

Dadas as coleções de pontos *A* e *B,* a adição de Minkowski é dada pela Equação 2.12 e subtração pela equação 2.13 (YOUNG, 2004).

$$A \oplus B = \bigcup_{\beta \in B} (a + \beta) \tag{2.12}$$

$$A \Theta B = \bigcap_{\beta \in B} (a + \beta) \tag{2.13}$$

2.3.2.3.1 Erosão e Dilatação

A partir das operações básicas de Minkowski definidas na seção 2.3.2.3, pode-se definir as operações básicas de morfologia Dilatação e Erosão (YOUNG, 2004):

$$Dilatação \rightarrow \quad D(A,B) = A \oplus B = \bigcup_{\beta \in B} (A + \beta) \tag{2.14}$$

$$Erosão \rightarrow \quad E(A,B) = A\Theta(-B) = \bigcap_{\beta \in B} (A - \beta) \tag{2.15}$$

onde $-B=\{-\beta|\beta \in B\}$. Estas duas operações estão ilustradas na Figura 2.8 para os objetos definidos na Figura 2.7 (YOUNG, 2004).

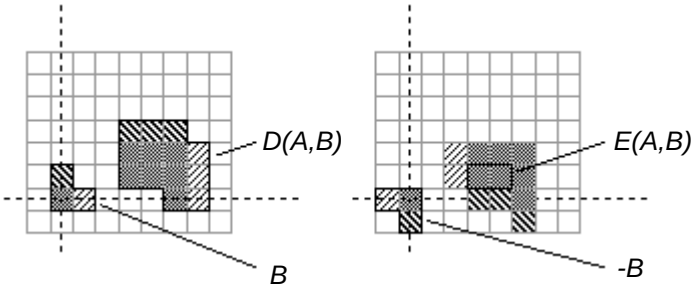


Figura 2.8 - Uma imagem binária contendo dois objetos A e B. Os 3 pixels em B estão hachurados assim como seu efeito no resultado.

Embora ambas as coleções de pontos *A* e *B* sejam imagens, *A* é usualmente considerada como *imagem* propriamente dita e *B* é chamado de *elemento de estrutura*. O elemento de estrutura representa para a morfologia matemática o que o núcleo de convolução representa para a teoria de filtros lineares (YOUNG, 2004).

A dilatação, em geral, provoca o aumento do volume dos objetos, enquanto a erosão causa o encolhimento. O “quanto” e o “como” os objetos crescem ou encolhem depende da escolha do elemento de estrutura. Os dois elementos de estrutura mais comuns para um plano cartesiano, são as coleções de “4 conectados” e “8 conectados”, N_4 e N_8, respectivamente (YOUNG, 2004). A Figura 2.9 mostra os elementos de estrutura padrão “4 conectados” e “8 conectados”.

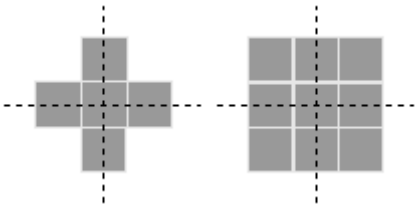
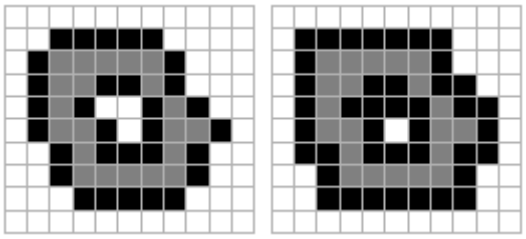


Figura 2.9 - Elementos de estrutura padrão N_4 e N_8

A Figura 2.10 mostra o efeito de dilatação sobre uma coleção de pixels para os casos de utilização dos elementos de estrutura padrão “4 conectados” e “8 conectados”.



a) B=N_4 b) B=N_8

Figura 2.10 - Ilustração de dilatação. Pixels originais de objeto estão em cinza; pixels adicionados por dilatação estão em preto.

2.3.2.3.2 Abertura e Fechamento

Combinando *Dilatação* e *Erosão*, obtêm-se duas operações para tratamento e identificação de objetos em imagens: *Abertura* e *Fechamento*, definidos pelas Equações 2.16 e 2.17 (YOUNG, 2004).

$$Abertura \rightarrow O(A,B) = A \circ B = D(E(A,B),B) \quad (2.16)$$

$$Fechamento \rightarrow C(A,B) = A \bullet B = E(D(A,-B),-B) \quad (2.17)$$

A Figura 2.11 exemplifica a aplicação das operações de dilatação, erosão, abertura e fechamento, todas em uma mesma imagem A.

a) Imagem A b) Dilatação de A c) Erosão de A

d) Abertura de A e) Fechamento de A

Figura 2.11 - Exemplo de operações matemáticas de morfologia

2.3.3 Técnicas de Segmentação de Imagens

Os algoritmos apresentados anteriormente podem ser combinados para resolver problemas específicos de processamento de imagem como, por exemplo, *correção de sombra*, *realce básico, técnicas de restauração* e *segmentação* (YOUNG, 2004).

Nesta seção estão descritas apenas as técnicas de segmentação utilizadas no desenvolvimento do projeto, que foram *limiar* e *bordas.*

Na análise de objetos em imagens é essencial que se possa distinguir entre os objetos de interesse e "o restante". Este último grupo também é referido como “fundo”. As técnicas que são usadas para encontrar os objetos de interesse normalmente são chamadas de técnicas de segmentação. Essas técnicas separam

o “primeiro plano” do “fundo” da imagem. É importante destacar que (YOUNG, 2004):

a) Não há uma técnica de segmentação universalmente aplicável que funcione para todas as imagens, e;
b) Nenhuma técnica de segmentação é perfeita.

2.3.3.1 Limiar (*Thresholding*)

Esta técnica é baseada em um conceito simples. Um parâmetro θ chamado de *limiar de brilho* é escolhido e aplicado à imagem *a[m,n]* segundo a seguinte lógica:

Se $a[m,n] \geq \theta$ *então* *a[m,n] = objeto = 1*
Senão *a[m,n] = fundo = 0*

Esta versão do algoritmo supõe que os objetos de interesse estão iluminados sob um fundo escuro. Para objetos escuros num fundo iluminado usa-se:

Se $a[m,n] < \theta$ *então* *a[m,n] = objeto = 1*
Senão *a[m,n] = fundo = 0*

A saída é o "objeto" de etiqueta ou "fundo" que, devido a sua natureza dicotômica, pode ser representado como uma variável Booleana "1" ou "0". Em princípio, a condição de prova pode ser baseada sobre outra propriedade que não o brilho simplesmente (por exemplo, *Se (Vermelhidão {a[m,n]} >= $\theta_{vermelho}$)*, sem prejuízo do conceito (YOUNG, 2004).

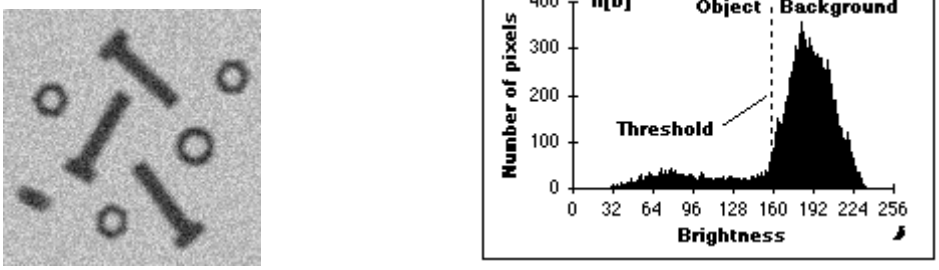
A questão central em torno dos processos de segmentação é como escolher o θ de limiar apropriado. Como não existe um procedimento universal para seleção de limiar que tenha funcionamento garantido com qualquer imagem, há uma variedade de alternativas. Na seqüência são abordadas as mais comumente utilizadas (YOUNG, 2004).

2.3.3.1.1 Limiar Fixo

Uma alternativa para escolha de um θ apropriado é utilizar um limiar que seja escolhido independentemente dos dados de imagem. Se for sabido que se lida com imagens de alto-contraste, onde os objetos são muito escuros e o fundo é homogêneo e muito iluminado, então um limiar constante de 128 numa escala de 0 a 255 talvez seja suficientemente exato. Por exatidão entende-se que o número de pixels falsamente classificados é mantido ao mínimo (YOUNG, 2004).

2.3.3.1.2 Limiares Derivados de Histogramas

Na maioria de casos o limiar é escolhido a partir do histograma de brilho da região ou imagem que se deseja dividir. Uma imagem e seu histograma de brilho associado são mostrados na Figura 2.12. Os pixels sob o limiar ($a[m,n] < \theta$) serão marcados como pixels de objeto e os acima do limiar estarão marcados como pixels de fundo (YOUNG, 2004).



(a) Imagem a ser processada (b) Histograma de brilho da imagem

Figura 2.12 - Imagem de entrada e histograma de brilho

2.3.3.1.3 Algoritmo Isodata

O histograma é inicialmente dividido em duas partes, usando um valor de limiar inicial correspondente a metade do alcance dinâmico máximo. A amostra central dos valores de cinza associados aos pixels de primeiro plano e a amostra central dos valores de cinza associados aos pixels de fundo são calculados. Um novo valor de limiar é calculado como sendo a média destas duas amostras. Este processo é repetido recursivamente, baseado no novo limiar, até que o valor de limiar não mude mais (RIDLER[1], 1978 apud YOUNG 2004).

2.3.3.1.4 Algoritmo de Simetria de Fundo

Esta técnica supõe um pico dominante distinto para o fundo, que é simétrico ao seu máximo. O pico máximo ($p_{máx}$) corresponde ao valor máximo do histograma. O algoritmo então procura no lado dos pixels não pertencentes a objetos do máximo para determinar um ponto p% (YOUNG, 2004).

[1] RIDLER, T.W.; Calvard, S. Picture thresholding using an iterative selection method. IEEE Trans. on Systems, Man, and Cybernetics, 1978. SMC-8(8): p. 630-632.

Utilizando o exemplo da Figura 2.12b, onde os pixels de objeto são localizados à esquerda do pico de fundo (brilho = 183), significaria localizar a direita desse pico, por exemplo, o valor de 95%. Neste valor de brilho, 5% dos pixels estão à direita (acima) desse valor. Isto ocorre para o valor de brilho 216 na Figura 2.12b. Em virtude da suposta simetria, é utilizado como limiar um deslocamento à esquerda do máximo, que é igual ao deslocamento à direita onde o p% foi encontrado. Para o exemplo em questão, este valor de limiar seria dado por 183 - (216 - 183) = 150. Tal valor é obtido de acordo com a Equação 2.18 (YOUNG, 2004).

$$\theta = P_{máx} - (P\% - P_{máx}) \qquad (2.18)$$

2.3.3.1.5 Algoritmo do Triângulo

Uma linha é traçada entre o máximo do histograma de brilho B_{max} e o menor valor B_{min}= *(P=0)%* na imagem. A distância *d* entre a linha e o histograma *h[B]* é calculada para todos os valores de *B*, onde $B = B_{min}$ até $B = B_{max}$. O valor de brilho B_0, onde a distância entre $h[B_0]$ e a linha é máxima, corresponde ao valor de limiar, ou seja, $\theta=B_0$. Esta técnica é particularmente eficiente quando os pixels de objeto produzem um pico fraco no histograma. A Figura 2.13 ilustra a obtenção da distância *d* a partir do histograma de brilho da imagem, utilizada no algoritmo do triângulo (ZACK[2], 1977 apud YOUNG, 2004).

[2] ZACK, G.W.; Rogers, W.E.; Latt, S.A. Automatic Measurement of Sister Chromatid Exchange Frequency. 1977. 25(7): p. 741-753.

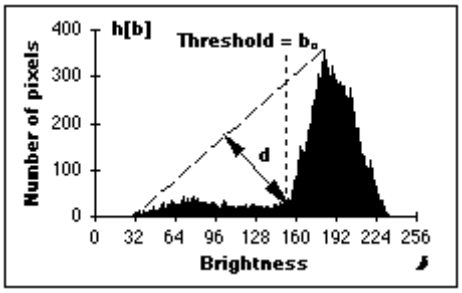


Figura 2.13 - Ilustração do algoritmo do triângulo

Os três procedimentos descritos acima resultam nos valores θ=139 para o algoritmo de Isodata, e θ=150 para o algoritmo de simetria de fundo com 5% de nível, e θ=152 para o algoritmo de triângulo, considerando a imagem da Figura 2.12a (YOUNG, 2004).

O filtro de limiar (*thresholding*) não precisa necessariamente ser aplicado a imagens inteiras, podendo ser utilizado em regiões específicas da imagem. Chow e Kaneko[3] (1972 apud YOUNG, 2004) desenvolveram uma algoritmo em que a imagem de *NxM* é dividida em regiões não sobrepostas. Para cada região um limiar é calculado e os valores resultantes de limiar são montados (interpolados) formando uma superfície de limiarização para a imagem inteira. As regiões devem ser de tamanho razoável, de modo que haja um número suficiente de pixels em cada região para se fazer uma estimativa do histograma e, conseqüentemente, do valor de limiar. A utilidade deste procedimento - como tantos outros - depende da aplicação em questão (YOUNG, 2004).

2.3.3.2 Bordas (*Edges*)

O *Thresholding* produz uma segmentação que resulta em todos os pixels que, em princípio, pertencem ao objeto (ou objetos) de interesse numa imagem. Uma alternativa a está técnica é encontrar os pixels que pertencem às fronteiras dos

[3] CHOW, C.K.; Kaneko, T. Automatic boundary detection of the left ventricle from cineangiograms. Computers and Biomedical Research, 1972. 5: p. 388-410.

objetos. As técnicas que têm esse objetivo são denominadas técnicas de “descoberta de borda”, ou simplesmente “bordas” (*edges*) (YOUNG, 2004).

2.3.3.2.1 Procedimento baseado em Gradiente

O desafio central das técnicas de bordas é definir procedimentos que produzam contornos fechados ao redor dos objetos de interesse. Para objetos de SNR[4] particularmente alto, pode-se calcular o gradiente e então utilizar um valor de limiar conveniente. A Figura 2.14 ilustra a aplicação do algoritmo de bordas baseadas em gradiente para duas imagens com SNR distintos (YOUNG, 2004).

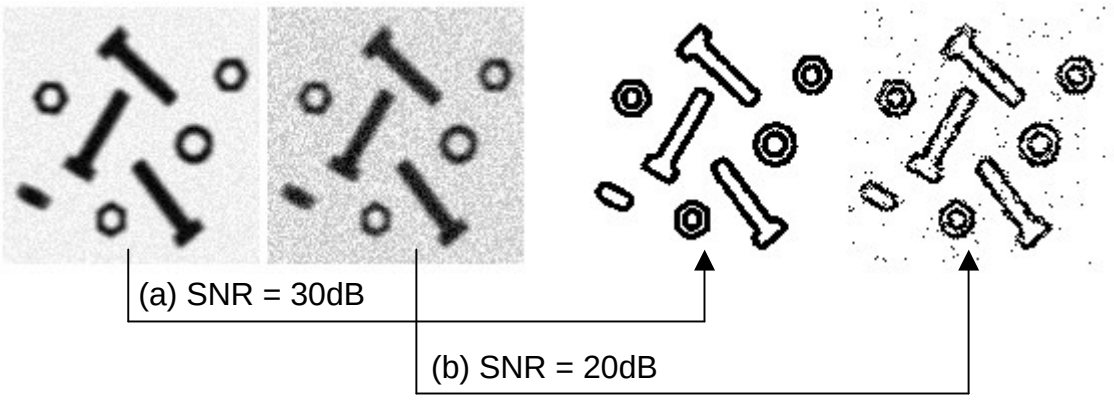


Figura 2.14 - Bordas baseadas em gradiente, combinado com limiar/isodata

Observa-se que a técnica funciona bem para a imagem com SNR igual a 30dB (Figura 2.14a), mas não consegue fornecer uma determinação exata dos pixels associados às bordas de objeto para a imagem com SNR igual a 20dB (Figura 2.14b) (YOUNG, 2004).

[4] *Signal-to-Noise Ratio* ou Taxa de Sinal-Ruído

Há uma variedade de técnicas de amaciamento para se reduzir os efeitos do ruído antes do operador de gradiente ser aplicado, conforme seção 2.3.2.2 deste capítulo.

2.3.3.2.2 Procedimento baseado em Cruzamento em Zero

Uma abordagem mais moderna da manipulação do problema de bordas em imagens ruidosas é usar os cruzamentos em zero (*zero-crossing*) obtidos do Laplaciano de uma imagem. O raciocínio parte do modelo de borda ideal, ou seja, uma função de passo, turvada por uma transformada de Fourier. Este procedimento deve produzir resultado como o mostrado na Figura 2.15 (YOUNG, 2004).

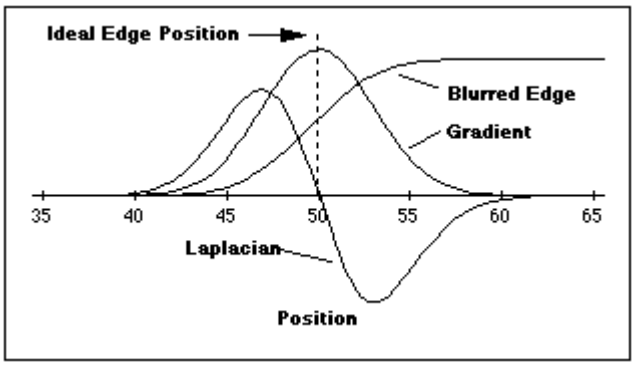


Figura 2.15 - Bordas baseado no cruzamento de zeros obtido do laplaciano da imagem.

A localização das bordas é dada, de acordo com o modelo, no lugar na imagem onde o Laplaciano muda de sinal (cruzamento do zero). Como o Laplaciano envolve uma derivada segunda, isto implica em um aumento potencial do ruído para freqüências espaciais altas. Para evitar que este aumento de ruído influencie na busca dos zeros é preciso aplicar um algoritmo de amaciamento (YOUNG, 2004).

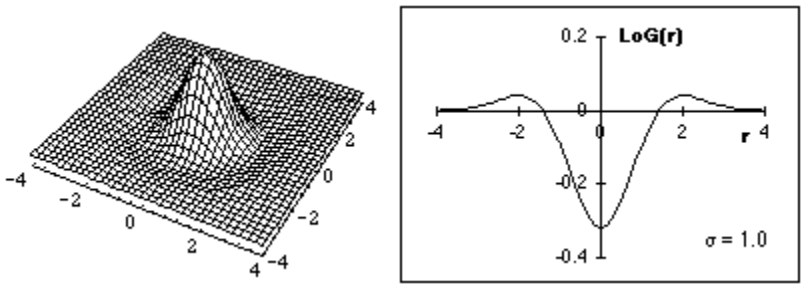
O filtro de amaciamento apropriado, dentre os descritos no item 2.3.2.2, tem as seguintes propriedades (CANNY[5], 1986 apud YOUNG, 2004):

a) No domínio da freqüência, (u,v) ou (Ω, ψ), o filtro deve ser suficientemente estreito para suprimir ruídos de alta freqüência;
b) No domínio do espaço, (x,y) ou (*m,n*), o filtro deve ser suficientemente estreito para prover a localização das bordas. Um filtro muito largo produzirá imprecisão na localização das bordas.

O filtro de amaciamento que satisfaz ambas as propriedades – largura de banda mínima e largura espacial mínima – é o filtro Gaussiano, descrito no item 2.3.2.2.1.3. Isto implica que a imagem deve ser amaciada através de um filtro Gaussiano com um θ apropriado, seguido da aplicação do Laplaciano, conforme Equação 2.19 (YOUNG, 2004).

$$ZeroCross\{a(x,y)\} = \left\{(x,y) \mid \Delta^2\{g_{2D}(x,y) \otimes a(x,y)\} = 0\right\} \quad (2.19)$$

A Figura 2.16 mostra um exemplo de filtro de detecção de bordas utilizando cruzamentos em zero, também conhecido como “Filtro do Chapéu Mexicano” (YOUNG, 2004).



(a) LoG(x,y) (b) LoG(r)

Figura 2.16 - Filtro do “Chapéu Mexicano”

[5] CANNY, J. A Computational Approach to Edge Detection. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1986. PAMI-8(6): p. 679-698.

O termo LoG mostrado na Figura 2.16a e 2.16b refere-se a abreviatura de Laplaciano do Gaussiano (*Laplacian of Gaussian*), conforme procedimento matemático descrito na Equação 2.19.

2.3.3.2.3 Procedimento baseado em Soma

Uma alternativa mais precisa ao procedimento de Cruzamento de Zeros para detecção de bordas é o filtro de soma (VERBEEK-VILET[6], 1994 apud YOUNG, 2004).

Todos os métodos baseados em cruzamentos de zero no Laplaciano devem ser capazes de distinguir entre cruzamentos no zero e valores zero. Enquanto o primeiro representa posições de borda, o último pode ser gerado por regiões que não são mais complexas que superfícies bi-lineares. Para distinguir entre estas duas situações, primeiramente são encontradas as posições de cruzamentos de zero que são marcadas com "1" e todos os outros pixels como "0". Então multiplica-se a imagem resultante por uma medida da força de borda em cada pixel. Há várias medidas para a força de borda e são todas baseadas no gradiente da imagem. O uso de um gradiente morfológico como uma medida de força de borda é particularmente eficiente (LEE[7], 1986 apud YOUNG, 2004).

Depois da multiplicação a imagem é então amaciada para produzir o resultado final. O procedimento está ilustrado na Figura 2.17 (YOUNG, 2004).

[6] VERBEEK-VILET; Verbeek, P.W., L.J. Van Vilet. On the Location Error of Curved Edges in Low-Pass Filtered 2D and 3D Images. IEEE Transactions on Pattern Analysis and Machine Intelligence, 1994. 16(7): p. 726-733.

[7] LEE, J.S.L., R.M. Haralick, and L.S. Shapiro. Morphologic Edge Detection. in 8th International Conference on Pattern Recognition. 1986. Paris: IEEE Computer Society.

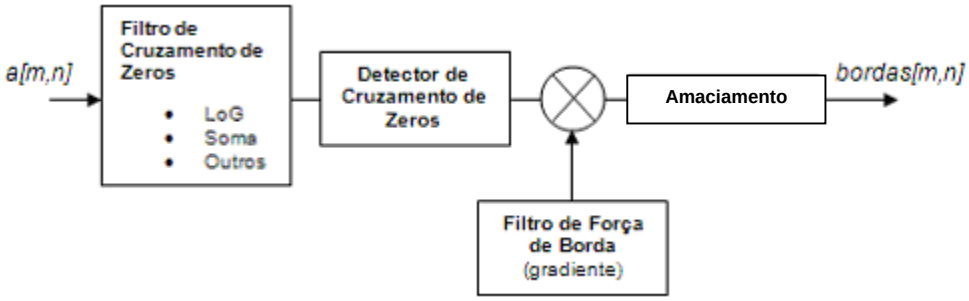


Figura 2.17 - Estratégia geral para "Bordas" baseado em cruzamentos de zeros

Os resultados destas duas técnicas de determinação de bordas baseadas em cruzamentos de zero, filtro LoG e filtro de Soma, são mostrados na Figura 2.18, com SNR de 20dB (YOUNG, 2004).

a) Imagem c/ SNR = 20dB b) Filtro LoG c) Filtro Soma

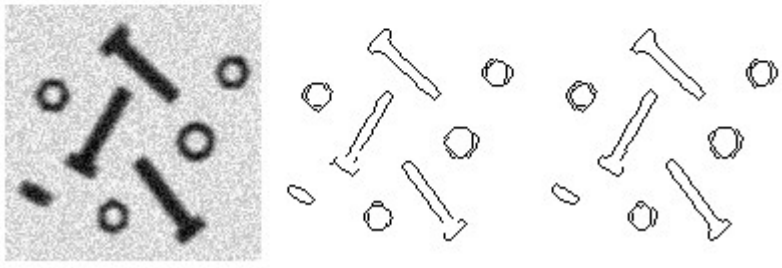


Figura 2.18 - Detecção de bordas usando algoritmos LoG e Soma

3 DETECÇÃO DE MOVIMENTO POR ANÁLISE DE IMAGENS

Este capítulo apresenta dois métodos de detecção de movimento através de análise de imagens. O primeiro método apresentado, de *diferença entre dois quadros sucessivos* não foi utilizado no projeto, porém sua comparação com o segundo método, *diferença entre o quadro atual e uma imagem com o cenário de fundo armazenado*, que foi a abordagem adotada, colabora para o entendimento dos motivos de sua utilização.

3.1 Preâmbulo

Em ambientes onde se queira detectar pessoas ou objetos, geralmente não há controle sobre o que haverá no plano de fundo da imagem capturada (e como conseqüência, não se tem muito controle sobre sua complexidade), bem como na intensidade de iluminação e outros objetos em movimento. As pessoas também têm sua aparência sujeita a uma enorme variação, desde vestimentas, cor da pele, ausência de algum dos membros, altura e diferentes possibilidades de movimentos (SOARES, 2004).

Todos estes fatores dificultam a identificação de formas na imagem, embora algumas tentativas já tenham sido realizadas com taxas de sucesso razoáveis (WREN[8], 1997 apud SOARES, 2004) (BAUMBERG[9], 1994 apud SOARES, 2004).

[8] C. WREN, A. Azarbayejani, T. Darrell, A. Pentland. Pfinder: Real-Time tracking of the human body, IEEE. Trans. Pattern Anal. Mach. Intell. Vol 19 no. 7, 1997: p. 780-785

[9] A. BAUMBERG, D. Hogg. An efficient method for contour tracking using active shape models, Proc. of IEEE Workshop on Motion of Non-Rigid and Articulated Objects. Austin, 1994: p. 194–199.

3.2 Método: Diferença entre dois quadros sucessivos

O movimento é um poderoso artifício empregado por seres humanos e animais para extrair objetos de interesse de um cenário de fundo (GONZALEZ[10], 1992). Neste método, calcula-se para cada quadro a diferença entre este e o anterior, o qual é tomado como referência. Assim, se uma imagem for expressa como $I=f(x,y,t)$, onde I é um valor representando a intensidade luminosa no ponto de coordenadas (x,y) no instante t, tem-se que (SOARES, 2004):

$$\Delta I=|f(x,y,t_2)- f(x,y,t1)| \qquad (3.1)$$

onde ΔI é o módulo da diferença entre a imagem no instante t_2 (imagem atual) e a imagem no instante t_1 (imagem anterior).

Em um ambiente controlado, com pouca variação, a diferença entre os dois quadros sucessivos terá valores iguais ou muito próximos a zero nas regiões onde existe pouca ou nenhuma variação na intensidade luminosa (ou seja, no cenário de fundo que permaneceu estático), e valores diferentes de zero nas regiões onde objetos estão em movimento. Pode-se considerar que variações, dentro de certa faixa de tamanho (extensão no plano) e proporções, correspondam a uma pessoa em movimento e, fora desta faixa, simples objetos (SOARES, 2004).

Este método está sujeito a falhas, como por exemplo, quando uma pessoa pára no centro da imagem, entra outra pessoa na cena, e as duas saem juntas; ou quando duas pessoas passam muito próximas na frente da câmera. Outro problema é quando uma pessoa em movimento pára, conseqüentemente desaparece (objetos estáticos não produzem diferenças entre dois quadros). Este problema pode ser resolvido com um controle da posição de regiões e assumindo-se que as pessoas entram e saem do local monitorado passando pelas regiões correspondentes as laterais da imagem (SOARES, 2004).

[10] GONZALEZ R.C., Woods R. E. Digital Image Processing - Addison-Wesley Publishing Company, 1992, ISBN 0-201-50803-6, p. 632

Um parâmetro a ser ajustado no sistema é o valor de limiarização da diferença entre as duas imagens. Este limiar indicará o que é realmente uma diferença entre as duas imagens (considerado como uma variação grande) e o que pode ser encarado como ruído (variação pequena). Valores muito baixos de limiar produzirão diferenças por toda imagem, uma vez que duas imagens sucessivas não são perfeitamente iguais devido a erros introduzidos no processo de aquisição de imagens e variações na iluminação. Valores de pixel inferiores ao limiar são substituídos por zero e valores acima por 255. Os valores iguais a 255 são identificados como presença de movimento (SOARES, 2004).

Valores elevados de limiar não permitirão a detecção de diferenças entre as duas imagens. Valores de limiar entre os extremos produzem nas regiões onde objetos ou pessoas encontram-se em movimento, um contorno da silhueta da pessoa, correspondente ao movimento que ocorreu durante o tempo de transição de um quadro (SOARES, 2004).

O resultado da limiarização neste método produz regiões normalmente muito parecidas com o contorno da pessoa em movimento (SOARES, 2004).

3.3 Método: Diferença entre o quadro atual e uma imagem com o cenário de fundo armazenado

O método apresentado consiste basicamente em detectar variações na imagem, em relação a um cenário de fundo armazenado previamente. Assim, as restrições no tipo de imagens utilizadas são variações lentas ou em pequenas partes (menores que o tamanho do menor objeto que será detectado) do cenário de fundo e, como conseqüência, a câmera deve ficar fixa (SOARES, 2004).

Neste método, o cenário de fundo (ambiente) é armazenado inicialmente na forma de uma imagem de referência, a qual é subtraída de cada quadro da seqüência. Assumindo a mesma representação para a imagem adotada no método da diferença entre dois quadros sucessivos, temos a diferença entre as imagens representada como (SOARES, 2004):

$$\Delta I = |f(x,y,t) - f_r(x,y)| \quad (3.2)$$

onde ΔI é o módulo da diferença entre a imagem no instante t (imagem atual) e $f_r(x,y)$ é imagem de referência do cenário de fundo. A Figura 3.1 ilustra o método de diferença entre o quadro atual de uma imagem e o cenário de fundo armazenado.

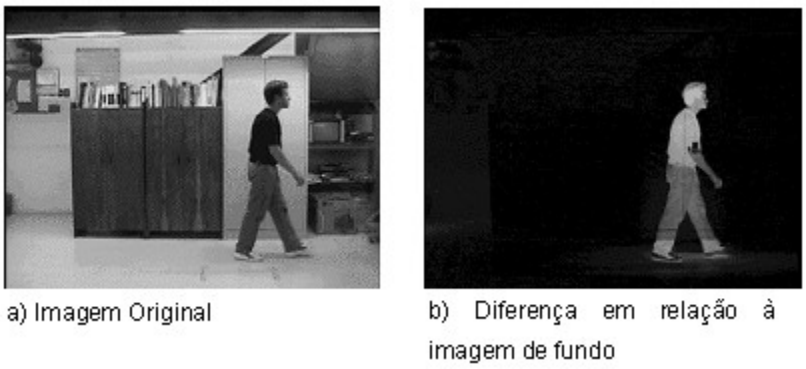


Figura 3.1 - Ilustração do método de diferença entre o quadro atual e uma imagem com o cenário de fundo armazenado

Esta diferença está relacionada com a variação na intensidade luminosa entre o instante atual e o inicial em cada pixel da imagem. Assim, objetos presentes na imagem e que não foram registrados na imagem de referência produzem normalmente valores de diferença significativos, enquanto que em outros pontos da imagem esta diferença será igual ou muito próxima de zero. No caso de uma pessoa entrar na imagem, será produzida uma diferença entre as duas imagens na posição onde a pessoa se encontra. Esta variação deve ser limiarizada para poder-se distinguir o que é uma variação significativa das pequenas variações já citadas no método de diferença entre quadros consecutivos (subseção 2.3.1). Valores de pixel inferiores ao limiar são substituídos por zero e valores superiores por 255. Valores iguais a 255 são identificados como presença de um objeto (SOARES, 2004).

A limiarização neste método retorna a silhueta da pessoa ou objeto presente na imagem. Contudo, como as diferenças de intensidade luminosa podem ser pequenas dependendo de como a luz é refletida por cada pessoa, esta silhueta

poderá não ser perfeita. Contudo, em casos com bastante contraste entre a imagem da pessoa e do fundo, os poucos defeitos presentes na imagem poderão ser corrigidos através de filtragem morfológica. Sucessivas operações de dilatação seguidas por operações de erosão podem preencher eventuais buracos na silhueta (SOARES, 2004).

Este método tem a vantagem de não provocar o desaparecimento quando a pessoa fica parada, em relação ao método *Diferença entre dois quadros Sucessivos*, por exemplo. Para sistemas com um funcionamento contínuo durante um longo período de tempo, normalmente a imagem de referência do cenário de fundo deve sofrer uma atualização lenta, de forma a compensar variações no ambiente que possam ocorrer (SOARES, 2004).

4 DESCRIÇÃO DO *HARDWARE*

Este capítulo trata da descrição do protótipo construído, dos componentes eletrônicos e de *hardware* utilizados.

4.1 Diagrama de Blocos

O protótipo construído está ilustrado através do diagrama de blocos representado na Figura 4.1, mostrando os principais componentes e o fluxo de operação.

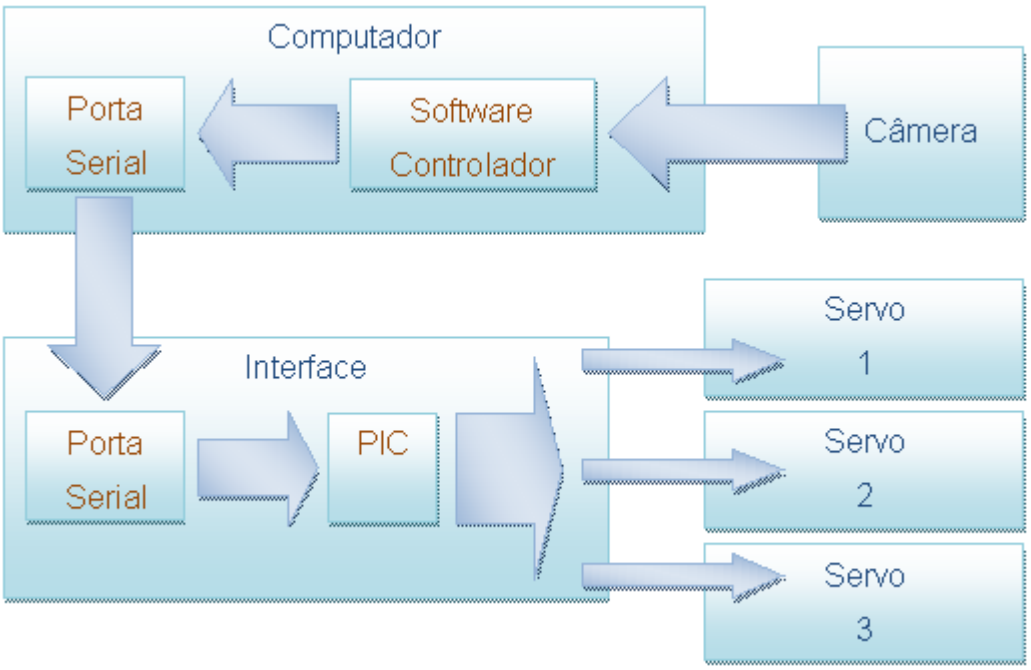


Figura 4.1 - Diagrama de blocos do protótipo

4.2 A Câmera

A câmera utilizada é uma câmera web comum, marca A4Tech, modelo PK-635M, colorida, com definição máxima de 640 X 480 e 350K pixels. Para o projeto está sendo utilizada com a resolução de 320 X 240 com o intuito de melhorar a performance do processamento das imagens, uma vez que para os testes realizados esta resolução mostrou-se suficientemente precisa. A câmera possui recurso de balanço dos brancos automático, o que reduz o nível de ruído nas imagens por variação de intensidade luminosa do ambiente.

A conexão com o computador é feita através de uma porta USB, sendo ativada manualmente através do software controlador, o qual captura os quadros (*frames*) para processamento.

Inicialmente a câmera havia sido fixada na mesma estrutura que sustenta os motores e a arma, porém a eventual vibração do conjunto em função do movimento dos motores causou distorções nas imagens coletadas. O problema foi solucionado utilizando-se uma estrutura independente para sustentar a câmera.

4.3 Porta Serial

A comunicação entre o computador e a interface (placa controladora) é realizada através de uma porta serial. O software controlador é responsável por enviar os comandos de movimentação dos servomotores para a interface de controle por esta porta.

4.3.1 Comunicação com RS-232

A interface serial mais comumente utilizada nos microcomputadores é a RS-232. Originalmente foi criada para facilitar a interconexão dos terminais e dos equipamentos de comunicação de dados (FERNANDES, 2005).

Na interface RS-232 os pinos mais comumente utilizados são três, sendo um com a função de enviar e outro com a função de receber os dados. Uns poucos pinos no conector são absolutamente previsíveis conforme mostrado na Tabela 4.1 (FERNANDES, 2005).

Tabela 4.1 - Pinos para comunicação serial

Pino	Função
Pino 2	Pino para transmissão
Pino 3	Pino para recepção
Pino 5	Circuito comum

No que diz respeito às características elétricas, o padrão RS-232 define atualmente 4 níveis lógicos. As entradas têm definições diferentes dos dados. Para as saídas, o sinal é considerado na condição de estado “1”, quando a tensão no circuito de transferência, medida no ponto de interface é menor que –5V e maior que –15V, com relação ao circuito de referência (terra). O sinal é considerado na condição de estado “0”, quando a tensão for maior que +5V e menor que +15V, também com relação ao circuito de referência (terra) (FERNANDES, 2005)..

Para as entradas, o sinal é considerado em condição de marca, ou estado “1”, quando a tensão no circuito de transferência, medida no ponto de interface, é menor que –3V e maior que –15V, com relação ao circuito terra. O sinal é considerado na condição de espaço ou estado “0”, quando a tensão for maior que +3V e menor que +15V, também com relação ao circuito terra. A região compreendida entre –3V e +3v é definida como região de transição (FERNANDES, 2005)..

Durante a transmissão dos dados, a condição de marca é usada para discriminar o estado binário “1”, e a condição de espaço é usada para discriminar o estado binário “0” (FERNANDES, 2005).

Muitas aplicações utilizam a conexão direta via cabo para trocar informações entre dois computadores. As utilidades vão desde o simples compartilhamento de arquivos sem a utilização de placas de rede até o jogo entre dois adversários em

computadores diferentes. Cada computador dispõe de pelo menos uma porta serial, o conector pode ser um DB9 ou um DB25 (FERNANDES, 2005).

4.4 A Interface (Placa Controladora)

A placa controladora de servos utilizada é uma placa típica para controle através do computador ou ainda através de um microcontrolador. A interface suporta até oito servomotores simultaneamente, embora neste projeto sejam utilizados apenas três.

A interface com o computador é feita através de uma porta serial (padrão RS-232), como dito anteriormente. Para alimentação do conjunto placa / servomotores é utilizada uma fonte comum de 5V e 1,5A. A placa interpreta através de um software embarcado, um conjunto de comandos pré-definidos, conforme exemplos:

- Exemplo de comando de inicialização da placa:

```
SCSTART100700
```

Legenda:

SCSTART	: Comando de inicialização
10	: Valor da velocidade de comunicação (0 a 99)
0700	: Valor da posição inicial dos motores (0050 a 1250)

- Exemplo de comando de posicionamento:

```
SP107501200 0
```

Legenda:

SP	: Comando de posicionamento de servo
1	: Servo a ser movimentado (de 1 a 8)
0750	: Posição inicial do servo (0050 a 1250)
1200	: Posição final do servo (0050 a 1250)
0	: Sentido invertido (1 = sim, 2 = não)

Figura 4.2 – Foto da placa controladora / Conector para porta serial

4.5 Os Servomotores

Os servomotores utilizados são de uso típico em aeromodelismo, eficientes por possuírem bom torque (3,06kg/cm) em relação às dimensões que são reduzidas (41mm x 20mm x 36mm). O ângulo máximo de giro é 180º, o que limita o raio de ação do dispositivo. Porém, considerando-se a resolução da câmera utilizada, isso não mostrou-se problemático, uma vez que as cenas percebidas são sempre cobertas pelo raio de ação dos motores. A figura 4.3 mostra fotos do tipo de servomotor utilizado.

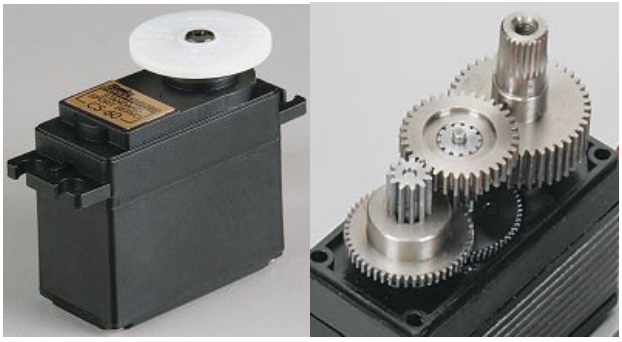


Figura 4.3 - Servomotores utilizados no projeto

Foram utilizados três servomotores, distribuídos da seguinte forma:

a) 1 para controle de movimento horizontal (eixo X);
b) 1 para controle de movimento vertical (eixo Y) e;
c) 1 para pressão do gatilho da arma.

Os servomotores possuem um cabo de conexão com terminação em 3 pinos (neutro central) através dos quais são conectados à placa controladora, conforme ilustrado na Figura 4.4.

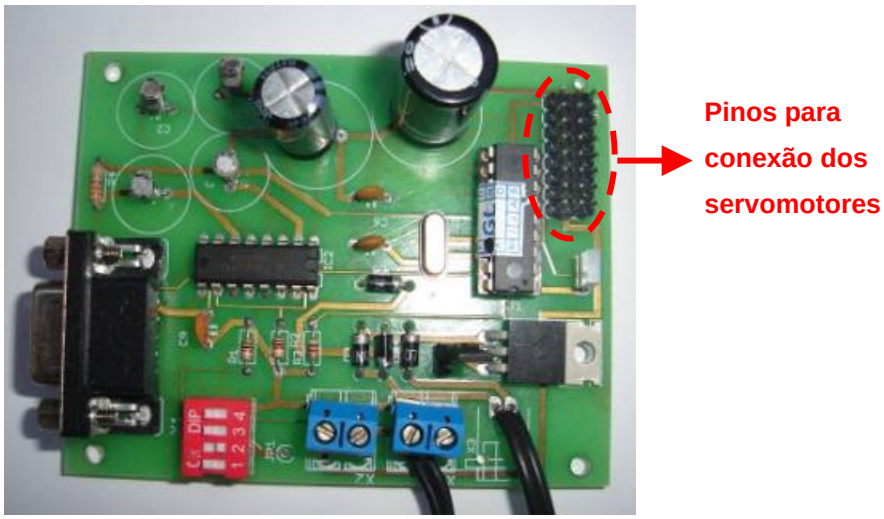


Figura 4.4 – Pinos de conexão dos servomotores com a placa controladora

5 DESCRIÇÃO DO *SOFTWARE*

Este capítulo detalha a construção do *software* controlador, que está dividido em três módulos principais: Detecção de Movimento, Controle da Arma e Camada de Apresentação*.* Há ainda um tópico específico para apresentação do framework de código aberto Aforge.Net, utilizado no desenvolvimento e que agrega uma série de funcionalidades genéricas para processamento de imagens, tais como os filtros e algoritmos descritos no capítulo 2. O código-fonte completo do software está no Apêndice 1.

5.1 Tecnologia Utilizada

Para a construção do software controlador foi utilizado o Microsoft Framework 2.0 [11], linguagem C#. A escolha deu-se pela familiaridade com o ambiente e pela facilidade que a linguagem traz para implementação de orientação a objeto, além de sua utilização e distribuição serem gratuitas.

5.2 AForge.Net *Framework*

O Aforge.Net é um *framework* de código aberto (*open source*), desenvolvido por Andrew Kirillov[12], licenciado sob os termos da *GNU General Public License 2.0*, escrito em linguagem C#, com o objetivo de auxiliar desenvolvedores e pesquisadores nos campos da “Visão por Computador” e “Inteligência Artificial”, como processamento de imagens, redes neurais, algoritmos de genética e aprendizagem de máquinas.

[11] Ver http://msdn2.microsoft.com/pt-br/netframework/default.aspx
[12] O código fonte, bem como detalhes do projeto, podem ser encontrados em http://code.google.com/p/aforge/

Atualmente o projeto é composto de 5 bibliotecas principais, além de algumas de uso genérico (apoio). As principais são:

a) AForge.Imaging – Biblioteca para rotinas de processamento de imagens e filtros;
b) AForge.Neuro – Biblioteca para computação em redes neurais;
c) AForge.Genetic – Biblioteca para algoritmos na área de genética;
d) AForge.Vision – Biblioteca para Visão por Computador;
e) AForge.Machine Learning – Biblioteca para aprendizagem de máquinas.

Neste projeto apenas a biblioteca AForge.Imaging foi utilizada como parte integrante do software desenvolvido. Esta biblioteca contém as implementações de filtros e algoritmos de processamento de imagem que são consumidos pelo módulo de Detecção de Movimento.

5.3 Módulo de Detecção de Movimento

O módulo de detecção de movimento foi o primeiro a ser desenvolvido e implementa as seguintes funcionalidades principais, que serão descritas em detalhes:

a) Aquisição de imagens;
b) Pré-processamento das imagens;
c) Separação do cenário de fundo da imagem;
d) Aplicação dos filtros e algoritmos para detecção de movimento, identificação e posicionamento dos objetos em cena e;
e) Pós-processamento das imagens.

A Figura 5.1 ilustra a seqüência de passos executados pelo módulo de detecção de movimento, de forma resumida.

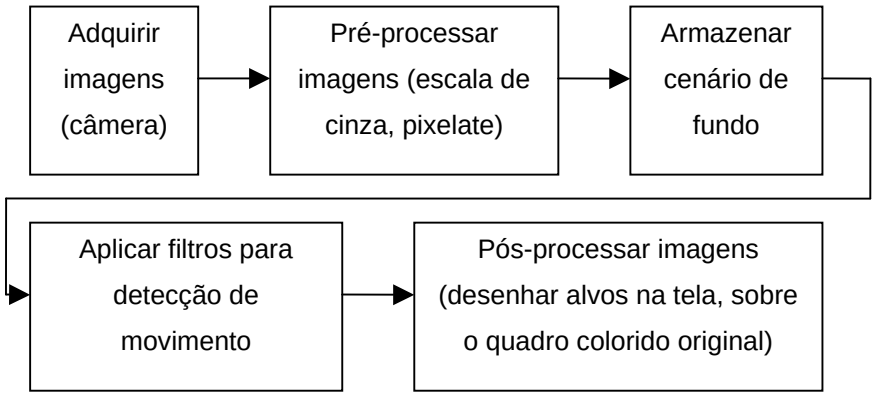


Figura 5.1 – Seqüência de passos executados pelo módulo de detecção de movimento

A Figura 5.2 mostra o diagrama com as classes mais relevantes do módulo de detecção de movimento

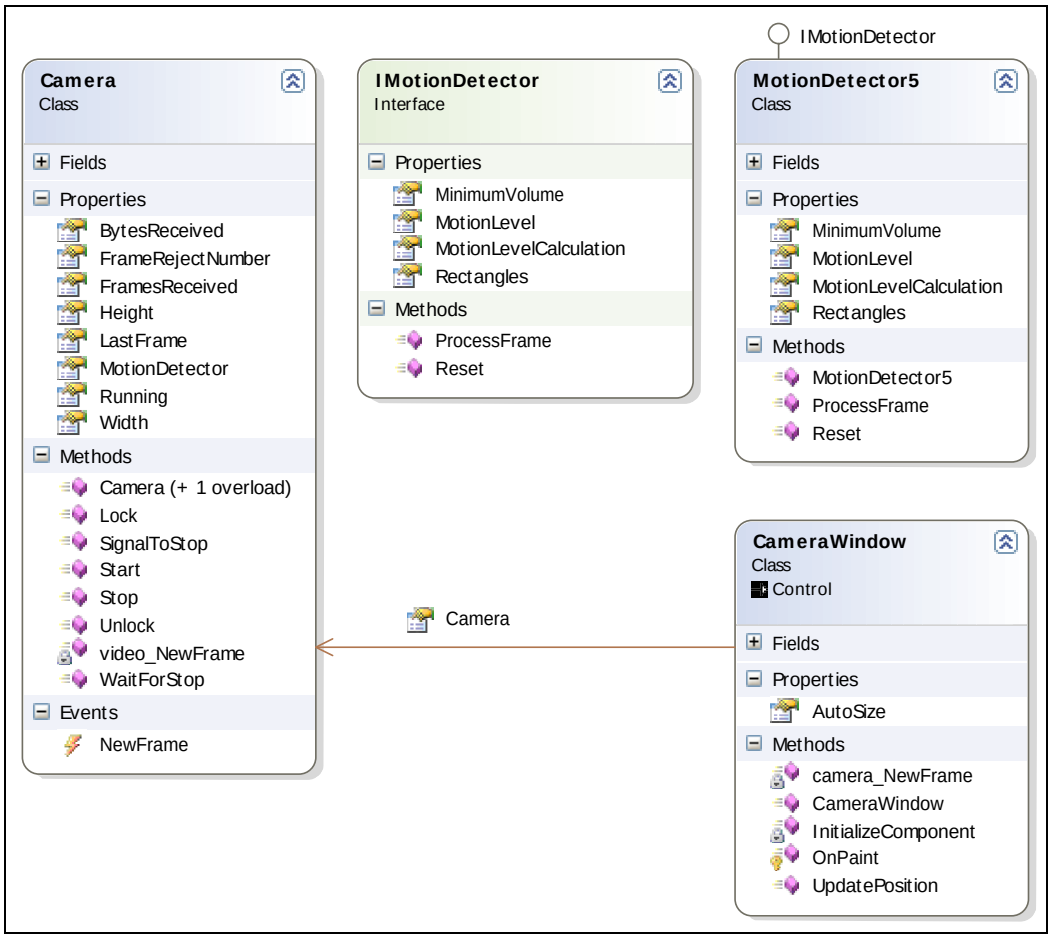


Figura 5.2 - Classes principais do módulo de detecção de movimento

5.3.1 Aquisição de Imagens

Foram desenvolvidas classes que implementam as funções para conexão com a câmera, captura do *streaming* de vídeo e sua conversão para o formato “mapa de bits”. As subseções 5.3.1.1, 5.3.1.2, 5.3.1.3, 5.3.1.4 e 5.3.1.5 as descrevem e seus códigos-fonte estão no Apêndice A.

5.3.1.1 Interface *IVideoSource*

Esta classe é uma interface para qualquer objeto que forneça um *streaming* de vídeo, provendo compatibilidade entre diversas fontes de vídeo como, por exemplo, arquivos em formato AVI, JPEG streamings e MJPEG streamings. Além de declarar propriedades comuns, ela possui um *Event Handler* para tratar os frames fornecidos pela fonte produtora de vídeo.

5.3.1.2 Classe *CaptureDevice*

Esta classe implementa a interface abstrata *IVideoSource* para o caso de um dispositivo local (conectado diretamente ao computador) ser utilizado como fonte de streaming de vídeo.

5.3.1.3 Classe *Camera*

Esta classe implementa um objeto como função principal publicar os quadros (*frames*) obtidos, encapsulando o objeto *videosource* e provendo serviços de alto

nível para o controle da câmera. É também responsável por manter a instância do tipo de detector de movimento utilizado.

5.3.2 Pré-processamento de Imagens

Esta funcionalidade foi embutida diretamente nas classes que implementam a interface *IMotionDetector*. São basicamente conversão para escala de cinza e simplificação dos valores de brilho dos pixels da imagem para números inteiros (média dos canais).

5.3.3 Separação do cenário de fundo da imagem

Esta funcionalidade é parte do método de detecção utilizado e também é executada diretamente pelas classes que implementam a interface *IMotionDetector*, no método *ProcessFrame.*

5.3.4 Aplicação dos filtros e algoritmos para detecção de movimento, identificação e posicionamento dos objetos em cena

Neste ponto ocorre a utilização dos filtros providos pelo *framework* AForge.Net, cuja teoria foi introduzida no capítulo 2. O primeiro passo consiste em aplicar o filtro de diferença entre as imagens de fundo (*overlay image*) e a imagem atual, conforme descrito na subseção 2.3.2.1.1.

A imagem resultante é então limiarizada através de um filtro baseado no histograma de brilho da imagem, conforme descrito na subseção 2.3.3.1.2. O passo

seguinte consiste em processar a imagem resultante pela classe *BlobCounter* que executa filtro de Bordas baseado em soma (vide subseção 2.3.3.2.3) e Dilatação (vide subseção 2.3.2.3.1) para extrair os retângulos que correspondem aos alvos identificados.

Na seqüência, o volume do retângulo é avaliado para verificar a necessidade de desprezo do alvo, de acordo com parâmetro definido no início da aplicação pelo usuário, a ser descrito na subseção.

A seqüência desses passos está ilustrada na Figura 5.3

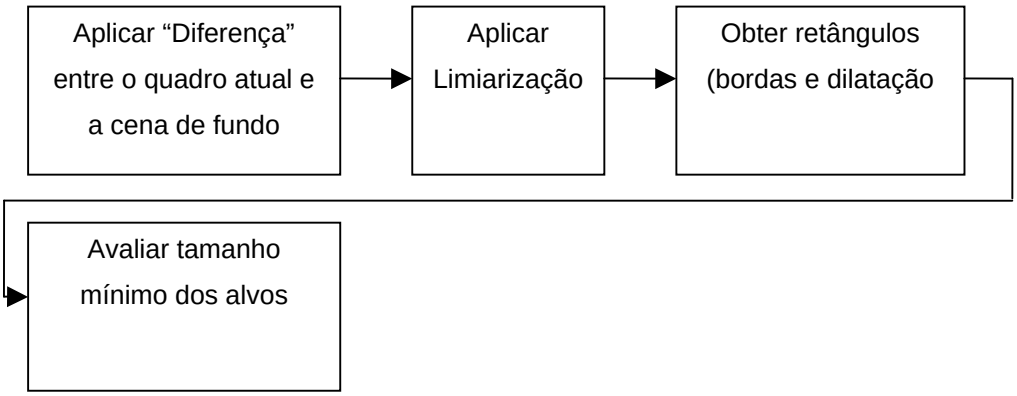


Figura 5.3 – Seqüência de passos executados pelo módulo de detecção de movimento para detecção de movimento, identificação e posicionamento dos objetos em cena

5.3.4.1 Interface *IMotionDetector*

Esta interface abstrai objetos genéricos responsáveis por aplicar algoritmos e filtros de imagens, com o intuito de detectar movimento e identificar os objetos que não pertencem à cena de fundo e que serão tratados como alvos. As classes que implementam essa interface se relacionam diretamente com as classes do processo de aquisição de imagens.

5.3.4.2 Classe *MotionDetector5*

Esta classe implementa um objeto responsável por tratar as imagens adquiridas, detectando movimento e identificando os objetos que não pertencem à cena de fundo e que serão tratados como alvos. Existem outras implementações de *IMotionDetector* que foram realizadas na fase de testes e que foram mantidas no projeto para fins de histórico e entendimento da seqüência que levou a obtenção deste algoritmo final.

5.3.5 Pós-processamento de imagens

O pós-processamento realizado das imagens consiste em prepará-las para exibição na tela, através da camada de apresentação. As imagens originais são mantidas inalteradas e sob elas são aplicados os desenhos dos retângulos numerados que representam os alvos presentes na cena em questão.

5.4 Módulo de Controle da Arma

O módulo de controle de arma implementa as seguintes funcionalidades principais, que serão descritas em detalhes:

a) Análise e monitoração de alvos;
b) Comunicação com a placa controladora;
c) Movimentação dos servos;

A Figura 5.4 ilustra a seqüência de passos executados pelo módulo de controle da arma.

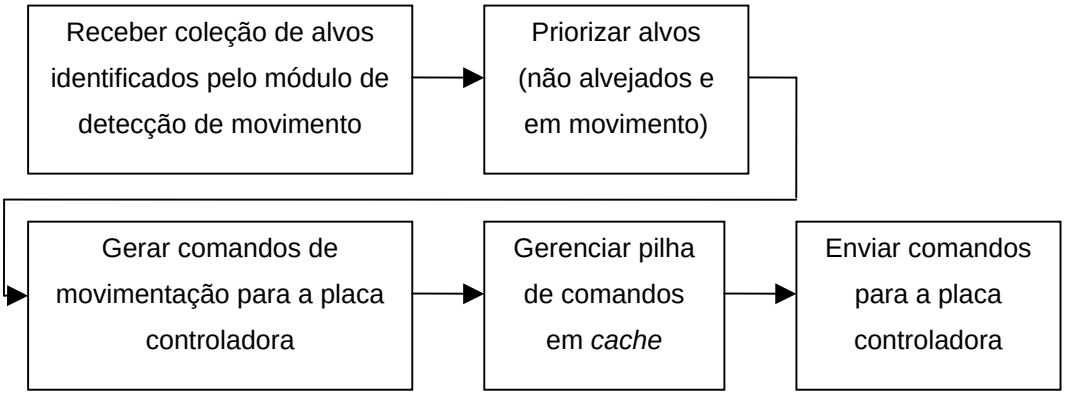


Figura 5.4 – Seqüência de passos executados pelo módulo de controle da arma

A Figura 5.5 mostra o diagrama com as classes mais relevantes do módulo de controle da arma.

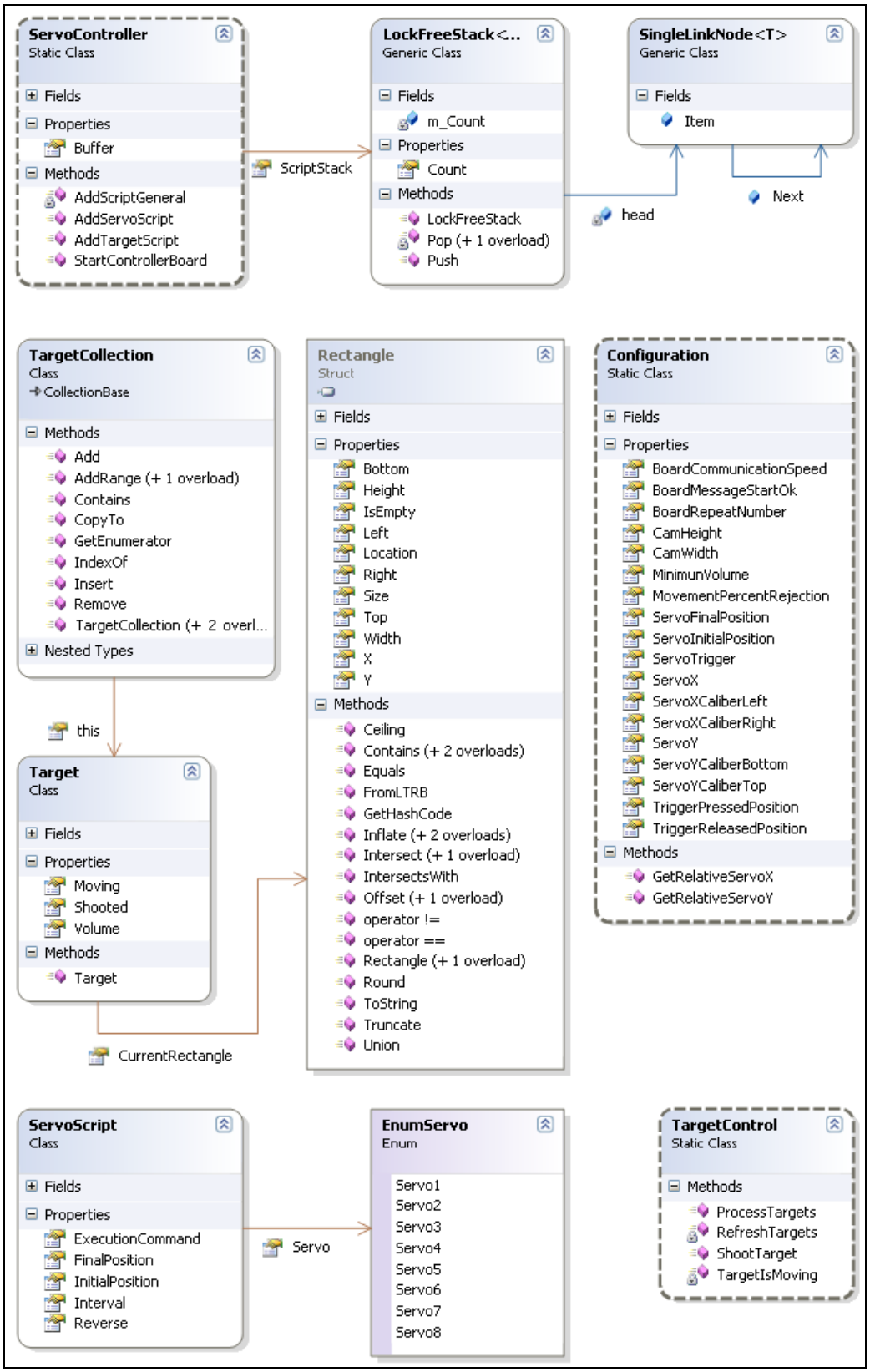


Figura 5.5 – Classes principais do módulo de controle da arma

5.4.1 Análise e Monitoração de Alvos

O procedimento de adquirir os objetos identificados pelos algoritmos de detecção de movimento, categorizá-los como alvos e disparar os eventos de movimento da arma são executados pela classe *TargetControl*.

5.4.2 Comunicação com a Placa Controladora

A inicialização da comunicação com a placa controladora, bem como o empilhamento de comandos (*ServoScripts*), são procedimentos realizados pela classe estática *ServoController.* Esta classe ainda efetua o controle da pilha de comandos que serão consumidos pela placa controladora, limitando o tamanho da pilha, descartando os comandos mais antigos.

A camada de apresentação controla a *thread* que consome os comandos de movimentação empilhados no método *AddTargetScript*.

5.4.3 Movimentação dos Servos

A geração da seqüência de caracteres (*string*) que representam os comandos de movimentação dos servomotores é produzida pela propriedade *ExecutionCommand* do objeto *ServoScript*, conforme ilustrado na Figura 5.5

5.5 Camada de Apresentação

A camada de apresentação, composta das interfaces gráficas, centraliza toda a interação com o usuário para controle do protótipo. Além disso, ela é responsável por iniciar a *thread* que retira os comandos de movimentação da pilha de comandos (módulo de controle da arma) e enviar para a placa controladora.

A Figura 5.6 mostra a interface principal da aplicação, bem como suas respectivas áreas de conteúdo.

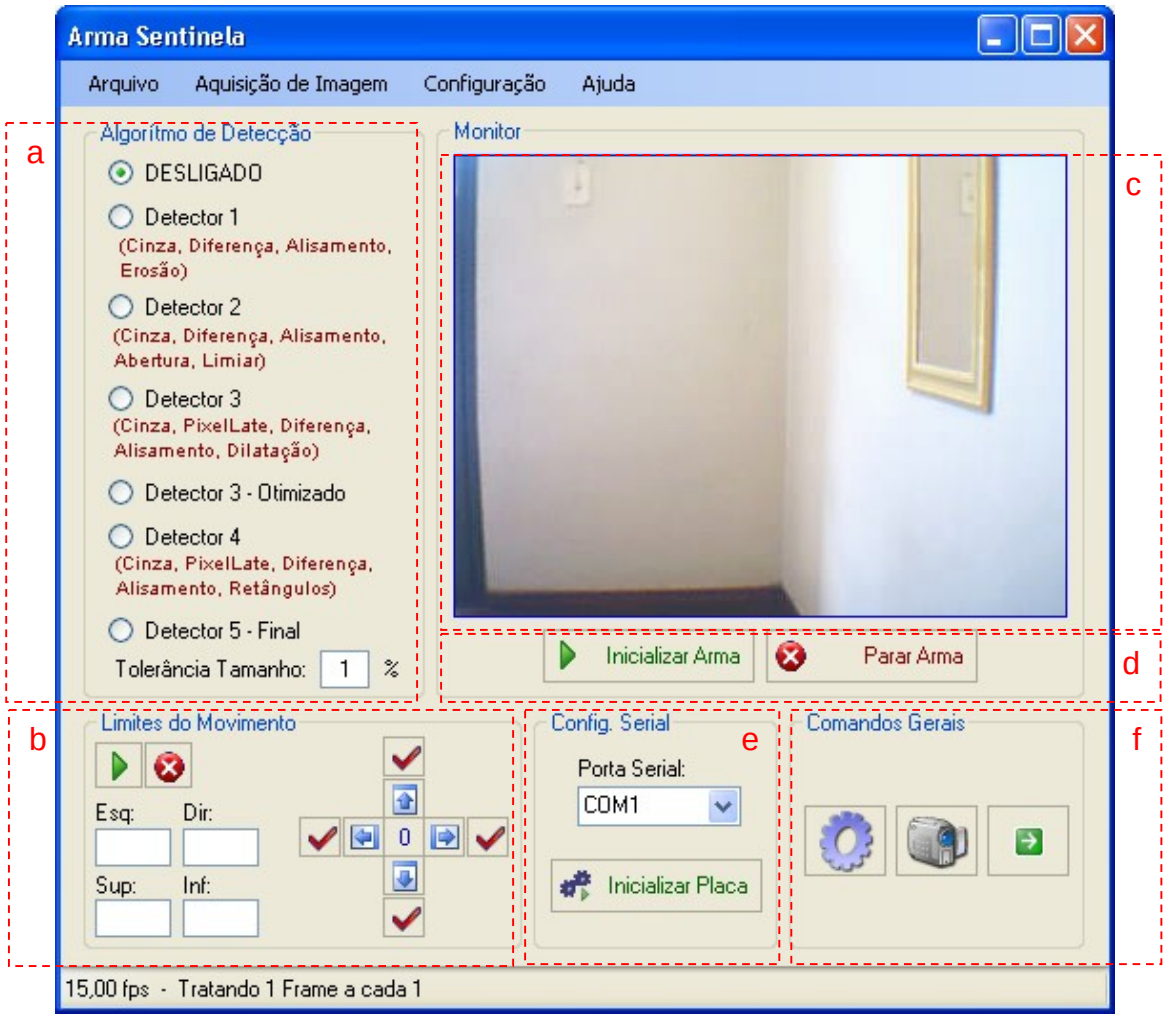


Figura 5.6 - Interface principal da camada de apresentação

Na Figura 5.6, os quadros destacados representam, respectivamente:

a) Grupo de seleção de algoritmos de detecção de movimento
b) Quadro de calibração manual dos servomotores
c) Quadro de visualização de imagens da câmera e alvos
d) Botões de comandos da arma
e) Configuração de conexão serial
f) Comandos gerais de seleção e inicialização de câmera e abertura da interface de comandos manuais dos motores.

5.5.1 Inicialização da Placa

Para iniciar a comunicação basta selecionar a porta no *combo* “Porta Serial” e clicar no botão “Inicializar Placa”, conforme Figura 5.6e.

5.5.2 Calibração dos Motores

O procedimento de calibração manual dos motores é necessário para poder-se calcular a relação da posição angular dos motores com os pontos da imagem. Os comandos da interface utilizados para calibração são os da Figura 5.6b. Os cálculos são realizados na classe *TargetControl*, utilizando os parâmetros definidos na classe *Configuration*.

5.5.3 Seleção do Algoritmo de Detecção de Movimento

Existem 6 opções para escolha do algoritmo de detecção de movimento a ser utilizado. Apenas o algoritmo 5 “Detector 5 – final” é utilizado para a operação da arma. Os demais estão incluídos na interface para demonstrar as variações de resultado obtidas com diferentes seqüências de aplicação de filtros, conforme Figura 5.6a.

5.5.4 Comandos de Inicialização da Câmera

Existem dois botões destinados ao controle da câmera. O primeiro (Figura 5.7a) abre a janela de seleção de dispositivo local (Figura 5.8).

(a) (b)

Figura 5.7 - Botões para controle da câmera

O segundo botão (Figura 5.7b) inicializa a câmera.

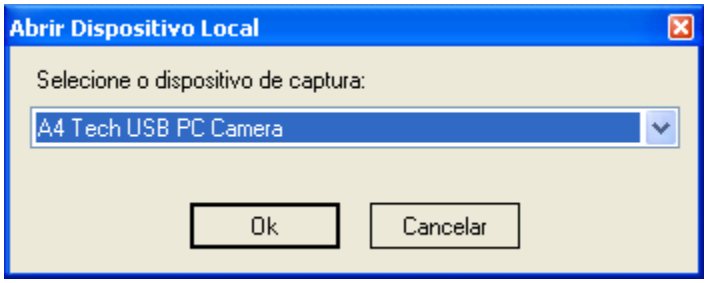


Figura 5.8 - Interface para seleção de câmera

5.5.5 Janela de Monitoração

Conforme ilustram as Figuras 5.6c e 5.9, há uma janela na interface principal que exibe a cena atual da câmera, bem como a identificação dos alvos detectados.

Figura 5.9 - Interface principal exibindo alvo identificado

5.5.6 Interface de Geração de Comandos Manuais

A função de posicionamento manual dos motores foi feita na fase inicial do projeto para testes de comportamento dos servos. A função foi mantida por sua utilidade na programação de movimentos pré-definidos da arma, ou testes de posicionamento dos motores. A interface pode ser acionada a partir do botão ilustrado na Figura 5.10.

Figura 5.10 - Botão de acionamento da interface de comandos manuais

A interface de comandos manuais está ilustrada na Figura 5.11. Nela pode-se configurar a conexão serial e definir os parâmetros de configuração da comunicação enviados para a inicialização da placa controladora.

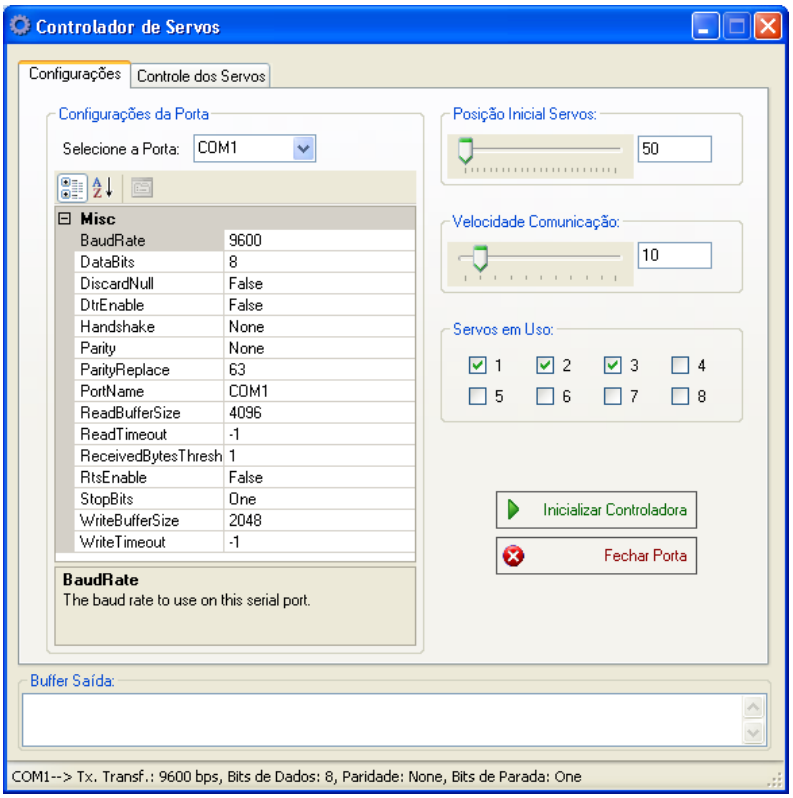


Figura 5.11 - Interface de comandos manuais - Aba Configurações

A Figura 5.12 mostra a aba de controle dos servos com alguns comandos de movimentação empilhados para execução, que podem ser estacionários (mover para uma posição e parar) ou intermitentes (variar de uma posição “A” para uma posição “B” e vice-versa, até o próximo comando para o servo em questão)

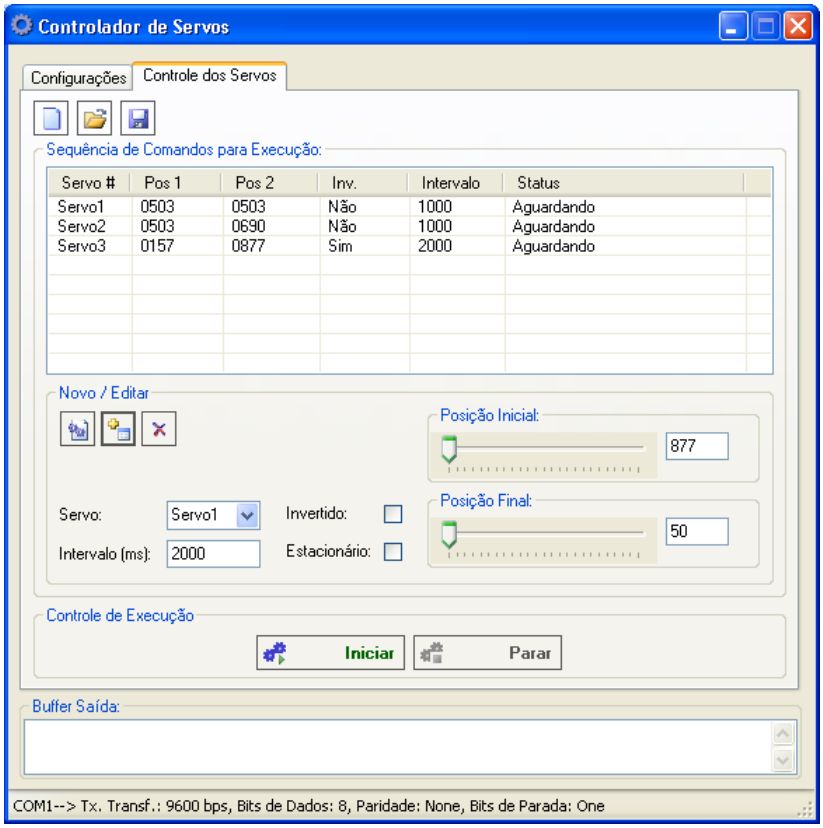


Figura 5.12 - Interface de comandos manuais - Aba Controle dos Servos

5.5.7 Comandos de Controle da Arma

Existem dois comandos de controle da arma: inicialização e parada (Figura 5.6d). Esses comandos ativam ou desativam a *thread* denominado *backWorker* que executa os comandos de movimentação.

6 CONSIDERAÇÕES FINAIS

Este capítulo contém as considerações finais sobre o trabalho, divididas em quatro subseções, a saber: dificuldades encontradas, resultados obtidos, conclusões e sugestões para trabalhos futuros.

6.1 Dificuldades Encontradas

Algumas dificuldades foram encontradas durante o desenvolvimento deste trabalho e serão descritas neste tópico com o intuito de explicitar a experiência do projeto, visando contribuir com trabalhos futuros.

A primeira dificuldade (e talvez a mais relevante) foi quanto à aquisição dos equipamentos necessários para a construção do protótipo. No mercado nacional não se encontram alguns equipamentos comuns para projetos de eletrônica ou servomecanismo. Há ainda as barreiras alfandegárias que dificultam e até mesmo inviabilizam (questões orçamentárias e de tempo) a aquisição de equipamentos produzidos fora do Brasil. Julga-se digno de nota, pois este tipo de percalço além de causar impacto considerável nos cronogramas do projeto, pode inviabilizar a realização dos modelos inicialmente projetados.

Neste projeto, os componentes inicialmente propostos foram substituídos por alternativas nacionais similares, porém com impacto para todo o modelo. Exemplo disso é o caso da placa controladora que, por ser projetada para o fim específico de montagem de animatrônicos, possui funcionalidades embutidas e não desligáveis que precisaram ser adaptadas através do software. Por exemplo, o fato de a placa ser preparada para executar movimentos intermitentes e repetitivos não possibilita o posicionamento simultâneo dos servomotores, havendo necessidade de envio dos comandos em seqüência, de forma não concorrente.

Outra dificuldade que se traz a baila, diz respeito a grande quantidade de conceitos de processamento e análise de imagem que precisaram ser estudadas e compreendidas para a correta utilização dos filtros e algoritmos relacionados a este tema.

6.2 Resultados Obtidos

Um dos resultados obtidos mais relevantes com relação ao protótipo foi o tempo de resposta (performance) dos algoritmos. Por várias vezes houve a necessidade de se refatorar os códigos para que produzissem resultados aceitáveis em termos de tempo de processamento. Uma das soluções mais significativas foi a restrição do tamanho da pilha de comandos que são enviados para a placa controladora, de forma que os comandos que excedem o tamanho previsto da pilha são descartados e não processados, evitando que eventos ocorridos há vários segundos influenciem o direcionamento da arma, causando efeitos indesejáveis como retardo na perseguição do alvo ou miras falhas.

Em média, o tempo decorrido entre a presença do alvo e a finalização do movimento é de 2 segundos, para um único alvo. Para múltiplos alvos o tempo médio de reação é proporcional a quantidade de alvos, porém o software prioriza os alvos em movimento que ou que ainda não foram alvejados.

Além da limitação do tamanho de pilha, utilizou-se uma implementação de pilha com acesso não exclusivo (*lock-free stack*), evitando *locks* de memória para controlar concorrência ao acesso da pilha, diminuindo o consumo de recursos de *cache* e processador.

Deve-se considerar ainda alguns resultados secundários, como contribuições metodológicas e práticas deste trabalho, sendo:

a) Descrição detalhada de conceitos e fundamentos de processamento de imagem;
b) Desenvolvimento de algoritmos eficientes para processamento e análise de imagem;
c) Técnicas de empilhamento com produtor (*push*) e consumidor (*pop*) executados por *threads* distintas e concorrentes e;
d) Alto grau de encapsulamento e componentização do código-fonte do software, possibilitando reuso e utilização de objetos ou algoritmos de forma parcial por trabalhos futuros.

6.3 Conclusões

A implementação de um protótipo de arma sentinela, utilizando técnicas de processamento de imagens e servomecanismo, conforme objetivos delineados e descritos no início deste trabalho, foram sucedidos.

O método utilizado para detecção de movimento através de processamento e análise de imagens mostrou-se uma alternativa eficiente em substituição aos sensores mecânicos, eletromagnéticos, térmicos ou acústicos, para o caso proposto. Uma grande gama de aplicações pode ser implementada com a técnica apresentada, e algumas delas estão sugeridas na subseção 6.4.

Percebeu-se que com a utilização de análise de imagens a possibilidade de dar-se maior complexidade computacional, aumentando o número de parâmetros que podem ser avaliados para a tomada de decisões que influenciem o comportamento do circuito, é muito abrangente e promissora. Circuitos baseados em eventos booleanos podem ser evoluídos para analisar características factuais ou morfológicas dos eventos que são percebidos pela câmera, podendo-se tomar decisões baseadas em comportamentos ou situações específicas. Um exemplo disso foi o recurso de rejeição de alvos que não atinjam uma determinada área mínima, ou a priorização de alvos em movimento e não alvejados. Acredita-se que utilizando a alternativa de detecção de movimento através de análise de imagens o grau de eficiência de sistemas baseados em sensores de presença ou de movimento pode ser aumentado significativamente.

6.4 Sugestões de Trabalhos Futuros

Diversas linhas de pesquisa podem ser criadas a partir do trabalho aqui apresentado. As sugestões estão licitadas abaixo e serão divididas em dois grupos: “evoluções do trabalho atual” e “outras linhas de pesquisa”.

a) Evoluções do trabalho atual

- Criação de uma cena de fundo panorâmica (360º) que seja sobreposta gradativamente, conforme a posição da arma no eixo de giro;
- Acoplamento da câmera à arma, de forma que a detecção do alvo não dependa do cenário de fundo;
- Desenvolvimento de um sistema de autenticação para acesso seguro através do ambiente monitorado;
- Opção de escolha de tipo de disparo a ser efetuado, de acordo com o tipo de alvo, ou seja, caso o alvo seja reconhecido como uma figura humana, efetuar os disparos como letais ou imobilizadores, de acordo com o modo de operação configurado;
- Utilização de câmeras com maior definição, tratando os impactos de performance e tempo de resposta.

b) Outras linhas de pesquisa:

- Desenvolvimento de um sistema de vigilância através de câmeras que persista somente os trechos de vídeo onde haja eventos de interesse, com opções de configuração da qualidade do vídeo desejada;
- Desenvolvimento de um sistema de alerta de invasão de ambientes monitorados, que envie via *web* o vídeo ou fotos do evento que disparou o alarme;
- Desenvolvimento de um sistema de contagem de veículos ou pessoas que passam por um determinado ambiente;
- Nos moldes do projeto atual, porém com uma câmera em movimento, desenvolver protótipo de uma câmera perseguidora, que acompanhe um objeto de escolha do usuário.

REFERÊNCIAS

E. STRINGA, S. Regazzoni, Real-Time Video-Shot Detection for Scene Surveillance Applications IEEE Transactions on Image Processing, Vol. 9, no. 1, 2000: p. 69-79.

FERNANDES, J.P. Barbosa. Transmissão Multinível e Detecção e Correção de Erros no Projeto Transmissão Alternativa de Dados. Centro Universitário de Brasília – UniCEUB, Faculdade de Ciências Exatas e de Tecnologia – FAET, Curso de Engenharia da Computação. 2005: p. 12-13

SOARES, André Borin; Figueiró, Thiago; Susin, Altamiro. Artigo: Caracterização do desempenho de métodos de detecção de movimento aplicado a localização de pessoas através de visão computacional. Inst. de Informática - UFRGS , 2004: p. 1.
Disponível em: http://www.lapsi.eletro.ufrgs.br/~figueiro/SIDEEmov.pdf
Acesso em 20/04/2007

TAKAHASHI, Kazuhiko; SAKAGUCHI, Tatsumi; OHYA, Jun. Real-time estimation of human body postures using Kalman filter. In: INTERNATIONAL WORKSHOP ON ROBOT AND HUMAN INTERACTION, 8. 1999, Roma. Anais... Roma: 1999. p. 189-194. Disponível em:
http://www.mic.atr.co.jp/~tatsu/research/publication/pubtatsu.html
Acesso em: 04 de Abril 2007.

WHATIS. The Leading IT Encyclopedia and Learning Center.
Disponível em: http://whatis.techtarget.com/definition/0,,sid9_gci212900,00.html
Acesso em: 04 de Abril 2007.

YOUNG, I.T.; Gerbrands, J.J.; Van Vilet, L.J. Fundamentals of Image Processing. - Quantitative Imaging Group - Department of Imaging Science and Technology, Faculty of Applied Sciences. Delft University of Technology – Delft The Netherlands
Disponível em: http://www.ph.tn.tudelft.nl/Courses/FIP/noframes/fip.html
Acesso em: 05 de Abril 2007

APÊNDICE A – CÓDIGO FONTE DO SOFTWARE CONTROLADOR

Pacote SentryGun.Motion

AVIReader.cs

```
namespace SentryGun.AVI
{
	using System;
	using System.Drawing;
	using System.Drawing.Imaging;
	using System.Runtime.InteropServices;

	/// <summary>
	/// Reading AVI files using Video for Windows
	/// </summary>
	public class AVIReader : IDisposable
	{
		private IntPtr file;
		private IntPtr stream;
		private IntPtr getFrame;

		private int           width;
		private int           height;
		private int           position;
		private int           start;
		private int           length;
		private float  rate;
		private string codec;

		// Width property
		public int Width
		{
			get { return width; }
		}
		// Height property
		public int Height
		{
			get { return height; }
		}
		// FramesRate property
		public float FrameRate
		{
			get { return rate; }
		}
		// CurrentPosition property
		public int CurrentPosition
		{
			get { return position; }
			set
			{
		 try
		 {
		     if ((value < start) || (value >= start + length))
		         position = start;
		     else
		         position = value;
		 }
		 catch(Exception ex)
		 {
		     throw new Exception("Erro ao definir propriedade", ex);
		 }
			}
		}
		// Length property
		public int Length
		{
			get { return length; }
```

```
   }
   // Codec property
   public string Codec
   {
          get { return codec; }
   }


   // Constructor
   public AVIReader()
   {
try
{
    Win32.AVIFileInit();
}
catch(Exception ex)
{
    throw new Exception("Erro ao instanciar objeto", ex);
}
   }

   // Desctructor
   ~AVIReader()
   {
try
{
    Dispose(false);
}
catch (Exception ex)
{
    throw new Exception("Erro ao destruir objeto", ex);
}
   }

   // Free all unmanaged resources
   public void Dispose()
   {
try
{
    Dispose(true);
    // Remove me from the Finalization queue
    GC.SuppressFinalize(this);
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar dispose", ex);
}
   }

   protected virtual void Dispose(bool disposing)
   {
try
{
    if (disposing)
    {
        // Dispose managed resources

        // there is nothing managed yet
    }
    Close();
    Win32.AVIFileExit();
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar dispose", ex);
}
   }

   // Open an AVI file
   public void Open(string fname)
   {
try
{
    // close previous file
    Close();

    // open file
```

```
                if  (Win32.AVIFileOpen(out  file,  fname,  Win32.OpenFileMode.ShareDenyWrite,
IntPtr.Zero) != 0)
                    throw new ApplicationException("Failed opening file");

                // get first video stream
                if (Win32.AVIFileGetStream(file, out stream, Win32.mmioFOURCC("vids"), 0) !=
0)
                    throw new ApplicationException("Failed getting video stream");

                // get stream info
                Win32.AVISTREAMINFO info = new Win32.AVISTREAMINFO();
                Win32.AVIStreamInfo(stream, ref info, Marshal.SizeOf(info));

                width = info.rcFrame.right;
                height = info.rcFrame.bottom;
                position = info.dwStart;
                start = info.dwStart;
                length = info.dwLength;
                rate = (float)info.dwRate / (float)info.dwScale;
                codec = Win32.decode_mmioFOURCC(info.fccHandler);

                // prepare decompressor
                Win32.BITMAPINFOHEADER bih = new Win32.BITMAPINFOHEADER();

                bih.biSize = Marshal.SizeOf(bih.GetType());
                bih.biWidth = width;
                bih.biHeight = height;
                bih.biPlanes = 1;
                bih.biBitCount = 24;
                bih.biCompression = 0; // BI_RGB

                // get frame open object
                if ((getFrame = Win32.AVIStreamGetFrameOpen(stream, ref bih)) == IntPtr.Zero)
                {
                    bih.biHeight = -height;

                    if   ((getFrame   =   Win32.AVIStreamGetFrameOpen(stream,   ref   bih))   ==
IntPtr.Zero)
                        throw new ApplicationException("Failed initializing decompressor");
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao abrir AVI", ex);
            }
               }

               // Close file
               public void Close()
               {
            try
            {
                // release frame open object
                if (getFrame != IntPtr.Zero)
                {
                    Win32.AVIStreamGetFrameClose(getFrame);
                    getFrame = IntPtr.Zero;
                }
                // release stream
                if (stream != IntPtr.Zero)
                {
                    Win32.AVIStreamRelease(stream);
                    stream = IntPtr.Zero;
                }
                // release file
                if (file != IntPtr.Zero)
                {
                    Win32.AVIFileRelease(file);
                    file = IntPtr.Zero;
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao fechar AVI", ex);
            }
               }
```

```
            // Get next video frame
            public Bitmap GetNextFrame()
            {
        try
        {
            // get frame at specified position
            IntPtr pdib = Win32.AVIStreamGetFrame(getFrame, position);
            if (pdib == IntPtr.Zero)
                throw new ApplicationException("Failed getting frame");

            Win32.BITMAPINFOHEADER bih;

            // copy BITMAPINFOHEADER from unmanaged memory
            bih           =           (Win32.BITMAPINFOHEADER)Marshal.PtrToStructure(pdib,
typeof(Win32.BITMAPINFOHEADER));

            // create new bitmap
            Bitmap bmp = new Bitmap(width, height, PixelFormat.Format24bppRgb);

            // lock bitmap data
            BitmapData bmData = bmp.LockBits(
                new Rectangle(0, 0, width, height),
                ImageLockMode.ReadWrite,
                PixelFormat.Format24bppRgb);

            // copy image data
            int srcStride = bmData.Stride;      // width * 3;
            int dstStride = bmData.Stride;

            // check image direction
            if (bih.biHeight > 0)
            {
                // it`s a bottom-top image
                int dst = bmData.Scan0.ToInt32() + dstStride * (height - 1);
                int src = pdib.ToInt32() + Marshal.SizeOf(typeof(Win32.BITMAPINFOHEADER));

                for (int y = 0; y < height; y++)
                {
                    Win32.memcpy(dst, src, srcStride);
                    dst -= dstStride;
                    src += srcStride;
                }
            }
            else
            {
                // it`s a top bootom image
                int dst = bmData.Scan0.ToInt32();
                int src = pdib.ToInt32() + Marshal.SizeOf(typeof(Win32.BITMAPINFOHEADER));

                if (srcStride != dstStride)
                {
                    // copy line by line
                    for (int y = 0; y < height; y++)
                    {
                        Win32.memcpy(dst, src, srcStride);
                        dst += dstStride;
                        src += srcStride;
                    }
                }
                else
                {
                    // copy the whole image
                    Win32.memcpy(dst, src, srcStride * height);
                }
            }

            // unlock bitmap data
            bmp.UnlockBits(bmData);

            position++;

            return bmp;
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao obter próximo frame", ex);
        }
```

```
        }
        }
}
```

AVIWriter.cs

```
namespace SentryGun.AVI
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;
        using System.Runtime.InteropServices;

        public class AVIWriter : IDisposable
        {
                private IntPtr file;
                private IntPtr stream;
                private IntPtr streamCompressed;
                private IntPtr buf = IntPtr.Zero;

                private int           width;
                private int           height;
                private int           stride;
                private string codec = "DIB ";
                private int           quality = -1;
                private int           rate = 25;
                private int           position;

                // CurrentPosition property
                public int CurrentPosition
                {
                        get { return position; }
                }
                // Width property
                public int Width
                {
                        get
                        {
                 try
                 {
                     return (buf != IntPtr.Zero) ? width : 0;
                 }
                 catch (Exception ex)
                 {
                     throw new Exception("Erro ao obter propriedade");
                 }
                        }
                }
                // Height property
                public int Height
                {
                        get
            {
                 try
                 {
                     return (buf != IntPtr.Zero) ? height : 0;
                 }
                 catch (Exception ex)
                 {
                     throw new Exception("Erro ao obter propriedade");
                 }
                        }
                }
                // Codec property
                public string Codec
                {
                        get { return codec; }
                        set { codec = value; }
                }
                // Quality property
                public int Quality
                {
                        get { return quality; }
                        set { quality = value; }
```

```
   }
   // FrameRate property
   public int FrameRate
   {
          get { return rate; }
          set { rate = value; }
   }


   // Constructor
   public AVIWriter()
   {
try
{
    Win32.AVIFileInit();
}
catch (Exception ex)
{
    throw new Exception("Erro ao instanciar objeto", ex);
}
   }
   public AVIWriter(string codec) : this()
   {
try
{
    this.codec = codec;
}
catch (Exception ex)
{
    throw new Exception("Erro ao instanciar objeto", ex);
}
   }

   // Desctructor
   ~AVIWriter()
   {
try
{
    Dispose(false);
}
catch (Exception ex)
{
    throw new Exception("Erro ao destruir objeto", ex);
}
   }

   // Free all unmanaged resources
   public void Dispose()
   {
try
{
    Dispose(true);
    // Remove me from the Finalization queue
    GC.SuppressFinalize(this);
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar dispose", ex);
}
   }

   protected virtual void Dispose(bool disposing)
   {
try
{
    if (disposing)
    {
        // Dispose managed resources

        // there is nothing managed yet
    }

    Close();
    Win32.AVIFileExit();
}
catch (Exception ex)
{
```

```
                throw new Exception("Erro ao executar dispose", ex);
            }

               }

               // Create new AVI file
               public void Open(string fname, int width, int height)
               {

            try
            {
                // close previous file
                Close();

                // calculate stride
                stride = width * 3;
                int r = stride % 4;
                if (r != 0)
                    stride += (4 - r);

                // create new file
                if   (Win32.AVIFileOpen(out   file,   fname,   Win32.OpenFileMode.Create   |
Win32.OpenFileMode.Write, IntPtr.Zero) != 0)
                    throw new ApplicationException("Failed opening file");

                this.width = width;
                this.height = height;

                // describe new stream
                Win32.AVISTREAMINFO info = new Win32.AVISTREAMINFO();

                info.fccType = Win32.mmioFOURCC("vids");
                info.fccHandler = Win32.mmioFOURCC(codec);
                info.dwScale = 1;
                info.dwRate = rate;
                info.dwSuggestedBufferSize = stride * height;

                // create stream
                if (Win32.AVIFileCreateStream(file, out stream, ref info) != 0)
                    throw new ApplicationException("Failed creating stream");

                // describe compression options
                Win32.AVICOMPRESSOPTIONS opts = new Win32.AVICOMPRESSOPTIONS();

                opts.fccHandler = Win32.mmioFOURCC(codec);
                opts.dwQuality = quality;

                //
                // Win32.AVISaveOptions(stream, ref opts, IntPtr.Zero);

                // create compressed stream
                if  (Win32.AVIMakeCompressedStream(out  streamCompressed,  stream,  ref  opts,
IntPtr.Zero) != 0)
                    throw new ApplicationException("Failed creating compressed stream");

                // describe frame format
                Win32.BITMAPINFOHEADER bih = new Win32.BITMAPINFOHEADER();

                bih.biSize = Marshal.SizeOf(bih.GetType());
                bih.biWidth = width;
                bih.biHeight = height;
                bih.biPlanes = 1;
                bih.biBitCount = 24;
                bih.biSizeImage = 0;
                bih.biCompression = 0; // BI_RGB

                // set frame format
                if      (Win32.AVIStreamSetFormat(streamCompressed,      0,      ref      bih,
Marshal.SizeOf(bih.GetType())) != 0)
                    throw new ApplicationException("Failed creating compressed stream");

                // alloc unmanaged memory for frame
                buf = Marshal.AllocHGlobal(stride * height);

                position = 0;
            }
            catch (Exception ex)
```

```
{
    throw new Exception("Erro ao abrir AVI", ex);
}
   }

   // Close file
   public void Close()
   {
try
{
    // free unmanaged memory
    if (buf != IntPtr.Zero)
    {
        Marshal.FreeHGlobal(buf);
        buf = IntPtr.Zero;
    }
    // release compressed stream
    if (streamCompressed != IntPtr.Zero)
    {
        Win32.AVIStreamRelease(streamCompressed);
        streamCompressed = IntPtr.Zero;
    }
    // release stream
    if (stream != IntPtr.Zero)
    {
        Win32.AVIStreamRelease(stream);
        stream = IntPtr.Zero;
    }
    // release file
    if (file != IntPtr.Zero)
    {
        Win32.AVIFileRelease(file);
        file = IntPtr.Zero;
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao fechar AVI", ex);
}
   }

   // Add new frame to the AVI file
   public void AddFrame(System.Drawing.Bitmap bmp)
   {
try
{
    // check image dimension
    if ((bmp.Width != width) || (bmp.Height != height))
        throw new ApplicationException("Invalid image dimension");

    // lock bitmap data
    BitmapData bmData = bmp.LockBits(
        new Rectangle(0, 0, width, height),
        ImageLockMode.ReadWrite, PixelFormat.Format24bppRgb);

    // copy image data
    int srcStride = bmData.Stride;
    int dstStride = stride;

    int src = bmData.Scan0.ToInt32() + srcStride * (height - 1);
    int dst = buf.ToInt32();

    for (int y = 0; y < height; y++)
    {
        Win32.memcpy(dst, src, dstStride);
        dst += dstStride;
        src -= srcStride;
    }

    // unlock bitmap data
    bmp.UnlockBits(bmData);

    // write to stream
    if (Win32.AVIStreamWrite(streamCompressed, position, 1, buf,
        stride * height, 0, IntPtr.Zero, IntPtr.Zero) != 0)
        throw new ApplicationException("Failed adding frame");
```

```
                position++;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao adicionar frame", ex);
            }
               }
        }
}
```

ByteArrayUtils.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;

        /// <summary>
        /// Some array utilities
        /// </summary>
        internal class ByteArrayUtils
        {
               // Check if the array contains needle on specified position
               public static bool Compare(byte[] array, byte[] needle, int startIndex)
               {
            try
            {
                int needleLen = needle.Length;
                // compare
                for (int i = 0, p = startIndex; i < needleLen; i++, p++)
                {
                    if (array[p] != needle[i])
                    {
                        return false;
                    }
                }
                return true;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao verificar array", ex);
            }
               }

               // Find subarray in array
               public static int Find(byte[] array, byte[] needle, int startIndex, int count)
               {
            try
            {
                int needleLen = needle.Length;
                int index;

                while (count >= needleLen)
                {
                    index = Array.IndexOf(array, needle[0], startIndex, count - needleLen +
1);

                    if (index == -1)
                        return -1;

                    int i, p;
                    // check for needle
                    for (i = 0, p = index; i < needleLen; i++, p++)
                    {
                        if (array[p] != needle[i])
                        {
                            break;
                        }
                    }

                    if (i == needleLen)
                    {
                        // found needle
                        return index;
                    }

                    count -= (index - startIndex + 1);
```

```
                    startIndex = index + 1;
                }
                return -1;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao procurar subarray em array", ex);
            }
               }
        }
}
```

CameraEvents.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;
        using System.Drawing.Imaging;

        // NewFrame delegate
        public delegate void CameraEventHandler(object sender, CameraEventArgs e);

        /// <summary>
        /// Camera event arguments
        /// </summary>
        public class CameraEventArgs : EventArgs
        {
                private System.Drawing.Bitmap bmp;

                // Constructor
                public CameraEventArgs(System.Drawing.Bitmap bmp)
                {
                        this.bmp = bmp;
                }

                // Bitmap property
                public System.Drawing.Bitmap Bitmap
                {
                        get { return bmp; }
                }
        }
}
```

CaptureDevice.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;
        using System.IO;
        using System.Threading;
        using System.Runtime.InteropServices;
        using System.Net;

    using SentryGun.Dshow;
    using SentryGun.Dshow.Core;

        /// <summary>
        /// CaptureDevice - capture video from local device
        /// </summary>
        public class CaptureDevice : IVideoSource
        {
                private string source;
                private object userData = null;
                private int           framesReceived;

                private Thread thread = null;
                private ManualResetEvent stopEvent = null;

                // new frame event
                public event CameraEventHandler NewFrame;

                // VideoSource property
                public virtual string VideoSource
```

```
        {
                get { return source; }
                set { source = value; }
        }
        // Login property
        public string Login
        {
                get { return null; }
                set { }
        }
        // Password property
        public string Password
        {
                get { return null; }
                set {  }
        }
        // FramesReceived property
        public int FramesReceived
        {
                get
                {
                        int frames = framesReceived;
                        framesReceived = 0;
                        return frames;
                }
        }
        // BytesReceived property
        public int BytesReceived
        {
                get { return 0; }
        }
        // UserData property
        public object UserData
        {
                get { return userData; }
                set { userData = value; }
        }
        // Get state of the video source thread
        public bool Running
        {
    get
                {
            try
            {
                if (thread != null)
                {
                    if (thread.Join(0) == false)
                        return true;

                    // the thread is not running, so free resources
                    Free();
                }
                return false;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao obter propriedade", ex);
            }
                }
}


        // Constructor
        public CaptureDevice()
        {
        }

        // Start work
        public void Start()
        {
    try
    {
        if (thread == null)
        {
            framesReceived = 0;

            // create events
```

```
        stopEvent = new ManualResetEvent(false);

        // create and start new thread
        thread = new Thread(new ThreadStart(WorkerThread));
        thread.Name = source;
        thread.Start();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar Dispositivo Local", ex);
}
   }

   // Signal thread to stop work
   public void SignalToStop()
   {
try
{
    // stop thread
    if (thread != null)
    {
        // signal to stop
        stopEvent.Set();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao enviar sinal de parada", ex);
}
   }

   // Wait for thread stop
   public void WaitForStop()
   {
try
{
    if (thread != null)
    {
        // wait for thread stop
        thread.Join();

        Free();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao aguardar thread", ex);
}
   }

   // Abort thread
   public void Stop()
   {
try
{
    if (this.Running)
    {
        thread.Abort();
        // WaitForStop();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao parar Dispositivo Local", ex);
}
   }

   // Free resources
   private void Free()
   {
try
{
    thread = null;

    // release events
    stopEvent.Close();
```

```
                stopEvent = null;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao liberar recursos", ex);
            }
               }

               // Thread entry point
               public void WorkerThread()
               {
            try
            {
                // grabber
                Grabber grabber = new Grabber(this);

                // objects
                object graphObj = null;
                object sourceObj = null;
                object grabberObj = null;

                // interfaces
                IGraphBuilder graph = null;
                IBaseFilter sourceBase = null;
                IBaseFilter grabberBase = null;
                ISampleGrabber sg = null;
                IMediaControl mc = null;

                try
                {
                    // Get type for filter graph
                    Type srvType = Type.GetTypeFromCLSID(Clsid.FilterGraph);
                    if (srvType == null)
                        throw new ApplicationException("Failed creating filter graph");

                    // create filter graph
                    graphObj = Activator.CreateInstance(srvType);
                    graph = (IGraphBuilder)graphObj;

                    // ----
                    UCOMIBindCtx bindCtx = null;
                    UCOMIMoniker moniker = null;
                    int n = 0;

                    // create bind context
                    if (Win32.CreateBindCtx(0, out bindCtx) == 0)
                    {
                        // convert moniker`s string to a moniker
                        if (Win32.MkParseDisplayName(bindCtx, source, ref n, out moniker) ==
0)
                        {
                            // get device base filter
                            Guid filterId = typeof(IBaseFilter).GUID;
                            moniker.BindToObject(null, null, ref filterId, out sourceObj);

                            Marshal.ReleaseComObject(moniker);
                            moniker = null;
                        }
                        Marshal.ReleaseComObject(bindCtx);
                        bindCtx = null;
                    }
                    // ----

                    if (sourceObj == null)
                        throw new ApplicationException("Failed creating device object for
moniker");

                    sourceBase = (IBaseFilter)sourceObj;

                    // Get type for sample grabber
                    srvType = Type.GetTypeFromCLSID(Clsid.SampleGrabber);
                    if (srvType == null)
                        throw new ApplicationException("Failed creating sample grabber");

                    // create sample grabber
                    grabberObj = Activator.CreateInstance(srvType);
                    sg = (ISampleGrabber)grabberObj;
```

```
                    grabberBase = (IBaseFilter)grabberObj;

                    // add source filter to graph
                    graph.AddFilter(sourceBase, "source");
                    graph.AddFilter(grabberBase, "grabber");

                    // set media type
                    AMMediaType mt = new AMMediaType();
                    mt.majorType = MediaType.Video;
                    mt.subType = MediaSubType.RGB24;
                    sg.SetMediaType(mt);

                    // connect pins
                    if             (graph.Connect(DSTools.GetOutPin(sourceBase,             0),
DSTools.GetInPin(grabberBase, 0)) < 0)
                        throw new ApplicationException("Failed connecting filters");

                    // get media type
                    if (sg.GetConnectedMediaType(mt) == 0)
                    {
                        VideoInfoHeader                          vih                          =
(VideoInfoHeader)Marshal.PtrToStructure(mt.formatPtr, typeof(VideoInfoHeader));

                        System.Diagnostics.Debug.WriteLine("width = " + vih.BmiHeader.Width +
", height = " + vih.BmiHeader.Height);
                        grabber.Width = vih.BmiHeader.Width;
                        grabber.Height = vih.BmiHeader.Height;
                        mt.Dispose();
                    }

                    // render
                    graph.Render(DSTools.GetOutPin(grabberBase, 0));

                    //
                    sg.SetBufferSamples(false);
                    sg.SetOneShot(false);
                    sg.SetCallback(grabber, 1);

                    // window
                    IVideoWindow win = (IVideoWindow)graphObj;
                    win.put_AutoShow(false);
                    win = null;


                    // get media control
                    mc = (IMediaControl)graphObj;

                    // run
                    mc.Run();

                    while (!stopEvent.WaitOne(0, true))
                    {
                        Thread.Sleep(100);
                    }
                    mc.StopWhenReady();
                }
                // catch any exceptions
                catch (Exception e)
                {
                    System.Diagnostics.Debug.WriteLine("----: " + e.Message);
                }
                // finalization block
                finally
                {
                    // release all objects
                    mc = null;
                    graph = null;
                    sourceBase = null;
                    grabberBase = null;
                    sg = null;

                    if (graphObj != null)
                    {
                        Marshal.ReleaseComObject(graphObj);
                        graphObj = null;
                    }
                    if (sourceObj != null)
```

```
                {
                    Marshal.ReleaseComObject(sourceObj);
                    sourceObj = null;
                }
                if (grabberObj != null)
                {
                    Marshal.ReleaseComObject(grabberObj);
                    grabberObj = null;
                }
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao iniciar thread de leitura de Dispositivo Local ",
ex);
        }
           }

           // new frame
           protected void OnNewFrame(Bitmap image)
           {
        try
        {
            framesReceived++;
            if ((!stopEvent.WaitOne(0, true)) && (NewFrame != null))
                NewFrame(this, new CameraEventArgs(image));
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao executar evento", ex);
        }
           }

           // Grabber
           private class Grabber : ISampleGrabberCB
           {
                  private CaptureDevice parent;
                  private int width, height;

                  // Width property
                  public int Width
                  {
                         get { return width; }
                         set { width = value; }
                  }
                  // Height property
                  public int Height
                  {
                         get { return height; }
                         set { height = value; }
                  }

                  // Constructor
                  public Grabber(CaptureDevice parent)
                  {
                         this.parent = parent;
                  }

                  //
                  public int SampleCB(double SampleTime, IntPtr pSample)
                  {
                         return 0;
                  }

                  // Callback method that receives a pointer to the sample buffer
                  public int BufferCB(double SampleTime, IntPtr pBuffer, int BufferLen)
                  {
            try
            {
                // create new image
                System.Drawing.Bitmap     img     =     new     Bitmap(width,     height,
PixelFormat.Format24bppRgb);

                // lock bitmap data
                BitmapData bmData = img.LockBits(
                    new Rectangle(0, 0, width, height),
                    ImageLockMode.ReadWrite,
```

```
                        PixelFormat.Format24bppRgb);

                    // copy image data
                    int srcStride = bmData.Stride;
                    int dstStride = bmData.Stride;

                    int dst = bmData.Scan0.ToInt32() + dstStride * (height - 1);
                    int src = pBuffer.ToInt32();

                    for (int y = 0; y < height; y++)
                    {
                        Win32.memcpy(dst, src, srcStride);
                        dst -= dstStride;
                        src += srcStride;
                    }

                    // unlock bitmap data
                    img.UnlockBits(bmData);

                    // notify parent
                    parent.OnNewFrame(img);

                    // release the image
                    img.Dispose();

                    return 0;
                }
                catch (Exception ex)
                {
                    throw new Exception("Erro ao ler buffer da imagem", ex);
                }
                      }
                }
        }
}
```

IVideoSource.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;

        /// <summary>
        /// IVideoSource interface
        /// </summary>
        public interface IVideoSource
        {
                /// <summary>
                /// New frame event - notify client about the new frame
                /// </summary>
                event CameraEventHandler NewFrame;

                /// <summary>
                /// Video source property
                /// </summary>
                string VideoSource{get; set;}

                /// <summary>
                /// Login property
                /// </summary>
                string Login{get; set;}

                /// <summary>
                /// Password property
                /// </summary>
                string Password{get; set;}

                /// <summary>
                /// FramesReceived property
                /// get number of frames the video source received from the last
                /// access to the property
                /// </summary>
                int FramesReceived{get;}

                /// <summary>
                /// BytesReceived property
```

```
                /// get number of bytes the video source received from the last
                /// access to the property
                /// </summary>
                int BytesReceived{get;}

                /// <summary>
                /// UserData property
                /// allows to associate user data with an object
                /// </summary>
                object UserData{get; set;}

                /// <summary>
                /// Get state of video source
                /// </summary>
                bool Running{get;}

                /// <summary>
                /// Start receiving video frames
                /// </summary>
                void Start();

                /// <summary>
                /// Stop receiving video frames
                /// </summary>
                void SignalToStop();

                /// <summary>
                /// Wait for stop
                /// </summary>
                void WaitForStop();

                /// <summary>
                /// Stop work
                /// </summary>
                void Stop();
        }
}
```

JPEGStream.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;
        using System.Drawing;
        using System.IO;
        using System.Threading;
        using System.Net;

        /// <summary>
        /// JPEGSource - JPEG downloader
        /// </summary>
        public class JPEGStream : IVideoSource
        {
                private string source;
                private string login = null;
                private string password = null;
                private object userData = null;
                private int           framesReceived;
                private int           bytesReceived;
                private bool   useSeparateConnectionGroup = false;
                private bool   preventCaching = false;
                private int           frameInterval = 0;           //    frame    interval    in
miliseconds

                private const int     bufSize = 512 * 1024; // buffer size
                private const int     readSize = 1024;             // portion size to read

                private Thread thread = null;
                private ManualResetEvent stopEvent = null;

                // new frame event
                public event CameraEventHandler NewFrame;

                // SeparateConnectioGroup property
```

```
// indicates to open WebRequest in separate connection group
public bool    SeparateConnectionGroup
{
       get { return useSeparateConnectionGroup; }
       set { useSeparateConnectionGroup = value; }
}
// PreventCaching property
// If the property is set to true, we are trying to prevent caching
// appneding fake parameter to URL. It's needed is client is behind
// proxy server.
public bool    PreventCaching
{
       get { return preventCaching; }
       set { preventCaching = value; }
}
// FrameInterval property - interval between frames
// If the property is set 100, than the source will produce 10 frames
// per second if it possible
public int FrameInterval
{
       get { return frameInterval; }
       set { frameInterval = value; }
}
// VideoSource property
public virtual string VideoSource
{
       get { return source; }
       set { source = value; }
}
// Login property
public string Login
{
       get { return login; }
       set { login = value; }
}
// Password property
public string Password
{
       get { return password; }
       set { password = value; }
}
// FramesReceived property
public int FramesReceived
{
       get
       {
              int frames = framesReceived;
              framesReceived = 0;
              return frames;
       }
}
// BytesReceived property
public int BytesReceived
{
       get
       {
              int bytes = bytesReceived;
              bytesReceived = 0;
              return bytes;
       }
}
// UserData property
public object UserData
{
       get { return userData; }
       set { userData = value; }
}
// Get state of the video source thread
public bool Running
{
       get
       {
 try
 {
     if (thread != null)
     {
         if (thread.Join(0) == false)
```

```
                return true;

            // the thread is not running, so free resources
            Free();
        }
        return false;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao obter propriedade", ex);
    }
          }
   }

   // Constructor
   public JPEGStream()
   {
   }

   // Start work
   public void Start()
   {
          try
{
    if (thread == null)
              {
                     framesReceived = 0;
                     bytesReceived = 0;

                     // create events
                     stopEvent  = new ManualResetEvent(false);

                     // create and start new thread
                     thread = new Thread(new ThreadStart(WorkerThread));
                     thread.Name = source;
                     thread.Start();
              }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar JPEGStream", ex);
}
   }

   // Signal thread to stop work
   public void SignalToStop()
   {
try
{
    // stop thread
    if (thread != null)
    {
        // signal to stop
        stopEvent.Set();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao enviar sinal de parada", ex);
}
}

   // Wait for thread stop
   public void WaitForStop()
   {
try
{
    if (thread != null)
    {
        // wait for thread stop
        thread.Join();

        Free();
    }
}
catch (Exception ex)
{
```

```
                throw new Exception("Erro ao aguardar Thread", ex);
            }
               }

               // Abort thread
               public void Stop()
               {
            try
            {
                if (this.Running)
                {
                    thread.Abort();
                    WaitForStop();
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao parar JPEGStream", ex);
            }
               }

               // Free resources
               private void Free()
               {
            try
            {
                thread = null;

                // release events
                stopEvent.Close();
                stopEvent = null;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao liberar recursos", ex);
            }
               }

               // Thread entry point
               public void WorkerThread()
               {
            try
            {
                byte[] buffer = new byte[bufSize];  // buffer to read stream
                HttpWebRequest req = null;
                WebResponse resp = null;
                Stream stream = null;
                Random rnd = new Random((int)DateTime.Now.Ticks);
                DateTime start;
                TimeSpan span;

                while (true)
                {
                    int read, total = 0;

                    try
                    {
                        start = DateTime.Now;

                        // create request
                        if (!preventCaching)
                        {
                            req = (HttpWebRequest)WebRequest.Create(source);
                        }
                        else
                        {
                            req       =       (HttpWebRequest)WebRequest.Create(source       +
((source.IndexOf('?') == -1) ? '?' : '&') + "fake=" + rnd.Next().ToString());
                        }
                        // set login and password
                        if ((login != null) && (password != null) && (login != ""))
                            req.Credentials = new NetworkCredential(login, password);
                        // set connection group name
                        if (useSeparateConnectionGroup)
                            req.ConnectionGroupName = GetHashCode().ToString();

                        req.Proxy.Credentials = new NetworkCredential("c1119454", "10203033");
```

```
                    // get response
                    resp = req.GetResponse();

                    // get response stream
                    stream = resp.GetResponseStream();

                    // loop
                    while (!stopEvent.WaitOne(0, true))
                    {
                        // check total read
                        if (total > bufSize - readSize)
                        {
                            System.Diagnostics.Debug.WriteLine("flushing");
                            total = 0;
                        }

                        // read next portion from stream
                        if ((read = stream.Read(buffer, total, readSize)) == 0)
                            break;

                        total += read;

                        // increment received bytes counter
                        bytesReceived += read;
                    }

                    if (!stopEvent.WaitOne(0, true))
                    {
                        // increment frames counter
                        framesReceived++;

                        // image at stop
                        if (NewFrame != null)
                        {
                            Bitmap        bmp        =        (Bitmap)Bitmap.FromStream(new
MemoryStream(buffer, 0, total));
                            // notify client
                            NewFrame(this, new CameraEventArgs(bmp));
                            // release the image
                            bmp.Dispose();
                            bmp = null;
                        }
                    }

                    // wait for a while ?
                    if (frameInterval > 0)
                    {
                        // times span
                        span = DateTime.Now.Subtract(start);
                        // miliseconds to sleep
                        int msec = frameInterval - (int)span.TotalMilliseconds;

                        while ((msec > 0) && (stopEvent.WaitOne(0, true) == false))
                        {
                            // sleeping ...
                            Thread.Sleep((msec < 100) ? msec : 100);
                            msec -= 100;
                        }
                    }
                }
                catch (WebException ex)
                {
                    System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
                    // wait for a while before the next try
                    Thread.Sleep(250);
                }
                catch (Exception ex)
                {
                    System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
                }
                finally
                {
                    // abort request
                    if (req != null)
                    {
                        req.Abort();
```

```
                        req = null;
                    }
                    // close response stream
                    if (stream != null)
                    {
                        stream.Close();
                        stream = null;
                    }
                    // close response
                    if (resp != null)
                    {
                        resp.Close();
                        resp = null;
                    }
                }

                // need to stop ?
                if (stopEvent.WaitOne(0, true))
                    break;
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao inicirar thread de leitura de JPEGStream", ex);
        }
    }
    }
}
```

MJPEGStream.cs

```
namespace SentryGun.Motion.VideoSource
{
    using System;
    using System.Drawing;
    using System.IO;
    using System.Text;
    using System.Threading;
    using System.Net;

    /// <summary>
    /// MJPEGSource - MJPEG stream support
    /// </summary>
    public class MJPEGStream : IVideoSource
    {
        private string source;
        private string login = null;
        private string password = null;
        private object userData = null;
        private int           framesReceived;
        private int           bytesReceived;
        private bool   useSeparateConnectionGroup = true;

        private const int     bufSize = 512 * 1024; // buffer size
        private const int     readSize = 1024;             // portion size to read

        private Thread thread = null;
        private ManualResetEvent stopEvent = null;
        private ManualResetEvent reloadEvent = null;

        // new frame event
        public event CameraEventHandler NewFrame;

        // SeparateConnectioGroup property
        // indicates to open WebRequest in separate connection group
        public bool    SeparateConnectionGroup
        {
            get { return useSeparateConnectionGroup; }
            set { useSeparateConnectionGroup = value; }
        }
        // VideoSource property
        public string VideoSource
        {
            get { return source; }
```

```
        set
        {
 try
 {
     source = value;
     // signal to reload
     if (thread != null)
         reloadEvent.Set();
 }
 catch (Exception ex)
 {
     throw new Exception("Erro ao definir propriedade", ex);
 }
        }
}
// Login property
public string Login
{
        get { return login; }
        set { login = value; }
}
// Password property
public string Password
{
        get { return password; }
        set { password = value; }
}
// FramesReceived property
public int FramesReceived
{
        get
        {
                int frames = framesReceived;
                framesReceived = 0;
                return frames;
        }
}
// BytesReceived property
public int BytesReceived
{
        get
        {
                int bytes = bytesReceived;
                bytesReceived = 0;
                return bytes;
        }
}
// UserData property
public object UserData
{
        get { return userData; }
        set { userData = value; }
}
// Get state of the video source thread
public bool Running
{
        get
        {
 try
 {
     if (thread != null)
     {
         if (thread.Join(0) == false)
             return true;

         // the thread is not running, so free resources
         Free();
     }
     return false;
 }
 catch (Exception ex)
 {
     throw new Exception("Erro ao obter propriedade", ex);
 }
        }
}
```

```
   // Constructor
   public MJPEGStream()
   {
   }

   // Start work
   public void Start()
   {
try
{
    if (thread == null)
    {
        framesReceived = 0;
        bytesReceived = 0;

        // create events
        stopEvent = new ManualResetEvent(false);
        reloadEvent = new ManualResetEvent(false);

        // create and start new thread
        thread = new Thread(new ThreadStart(WorkerThread));
        thread.Name = source;
        thread.Start();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar MPEGStream", ex);
}
   }

   // Signal thread to stop work
   public void SignalToStop()
   {
try
{
    // stop thread
    if (thread != null)
    {
        // signal to stop
        stopEvent.Set();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao enviar sinal de parada", ex);
}
   }

   // Wait for thread stop
   public void WaitForStop()
   {
try
{
    if (thread != null)
    {
        // wait for thread stop
        thread.Join();

        Free();
    }
}

catch (Exception ex)
{
    throw new Exception("Erro ao aguardar Thread", ex);
}
   }

   // Abort thread
   public void Stop()
   {
try
{
    if (this.Running)
```

```
                {
                    thread.Abort();
                    WaitForStop();
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao parar MPEGStream", ex);
            }
               }

               // Free resources
               private void Free()
               {
            try
            {
                thread = null;

                // release events
                stopEvent.Close();
                stopEvent = null;
                reloadEvent.Close();
                reloadEvent = null;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao liberar recursos", ex);
            }
               }

               // Thread entry point
               public void WorkerThread()
               {
            try
            {
                byte[] buffer = new byte[bufSize];  // buffer to read stream

                while (true)
                {
                    // reset reload event
                    reloadEvent.Reset();

                    HttpWebRequest req = null;
                    WebResponse resp = null;
                    Stream stream = null;
                    byte[] delimiter = null;
                    byte[] delimiter2 = null;
                    byte[] boundary = null;
                    int boundaryLen, delimiterLen = 0, delimiter2Len = 0;
                    int read, todo = 0, total = 0, pos = 0, align = 1;
                    int start = 0, stop = 0;

                    // align
                    //  1 = searching for image start
                    //  2 = searching for image end
                    try
                    {
                        // create request
                        req = (HttpWebRequest)WebRequest.Create(source);
                        // set login and password
                        if ((login != null) && (password != null) && (login != ""))
                            req.Credentials = new NetworkCredential(login, password);
                        // set connection group name
                        if (useSeparateConnectionGroup)
                            req.ConnectionGroupName = GetHashCode().ToString();
                        // get response

                        //req.Proxy.Credentials     =     new     NetworkCredential("c1119454",
"10203033");

                        resp = req.GetResponse();

                        // check content type
                        string ct = resp.ContentType;
                        if (ct.IndexOf("multipart/x-mixed-replace") == -1)
                            throw new ApplicationException("Invalid URL");
```

```
                        // get boundary
                        ASCIIEncoding encoding = new ASCIIEncoding();
                        boundary = encoding.GetBytes(ct.Substring(ct.IndexOf("boundary=", 0) +
9));
                        boundaryLen = boundary.Length;

                        // get response stream
                        stream = resp.GetResponseStream();

                        // loop
                        while  ((!stopEvent.WaitOne(0,  true))  &&  (!reloadEvent.WaitOne(0,
true)))
                        {
                            // check total read
                            if (total > bufSize - readSize)
                            {
                                System.Diagnostics.Debug.WriteLine("flushing");
                                total = pos = todo = 0;
                            }

                            // read next portion from stream
                            if ((read = stream.Read(buffer, total, readSize)) == 0)
                                throw new ApplicationException();

                            total += read;
                            todo += read;

                            // increment received bytes counter
                            bytesReceived += read;

                            // does we know the delimiter ?
                            if (delimiter == null)
                            {
                                // find boundary
                                pos = ByteArrayUtils.Find(buffer, boundary, pos, todo);

                                if (pos == -1)
                                {
                                    // was not found
                                    todo = boundaryLen - 1;
                                    pos = total - todo;
                                    continue;
                                }

                                todo = total - pos;

                                if (todo < 2)
                                    continue;

                                // check new line delimiter type
                                if (buffer[pos + boundaryLen] == 10)
                                {
                                    delimiterLen = 2;
                                    delimiter = new byte[2] { 10, 10 };
                                    delimiter2Len = 1;
                                    delimiter2 = new byte[1] { 10 };
                                }
                                else
                                {
                                    delimiterLen = 4;
                                    delimiter = new byte[4] { 13, 10, 13, 10 };
                                    delimiter2Len = 2;
                                    delimiter2 = new byte[2] { 13, 10 };
                                }

                                pos += boundaryLen + delimiter2Len;
                                todo = total - pos;
                            }

                            // search for image
                            if (align == 1)
                            {
                                start = ByteArrayUtils.Find(buffer, delimiter, pos, todo);
                                if (start != -1)
                                {
                                    // found delimiter
                                    start += delimiterLen;
```

```
                                pos = start;
                                todo = total - pos;
                                align = 2;
                            }
                            else
                            {
                                // delimiter not found
                                todo = delimiterLen - 1;
                                pos = total - todo;
                            }
                        }

                        // search for image end
                        while ((align == 2) && (todo >= boundaryLen))
                        {
                            stop = ByteArrayUtils.Find(buffer, boundary, pos, todo);
                            if (stop != -1)
                            {
                                pos = stop;
                                todo = total - pos;

                                // increment frames counter
                                framesReceived++;

                                // image at stop
                                if (NewFrame != null)
                                {
                                    Bitmap     bmp     =     (Bitmap)Bitmap.FromStream(new
MemoryStream(buffer, start, stop - start));
                                    // notify client
                                    NewFrame(this, new CameraEventArgs(bmp));
                                    // release the image
                                    bmp.Dispose();
                                    bmp = null;
                                }
                                //
       System.Diagnostics.Debug.WriteLine("found image end, size = " + (stop - start));

                                // shift array
                                pos = stop + boundaryLen;
                                todo = total - pos;
                                Array.Copy(buffer, pos, buffer, 0, todo);

                                total = todo;
                                pos = 0;
                                align = 1;
                            }
                            else
                            {
                                // delimiter not found
                                todo = boundaryLen - 1;
                                pos = total - todo;
                            }
                        }
                    }
                }
                catch (WebException ex)
                {
                    System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
                    // wait for a while before the next try
                    Thread.Sleep(250);
                }
                catch (ApplicationException ex)
                {
                    System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
                    // wait for a while before the next try
                    Thread.Sleep(250);
                }
                catch (Exception ex)
                {
                    System.Diagnostics.Debug.WriteLine("==============: " + ex.Message);
                }
                finally
                {
                    // abort request
                    if (req != null)
                    {
```

```
                            req.Abort();
                            req = null;
                        }
                        // close response stream
                        if (stream != null)
                        {
                            stream.Close();
                            stream = null;
                        }
                        // close response
                        if (resp != null)
                        {
                            resp.Close();
                            resp = null;
                        }
                    }

                    // need to stop ?
                    if (stopEvent.WaitOne(0, true))
                        break;
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao iniciar thread de leitura de MPEGStream", ex);
            }
        }
    }
}
```

VideoFileSource.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;
        using System.IO;
        using System.Threading;

    using SentryGun.AVI;

        /// <summary>
        /// VideoFileSource
        /// </summary>
        public class VideoFileSource : IVideoSource
        {
                private string source;
                private object userData = null;
                private int           framesReceived;

                private Thread thread = null;
                private ManualResetEvent stopEvent = null;

                // new frame event
                public event CameraEventHandler NewFrame;

                // VideoSource property
                public virtual string VideoSource
                {
                        get { return source; }
                        set { source = value; }
                }
                // Login property
                public string Login
                {
                        get { return null; }
                        set { }
                }
                // Password property
                public string Password
                {
                        get { return null; }
                        set { }
                }
                // FramesReceived property
```

```
   public int FramesReceived
   {
          get
          {
                 int frames = framesReceived;
                 framesReceived = 0;
                 return frames;
          }
   }
   // BytesReceived property
   public int BytesReceived
   {
          get { return 0; }
   }
   // UserData property
   public object UserData
   {
          get { return userData; }
          set { userData = value; }
   }
   // Get state of the video source thread
   public bool Running
   {
          get
          {
    try
    {
        if (thread != null)
        {
            if (thread.Join(0) == false)
                return true;

            // the thread is not running, so free resources
            Free();
        }
        return false;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao obter propriedade", ex);
    }
          }
   }


   // Constructor
   public VideoFileSource()
   {
   }

   // Start work
   public void Start()
   {
try
{
    if (thread == null)
    {
        framesReceived = 0;

        // create events
        stopEvent = new ManualResetEvent(false);

        // create and start new thread
        thread = new Thread(new ThreadStart(WorkerThread));
        thread.Name = source;
        thread.Start();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar video source", ex);
}
   }

   // Signal thread to stop work
   public void SignalToStop()
   {
```

```
try
{
    // stop thread
    if (thread != null)
    {
        // signal to stop
        stopEvent.Set();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao enviar sinal de parada para video source", ex);
}
   }

   // Wait for thread stop
   public void WaitForStop()
   {
try
{
    if (thread != null)
    {
        // wait for thread stop
        thread.Join();

        Free();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao aguardar thread", ex);
}
   }

   // Abort thread
   public void Stop()
   {
try
{
    if (this.Running)
    {
        thread.Abort();
        WaitForStop();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao parar video source", ex);
}
   }

   // Free resources
   private void Free()
   {
try
{
    thread = null;

    // release events
    stopEvent.Close();
    stopEvent = null;
}
catch (Exception ex)
{
    throw new Exception("Erro ao liberar recursos", ex);
}
   }

   // Thread entry point
   public void WorkerThread()
   {
try
{
    AVIReader aviReader = new AVIReader();

    try
    {
```

```
                    // open file
                    aviReader.Open(source);

                    while (true)
                    {
                        // start time
                        DateTime start = DateTime.Now;

                        // get next frame
                        Bitmap bmp = aviReader.GetNextFrame();

                        framesReceived++;

                        // need to stop ?
                        if (stopEvent.WaitOne(0, false))
                            break;

                        if (NewFrame != null)
                            NewFrame(this, new CameraEventArgs(bmp));

                        // free image
                        bmp.Dispose();

                        // end time
                        TimeSpan span = DateTime.Now.Subtract(start);

                    }
                }
                catch (Exception ex)
                {
                    System.Diagnostics.Debug.WriteLine("exception : " + ex.Message);
                }

                aviReader.Dispose();
                aviReader = null;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao iniciar thread de leitura de video source", ex);
            }
               }
        }
}
```

VideoStream.cs

```
namespace SentryGun.Motion.VideoSource
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;
        using System.IO;
        using System.Threading;
        using System.Runtime.InteropServices;
        using System.Net;

        using SentryGun.Dshow;
        using SentryGun.Dshow.Core;

        /// <summary>
        /// VideoStream - stream downloader
        /// </summary>
        public class VideoStream : IVideoSource
        {
                private string source;
                private object userData = null;
                private int           framesReceived;

                private Thread thread = null;
                private ManualResetEvent stopEvent = null;

                // new frame event
                public event CameraEventHandler NewFrame;

                // VideoSource property
```

```
   public virtual string VideoSource
   {
          get { return source; }
          set { source = value; }
   }
   // Login property
   public string Login
   {
          get { return null; }
          set { }
   }
   // Password property
   public string Password
   {
          get { return null; }
          set {  }
   }
   // FramesReceived property
   public int FramesReceived
   {
          get
          {
                 int frames = framesReceived;
                 framesReceived = 0;
                 return frames;
          }
   }
   // BytesReceived property
   public int BytesReceived
   {
          get { return 0; }
   }
   // UserData property
   public object UserData
   {
          get { return userData; }
          set { userData = value; }
   }
   // Get state of the video source thread
   public bool Running
   {
get
{
    try
    {
        if (thread != null)
        {
            if (thread.Join(0) == false)
                return true;

            // the thread is not running, so free resources
            Free();
        }
        return false;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao obter propriedade", ex);
    }
}
   }


   // Constructor
   public VideoStream()
   {
   }

   // Start work
   public void Start()
   {
try
{
    if (thread == null)
    {
        framesReceived = 0;
```

```
        // create events
        stopEvent = new ManualResetEvent(false);

        // create and start new thread
        thread = new Thread(new ThreadStart(WorkerThread));
        thread.Name = source;
        thread.Start();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar VideoStream", ex);
}
   }

   // Signal thread to stop work
   public void SignalToStop()
   {
try
{
    // stop thread
    if (thread != null)
    {
        // signal to stop
        stopEvent.Set();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao enviar sinal de parada", ex);
}
   }

   // Wait for thread stop
   public void WaitForStop()
   {
try
{
    if (thread != null)
    {
        // wait for thread stop
        thread.Join();

        Free();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro aguardar thread", ex);
}
   }

   // Abort thread
   public void Stop()
   {
try
{
    if (this.Running)
    {
        thread.Abort();
        // WaitForStop();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao parar VideoStream", ex);
}
   }

   // Free resources
   private void Free()
   {
try
{
    thread = null;

    // release events
```

```
    stopEvent.Close();
    stopEvent = null;
}
catch (Exception ex)
{
    throw new Exception("Erro ao liberar recursos", ex);
}
   }

   // Thread entry point
   public void WorkerThread()
   {
try
{
    bool failed = false;

    // grabber
    Grabber grabber = new Grabber(this);

    // objects
    object graphObj = null;
    object sourceObj = null;
    object grabberObj = null;

    // interfaces
    IGraphBuilder graph = null;
    IBaseFilter sourceBase = null;
    IBaseFilter grabberBase = null;
    ISampleGrabber sg = null;
    IFileSourceFilter fileSource = null;
    IMediaControl mc = null;
    IMediaEventEx mediaEvent = null;

    int code, param1, param2;

    while ((!failed) && (!stopEvent.WaitOne(0, true)))
    {
        try
        {
            // Get type for filter graph
            Type srvType = Type.GetTypeFromCLSID(Clsid.FilterGraph);
            if (srvType == null)
                throw new ApplicationException("Failed creating filter graph");

            // create filter graph
            graphObj = Activator.CreateInstance(srvType);
            graph = (IGraphBuilder)graphObj;

            // Get type for windows media source filter
            srvType = Type.GetTypeFromCLSID(Clsid.WindowsMediaSource);
            if (srvType == null)
                throw new ApplicationException("Failed creating WM source");

            // create windows media source filter
            sourceObj = Activator.CreateInstance(srvType);
            sourceBase = (IBaseFilter)sourceObj;

            // Get type for sample grabber
            srvType = Type.GetTypeFromCLSID(Clsid.SampleGrabber);
            if (srvType == null)
                throw new ApplicationException("Failed creating sample grabber");

            // create sample grabber
            grabberObj = Activator.CreateInstance(srvType);
            sg = (ISampleGrabber)grabberObj;
            grabberBase = (IBaseFilter)grabberObj;

            // add source filter to graph
            graph.AddFilter(sourceBase, "source");
            graph.AddFilter(grabberBase, "grabber");

            // set media type
            AMMediaType mt = new AMMediaType();
            mt.majorType = MediaType.Video;
            mt.subType = MediaSubType.RGB24;
            sg.SetMediaType(mt);
```

```
                        // load file
                        fileSource = (IFileSourceFilter)sourceObj;
                        fileSource.Load(this.source, null);

                        // connect pins
                        if           (graph.Connect(DSTools.GetOutPin(sourceBase,           0),
DSTools.GetInPin(grabberBase, 0)) < 0)
                            throw new ApplicationException("Failed connecting filters");

                        // get media type
                        if (sg.GetConnectedMediaType(mt) == 0)
                        {
                            VideoInfoHeader                         vih                         =
(VideoInfoHeader)Marshal.PtrToStructure(mt.formatPtr, typeof(VideoInfoHeader));

                            grabber.Width = vih.BmiHeader.Width;
                            grabber.Height = vih.BmiHeader.Height;
                            mt.Dispose();
                        }

                        // render
                        graph.Render(DSTools.GetOutPin(grabberBase, 0));

                        //
                        sg.SetBufferSamples(false);
                        sg.SetOneShot(false);
                        sg.SetCallback(grabber, 1);

                        // window
                        IVideoWindow win = (IVideoWindow)graphObj;
                        win.put_AutoShow(false);
                        win = null;

                        // get events interface
                        mediaEvent = (IMediaEventEx)graphObj;

                        // get media control
                        mc = (IMediaControl)graphObj;

                        // run
                        mc.Run();

                        while (!stopEvent.WaitOne(0, true))
                        {
                            Thread.Sleep(100);

                            // get an event
                            if (mediaEvent.GetEvent(out code, out param1, out param2, 0) == 0)
                            {
                                // release params
                                mediaEvent.FreeEventParams(code, param1, param2);

                                //
                                if (code == (int)EventCode.Complete)
                                {
                                    System.Diagnostics.Debug.WriteLine("completed");
                                    break;
                                }
                            }
                        }

                        mc.StopWhenReady();
                    }
                    // catch any exceptions
                    catch (Exception e)
                    {
                        System.Diagnostics.Debug.WriteLine("----: " + e.Message);
                        failed = true;
                    }
                    // finalization block
                    finally
                    {
                        // release all objects
                        mediaEvent = null;
                        mc = null;
                        fileSource = null;
                        graph = null;
```

```
            sourceBase = null;
            grabberBase = null;
            sg = null;

            if (graphObj != null)
            {
                Marshal.ReleaseComObject(graphObj);
                graphObj = null;
            }
            if (sourceObj != null)
            {
                Marshal.ReleaseComObject(sourceObj);
                sourceObj = null;
            }
            if (grabberObj != null)
            {
                Marshal.ReleaseComObject(grabberObj);
                grabberObj = null;
            }
        }
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao iniciar thread de leitura de ViedoStream", ex);
}
   }

   // new frame
   protected void OnNewFrame(Bitmap image)
   {
try
{
    framesReceived++;
    if ((!stopEvent.WaitOne(0, true)) && (NewFrame != null))
        NewFrame(this, new CameraEventArgs(image));
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar evento", ex);
}
   }

   // Grabber
   private class Grabber : ISampleGrabberCB
   {
          private VideoStream parent;
          private int width, height;

          // Width property
          public int Width
          {
                 get { return width; }
                 set { width = value; }
          }
          // Height property
          public int Height
          {
                 get { return height; }
                 set { height = value; }
          }

          // Constructor
          public Grabber(VideoStream parent)
          {
                 this.parent = parent;
          }

          //
          public int SampleCB(double SampleTime, IntPtr pSample)
          {
                 return 0;
          }

          // Callback method that receives a pointer to the sample buffer
          public int BufferCB(double SampleTime, IntPtr pBuffer, int BufferLen)
          {
```

```
                try
                {
                    // create new image
                    System.Drawing.Bitmap     img     =     new     Bitmap(width,     height,
PixelFormat.Format24bppRgb);

                    // lock bitmap data
                    BitmapData bmData = img.LockBits(
                        new Rectangle(0, 0, width, height),
                        ImageLockMode.ReadWrite,
                        PixelFormat.Format24bppRgb);

                    // copy image data
                    int srcStride = bmData.Stride;
                    int dstStride = bmData.Stride;

                    int dst = bmData.Scan0.ToInt32() + dstStride * (height - 1);
                    int src = pBuffer.ToInt32();

                    for (int y = 0; y < height; y++)
                    {
                        Win32.memcpy(dst, src, srcStride);
                        dst -= dstStride;
                        src += srcStride;
                    }

                    // unlock bitmap data
                    img.UnlockBits(bmData);

                    // notify parent
                    parent.OnNewFrame(img);

                    // release the image
                    img.Dispose();

                    return 0;
                }
                catch (Exception ex)
                {
                    throw new Exception("Erro ao ler buffer da imagem", ex);
                }
                      }
                }
        }
}
```

Win32.cs

```
namespace SentryGun.AVI
{
        using System;
        using System.Runtime.InteropServices;

        /// <summary>
        /// Windows API functions and structures
        /// </summary>
        internal class Win32
        {
                // --- AVI Functions

                // Initialize the AVIFile library
                [DllImport("avifil32.dll")]
                public static extern void AVIFileInit();

                // Exit the AVIFile library
                [DllImport("avifil32.dll")]
                public static extern void AVIFileExit();

                // Open an AVI file
                [DllImport("avifil32.dll", CharSet=CharSet.Unicode)]
                public static extern int AVIFileOpen(
                        out IntPtr ppfile,
                        String szFile,
                        OpenFileMode mode,
                        IntPtr pclsidHandler);
```

```
        // Release an open AVI stream
        [DllImport("avifil32.dll")]
        public static extern int AVIFileRelease(
               IntPtr pfile);

        // Get address of a stream interface that is associated
        // with a specified AVI file
        [DllImport("avifil32.dll")]
        public static extern int AVIFileGetStream(
               IntPtr pfile,
               out IntPtr ppavi,
               int fccType,
               int lParam);

        // Create a new stream in an existing file and creates an interface to the new
stream
        [DllImport("avifil32.dll")]
        public static extern int AVIFileCreateStream(
               IntPtr pfile,
               out IntPtr ppavi,
               ref AVISTREAMINFO psi);

        // Release an open AVI stream
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamRelease(
               IntPtr pavi);

        // Set the format of a stream at the specified position
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamSetFormat(
               IntPtr pavi,
               int lPos,
               ref BITMAPINFOHEADER lpFormat,
               int cbFormat);

        // Get the starting sample number for the stream
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamStart(
               IntPtr pavi);

        // Get the length of the stream
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamLength(
               IntPtr pavi);

        // Obtain stream header information
        [DllImport("avifil32.dll", CharSet=CharSet.Unicode)]
        public static extern int AVIStreamInfo(
               IntPtr pavi,
               ref AVISTREAMINFO psi,
               int lSize);

        // Prepare to decompress video frames from the specified video stream
        [DllImport("avifil32.dll")]
        public static extern IntPtr AVIStreamGetFrameOpen(
               IntPtr pavi,
               ref BITMAPINFOHEADER lpbiWanted);
        [DllImport("avifil32.dll")]
        public static extern IntPtr AVIStreamGetFrameOpen(
               IntPtr pavi,
               int lpbiWanted);

        // Releases resources used to decompress video frames
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamGetFrameClose(
               IntPtr pget);

        // Return the address of a decompressed video frame
        [DllImport("avifil32.dll")]
        public static extern IntPtr AVIStreamGetFrame(
               IntPtr pget,
               int lPos);

        // Write data to a stream
        [DllImport("avifil32.dll")]
        public static extern int AVIStreamWrite(
               IntPtr pavi,
```

```
                        int lStart,
                        int lSamples,
                        IntPtr lpBuffer,
                        int cbBuffer,
                        int dwFlags,
                        IntPtr plSampWritten,
                        IntPtr plBytesWritten);

                // Retrieve the save options for a file and returns them in a buffer
                [DllImport("avifil32.dll")]
                public static extern int AVISaveOptions(
                        IntPtr hwnd,
                        int flags,
                        int streams,
                        [In,   MarshalAs(UnmanagedType.LPArray,   SizeParamIndex=0)]   IntPtr   []
ppavi,
                        [In,   MarshalAs(UnmanagedType.LPArray,   SizeParamIndex=0)]   IntPtr   []
plpOptions);

                // Free the resources allocated by the AVISaveOptions function
                [DllImport("avifil32.dll")]
                public static extern int AVISaveOptionsFree(
                        int streams,
                        [In,   MarshalAs(UnmanagedType.LPArray,   SizeParamIndex=0)]   IntPtr   []
plpOptions);

                // Create a compressed stream from an uncompressed stream and a
                // compression filter, and returns the address of a pointer to
                // the compressed stream
                [DllImport("avifil32.dll")]
                public static extern int AVIMakeCompressedStream(
                        out IntPtr ppsCompressed,
                        IntPtr psSource,
                        ref AVICOMPRESSOPTIONS lpOptions,
                        IntPtr pclsidHandler);

                // --- memory functions

                // memcpy - copy a block of memery
                [DllImport("ntdll.dll")]
                public static extern IntPtr memcpy(
                        IntPtr dst,
                        IntPtr src,
                        int count);
                [DllImport("ntdll.dll")]
                public static extern int memcpy(
                        int dst,
                        int src,
                        int count);

                // --- structures

                // Define the coordinates of the upper-left and
                // lower-right corners of a rectangle
                [StructLayout(LayoutKind.Sequential, Pack=1)]
                public struct RECT
                {
                        [MarshalAs(UnmanagedType.I4)] public int left;
                        [MarshalAs(UnmanagedType.I4)] public int top;
                        [MarshalAs(UnmanagedType.I4)] public int right;
                        [MarshalAs(UnmanagedType.I4)] public int bottom;
                }

                // Contains information for a single stream
                [StructLayout(LayoutKind.Sequential, CharSet=CharSet.Unicode, Pack=1)]
                public struct AVISTREAMINFO
                {
                        [MarshalAs(UnmanagedType.I4)] public int fccType;
                        [MarshalAs(UnmanagedType.I4)] public int fccHandler;
                        [MarshalAs(UnmanagedType.I4)] public int dwFlags;
                        [MarshalAs(UnmanagedType.I4)] public int dwCaps;
                        [MarshalAs(UnmanagedType.I2)] public short wPriority;
                        [MarshalAs(UnmanagedType.I2)] public short wLanguage;
                        [MarshalAs(UnmanagedType.I4)] public int dwScale;
                        [MarshalAs(UnmanagedType.I4)] public int dwRate;           //  dwRate  /
dwScale == samples/second
                        [MarshalAs(UnmanagedType.I4)] public int dwStart;
```

```
        [MarshalAs(UnmanagedType.I4)] public int dwLength;
        [MarshalAs(UnmanagedType.I4)] public int dwInitialFrames;
        [MarshalAs(UnmanagedType.I4)] public int dwSuggestedBufferSize;
        [MarshalAs(UnmanagedType.I4)] public int dwQuality;
        [MarshalAs(UnmanagedType.I4)] public int dwSampleSize;
        [MarshalAs(UnmanagedType.Struct, SizeConst=16)] public RECT rcFrame;
        [MarshalAs(UnmanagedType.I4)] public int dwEditCount;
        [MarshalAs(UnmanagedType.I4)] public int dwFormatChangeCount;
        [MarshalAs(UnmanagedType.ByValTStr, SizeConst=64)] public string szName;
}

// Contains information about the dimensions and color format of a DIB
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct BITMAPINFOHEADER
{
        [MarshalAs(UnmanagedType.I4)] public int biSize;
        [MarshalAs(UnmanagedType.I4)] public int biWidth;
        [MarshalAs(UnmanagedType.I4)] public int biHeight;
        [MarshalAs(UnmanagedType.I2)] public short biPlanes;
        [MarshalAs(UnmanagedType.I2)] public short biBitCount;
        [MarshalAs(UnmanagedType.I4)] public int biCompression;
        [MarshalAs(UnmanagedType.I4)] public int biSizeImage;
        [MarshalAs(UnmanagedType.I4)] public int biXPelsPerMeter;
        [MarshalAs(UnmanagedType.I4)] public int biYPelsPerMeter;
        [MarshalAs(UnmanagedType.I4)] public int biClrUsed;
        [MarshalAs(UnmanagedType.I4)] public int biClrImportant;
}

// Contains information about a stream and how it is compressed and saved
[StructLayout(LayoutKind.Sequential, Pack=1)]
public struct AVICOMPRESSOPTIONS
{
        [MarshalAs(UnmanagedType.I4)] public int fccType;
        [MarshalAs(UnmanagedType.I4)] public int fccHandler;
        [MarshalAs(UnmanagedType.I4)] public int dwKeyFrameEvery;
        [MarshalAs(UnmanagedType.I4)] public int dwQuality;
        [MarshalAs(UnmanagedType.I4)] public int dwBytesPerSecond;
        [MarshalAs(UnmanagedType.I4)] public int dwFlags;
        [MarshalAs(UnmanagedType.I4)] public int lpFormat;
        [MarshalAs(UnmanagedType.I4)] public int cbFormat;
        [MarshalAs(UnmanagedType.I4)] public int lpParms;
        [MarshalAs(UnmanagedType.I4)] public int cbParms;
        [MarshalAs(UnmanagedType.I4)] public int dwInterleaveEvery;
}

// --- enumerations

// File access modes
[Flags]
public enum OpenFileMode
{
        Read            = 0x00000000,
        Write                = 0x00000001,
        ReadWrite            = 0x00000002,
        ShareCompat     = 0x00000000,
        ShareExclusive = 0x00000010,
        ShareDenyWrite = 0x00000020,
        ShareDenyRead  = 0x00000030,
        ShareDenyNone  = 0x00000040,
        Parse                = 0x00000100,
        Delete               = 0x00000200,
        Verify               = 0x00000400,
        Cancel               = 0x00000800,
        Create               = 0x00001000,
        Prompt               = 0x00002000,
        Exist                = 0x00004000,
        Reopen               = 0x00008000
}

// ---

// Replacement of mmioFOURCC macros
public static int mmioFOURCC(string str)
{
        return (
                ((int)(byte)(str[0])) |
                ((int)(byte)(str[1]) << 8) |
```

```
                        ((int)(byte)(str[2]) << 16) |
                        ((int)(byte)(str[3]) << 24));
            }

            // Inverse of mmioFOURCC
            public static string decode_mmioFOURCC(int code)
            {
        try
        {
            char[] chs = new char[4];

            for (int i = 0; i < 4; i++)
            {
                chs[i] = (char)(byte)((code >> (i << 3)) & 0xFF);
                if (!char.IsLetterOrDigit(chs[i]))
                    chs[i] = ' ';
            }
            return new string(chs);
        }
        catch (Exception ex)
        {
            throw new Exception("erro ao executar método de decode", ex);
        }
            }

            // --- public methods

            // Version of AVISaveOptions for one stream only
            //
            // I don't find a way to interop AVISaveOptions more likely, so the
            // usage of original version is not easy. The version makes it more
            // usefull.
            //
            public static int AVISaveOptions(IntPtr stream, ref AVICOMPRESSOPTIONS opts,
IntPtr owner)
            {
        try
        {
            IntPtr[] streams = new IntPtr[1];
            IntPtr[] infPtrs = new IntPtr[1];
            IntPtr mem;
            int ret;

            // alloc unmanaged memory
            mem = Marshal.AllocHGlobal(Marshal.SizeOf(typeof(AVICOMPRESSOPTIONS)));

            // copy from managed structure to unmanaged memory
            Marshal.StructureToPtr(opts, mem, false);

            streams[0] = stream;
            infPtrs[0] = mem;

            // show dialog with a list of available compresors and configuration
            ret = AVISaveOptions(IntPtr.Zero, 0, 1, streams, infPtrs);

            // copy from unmanaged memory to managed structure
            opts            =            (AVICOMPRESSOPTIONS)Marshal.PtrToStructure(mem,
typeof(AVICOMPRESSOPTIONS));

            // free AVI compression options
            AVISaveOptionsFree(1, infPtrs);

            // clear it, because the information already freed by AVISaveOptionsFree
            opts.cbFormat = 0;
            opts.cbParms = 0;
            opts.lpFormat = 0;
            opts.lpParms = 0;

            // free unmanaged memory
            Marshal.FreeHGlobal(mem);

            return ret;
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao definir opções de salvamento de AVI", ex);
        }
```

```
                }
        }
}
```

Camera.cs

```
using System;
using System.Drawing;
using System.Threading;
using SentryGun.Motion.VideoSource;

namespace SentryGun.Motion
{
        public class Camera
        {
                private IVideoSource videoSource = null;
                private IMotionDetector motionDetecotor = null;
                private Bitmap lastFrame = null;

                // altura e largura da imagem
                private int width = -1, height = -1;

                // nível de alarme
                private double alarmLevel = 0.005;

        //contagem de Frames a serem rejeitados.
        //Exemplo: se m_FrameRejectCount = 5, notifica o cliente a cada 5 frames recebidos;
        private int m_FrameRejectNumber = 0;
        private int m_FrameRejectCount = 0;

        public int FrameRejectNumber
        {
            get { return m_FrameRejectNumber; }
            set { m_FrameRejectNumber = value; }
        }

                //
                public event EventHandler NewFrame;

                public Bitmap LastFrame
                {
                        get { return lastFrame; }
                }

        public int Width
                {
                        get { return width; }
                }

        public int Height
                {
                        get { return height; }
                }

        public int FramesReceived
                {
                        get { return ( videoSource == null ) ? 0 : videoSource.FramesReceived; }
                }

        public int BytesReceived
                {
                        get { return ( videoSource == null ) ? 0 : videoSource.BytesReceived; }
                }

        public bool Running
                {
                        get { return ( videoSource == null ) ? false : videoSource.Running; }
                }

        public IMotionDetector MotionDetector
                {
                        get { return motionDetecotor; }
                        set { motionDetecotor = value; }
                }
```

```
public Camera( IVideoSource source ) : this( source, null )
       {
}

public Camera( IVideoSource source, IMotionDetector detector )
       {
    try
    {
        this.videoSource = source;
        this.motionDetecotor = detector;
        videoSource.NewFrame += new CameraEventHandler(video_NewFrame);
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao instanciar objeto", ex);
    }
       }

       public void Start( )
       {
    try
    {
        if (videoSource != null)
        {
            videoSource.Start();
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao iniciar câmera", ex);
    }
}

       public void SignalToStop( )
       {
    try
    {
        if (videoSource != null)
        {
            videoSource.SignalToStop();
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao enviar sinal de parada", ex);
    }
       }

       public void WaitForStop( )
       {
    try
    {
        // lock
        Monitor.Enter(this);

        if (videoSource != null)
        {
            videoSource.WaitForStop();
        }
        // unlock
        Monitor.Exit(this);
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao aguardar thread", ex);
    }
       }

       public void Stop( )
       {
    try
    {
        // lock
        Monitor.Enter(this);

        if (videoSource != null)
```

```
    {
        videoSource.Stop();
    }
    // unlock
    Monitor.Exit(this);
}
catch (Exception ex)
{
    throw new Exception("Erro ao parar câmera", ex);
}
   }

   // Lock
   public void Lock( )
   {
try
{
    Monitor.Enter(this);
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar lock no objeto", ex);
}
   }

   // Unlock
   public void Unlock( )
   {
try
{
    Monitor.Exit(this);
}
catch (Exception ex)
{
    throw new Exception("Erro ao executar unlock no objeto", ex);
}
   }

   // evento disparado quando recebe um novo frame
   private void video_NewFrame( object sender, CameraEventArgs e )
   {
          try
          {
                 // lock
                 Monitor.Enter( this );

                 if ( lastFrame != null )
                        lastFrame.Dispose( );

                 lastFrame = (Bitmap) e.Bitmap.Clone( );

                 //aplica algoritmo de detecção
    if ( motionDetecotor != null )
                        motionDetecotor.ProcessFrame( ref lastFrame );

                 width = lastFrame.Width;
                 height = lastFrame.Height;
          }
          catch ( Exception )
          {
    //ignora exceções
          }
          finally
          {
                 // unlock
                 Monitor.Exit( this );
          }

//Controlar rejeição de frames...
m_FrameRejectCount++;//

if (m_FrameRejectCount > m_FrameRejectNumber)
{
    // notificar cliente
    if (NewFrame != null)
        NewFrame(this, new EventArgs());
```

```
                m_FrameRejectCount = 0;
            }
               }
        }
}
```

CameraWindow.cs

```
using System;
using System.Collections;
using System.ComponentModel;
using System.Drawing;
using System.Data;
using System.Windows.Forms;
using System.Threading;

namespace SentryGun.Motion
{

        public class CameraWindow : System.Windows.Forms.Control
        {
                private Camera camera = null;
                private bool autosize = false;
                private bool needSizeUpdate = false;
         private bool firstFrame = true;
                private Color rectColor = Color.Blue;

                [DefaultValue(false)]
                public bool AutoSize
                {
                        get { return autosize; }
                        set
                        {
                                autosize = value;
                                UpdatePosition( );
                        }
                }

                [Browsable(false)]
                public Camera Camera
                {
                        get { return camera; }
                        set
                        {
                 try
                 {
                     // lock
                     Monitor.Enter(this);

                     // remover eventos
                     if (camera != null)
                         camera.NewFrame -= new EventHandler(camera_NewFrame);

                     camera = value;
                     needSizeUpdate = true;
                     firstFrame = true;

                     // definir eventos
                     if (camera != null)
                         camera.NewFrame += new EventHandler(camera_NewFrame);

                     // unlock
                     Monitor.Exit(this);
                 }
                 catch (Exception ex)
                 {
                     throw new Exception("Erro ao obter propriedade", ex);
                 }
                        }
                }

                public CameraWindow( )
                {
            try
            {
                InitializeComponent();
```

```
                SetStyle(ControlStyles.AllPaintingInWmPaint | ControlStyles.DoubleBuffer |
                    ControlStyles.ResizeRedraw | ControlStyles.UserPaint, true);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar objeto", ex);
            }
               }

               #region Windows Form Designer generated code
               private void InitializeComponent()
               {
            this.SuspendLayout();
            this.ResumeLayout(false);

               }
               #endregion

               protected override void OnPaint( PaintEventArgs pe )
               {
            if ( ( needSizeUpdate ) || ( firstFrame ) )
                    {
                           UpdatePosition( );
                           needSizeUpdate = false;
                    }

                    // lock
                    Monitor.Enter( this );

                    Graphics g = pe.Graphics;
                    Rectangle rc = this.ClientRectangle;
                    Pen pen = new Pen( rectColor, 1 );

                    g.DrawRectangle( pen, rc.X, rc.Y, rc.Width - 1, rc.Height - 1 );

                    if ( camera != null )
                    {
                           try
                           {
                                  camera.Lock( );

                                  // desenhar frame
                                  if ( camera.LastFrame != null )
                                  {
                                         g.DrawImage( camera.LastFrame, rc.X + 1, rc.Y + 1,
rc.Width - 2, rc.Height - 2 );
                                         firstFrame = false;
                                  }
                                  else
                                  {
                                         Font drawFont = new Font( "Arial", 12 );
                                         SolidBrush drawBrush = new SolidBrush( Color.White
);

                                         g.DrawString(    "Conectando    ...",    drawFont,
drawBrush, new PointF( 5, 5 ) );

                                         drawBrush.Dispose( );
                                         drawFont.Dispose( );
                                  }
                           }
                           catch ( Exception )
                           {
                           }
                           finally
                           {
                                  camera.Unlock( );
                           }
                    }

                    pen.Dispose( );

                    // unlock
                    Monitor.Exit( this );

                    base.OnPaint( pe );
```

```
            }

            public void UpdatePosition( )
            {
         try
         {
             // lock
             Monitor.Enter(this);

             if ((autosize) && (this.Parent != null))
             {
                 Rectangle rc = this.Parent.ClientRectangle;
                 int width = 320;
                 int height = 240;

                 if (camera != null)
                 {
                     camera.Lock();

                     if (camera.LastFrame != null)
                     {
                         width = camera.LastFrame.Width;
                         height = camera.LastFrame.Height;
                     }
                     camera.Unlock();
                 }

                 this.SuspendLayout();
                 this.Size = new Size(width + 2, height + 2);
                 this.ResumeLayout();

             }
             // unlock
             Monitor.Exit(this);
         }
         catch (Exception ex)
         {
             throw new Exception("Erro ao atualizar posição do controle", ex);
         }
            }

            // evento
            private void camera_NewFrame( object sender, System.EventArgs e )
            {
                   Invalidate();
            }
        }
}
```

IMotionDetector.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;

        public interface IMotionDetector
        {
                bool MotionLevelCalculation{ set; get; }
                double MotionLevel{ get; }
         int MinimumVolume { get; set; }
                void ProcessFrame( ref Bitmap image );
         Rectangle[] Rectangles { get; set; }
             void Reset( );
        }
}
```

MotionDetector1.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;
```

```
using AForge.Imaging;
using AForge.Imaging.Filters;

public class MotionDetector1 : IMotionDetector
{
       private IFilter       grayscaleFilter = new GrayscaleBT709( );
       private Difference differenceFilter = new Difference( );
       private IFilter thresholdFilter = new Threshold( 15 );
       private IFilter erosionFilter = new Erosion( );
       private Merge mergeFilter = new Merge( );
       private IFilter extrachChannel = new ExtractChannel( RGB.R );
       private FiltersSequence      processingFilter = new FiltersSequence( );
       private Bitmap backgroundFrame;
       private bool calculateMotionLevel = false;
       private int width;
       private int height;
       private int pixelsChanged;
private int m_MinimumVolume;
private Rectangle[] m_rects;

public Rectangle[] Rectangles
{
    get { return m_rects; }
    set { m_rects = value; }
}

public int MinimumVolume
{
    get { return m_MinimumVolume; }
    set { m_MinimumVolume = value; }
}

// Calcula ou não o nível de movimento
       public bool MotionLevelCalculation
       {
              get { return calculateMotionLevel; }
              set { calculateMotionLevel = value; }
       }

       // Quantidade de alterações (percentual)
       public double MotionLevel
       {
              get
    {
        try
        {
            return (double)pixelsChanged / (width * height);
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao obter propriedade", ex);
        }
    }
       }


       public MotionDetector1( )
       {
    try
    {
        processingFilter.Add(differenceFilter);
        processingFilter.Add(thresholdFilter);
        processingFilter.Add(erosionFilter);
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao instanciar objeto", ex);
    }
       }

       public void Reset( )
       {
    try
    {
        if (backgroundFrame != null)
        {
            backgroundFrame.Dispose();
```

```
            backgroundFrame = null;
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao resetar detector", ex);
    }
       }

       public void ProcessFrame( ref Bitmap image )
       {
    try
    {
        if (backgroundFrame == null)
        {
            // criando background
            backgroundFrame = grayscaleFilter.Apply(image);

            width = image.Width;
            height = image.Height;

            return;
        }

        Bitmap tmpImage;

        //aplicando escala de cinza
        tmpImage = grayscaleFilter.Apply(image);

        //definindo overlay para o filtro, usando o fundo...
        differenceFilter.OverlayImage = backgroundFrame;

        Bitmap tmpImage2 = processingFilter.Apply(tmpImage);

        // calcula a quantidade de pixels alterados
        pixelsChanged = (calculateMotionLevel) ?
            CalculateWhitePixels(tmpImage2) : 0;

        backgroundFrame.Dispose();
        backgroundFrame = tmpImage;

        // extrai o canal da cor vermelha
        Bitmap redChannel = extrachChannel.Apply(image);

        // faz o merge do canal vermelho extraido com o objeto em movimento
        mergeFilter.OverlayImage = tmpImage2;
        Bitmap tmpImage3 = mergeFilter.Apply(redChannel);
        redChannel.Dispose();
        tmpImage2.Dispose();

        // redefine o canal do vermelho
        ReplaceChannel replaceChannel = new ReplaceChannel(RGB.R, tmpImage3);
        Bitmap tmpImage4 = replaceChannel.Apply(image);
        tmpImage3.Dispose();

        image.Dispose();
        image = tmpImage4;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao processar frame", ex);
    }
       }

       private int CalculateWhitePixels( Bitmap image )
       {
    try
    {
        int count = 0;

        BitmapData data = image.LockBits(new Rectangle(0, 0, width, height),
            ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);

        int offset = data.Stride - width;

        //perigo
        unsafe
```

```
                {
                    byte* ptr = (byte*)data.Scan0.ToPointer();

                    for (int y = 0; y < height; y++)
                    {
                        for (int x = 0; x < width; x++, ptr++)
                        {
                            count += ((*ptr) >> 7);
                        }
                        ptr += offset;
                    }
                }

                image.UnlockBits(data);

                return count;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao calcular pixels brancos", ex);
            }
               }
        }
}
```

MotionDetector2.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;

        using AForge.Imaging;
        using AForge.Imaging.Filters;

        public class MotionDetector2 : IMotionDetector
        {
                private IFilter       grayscaleFilter = new GrayscaleBT709( );
                private Difference differenceFilter = new Difference( );
                private IFilter thresholdFilter = new Threshold( 15 );
                private IFilter openingFilter = new Opening( );
                private IFilter edgesFilter = new Edges( );
                private Merge mergeFilter = new Merge( );
                private IFilter extrachChannel = new ExtractChannel( RGB.R );
                private MoveTowards moveTowardsFilter = new MoveTowards( );
                private FiltersSequence      processingFilter1 = new FiltersSequence( );
                private FiltersSequence      processingFilter2 = new FiltersSequence( );
                private Bitmap backgroundFrame;
                private int counter = 0;
                private bool calculateMotionLevel = false;
                private int width;
                private int height;
                private int pixelsChanged;
         private int m_MinimumVolume;
         private Rectangle[] m_rects;

         public Rectangle[] Rectangles
         {
             get { return m_rects; }
             set { m_rects = value; }
         }

         public int MinimumVolume
         {
             get { return m_MinimumVolume; }
             set { m_MinimumVolume = value; }
         }

         // Calcula ou não o nível de movimento
         public bool MotionLevelCalculation
                {
                        get { return calculateMotionLevel; }
                        set { calculateMotionLevel = value; }
                }
```

```
// Quantidade de alterações (percentual)
public double MotionLevel
       {
    get
    {
        try
        {
            return (double)pixelsChanged / (width * height);
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao obter propriedade", ex);
        }
    }
       }

       public MotionDetector2( )
       {
    try
    {
        processingFilter1.Add(differenceFilter);
        processingFilter1.Add(thresholdFilter);
        processingFilter2.Add(openingFilter);
        processingFilter2.Add(edgesFilter);
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao instanciar objeto", ex);
    }
       }

       public void Reset( )
       {
    try
    {
        if (backgroundFrame != null)
        {
            backgroundFrame.Dispose();
            backgroundFrame = null;
        }
        counter = 0;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao resetar detector", ex);
    }
       }

       public void ProcessFrame( ref Bitmap image )
       {
    try
    {
        if (backgroundFrame == null)
        {
            backgroundFrame = grayscaleFilter.Apply(image);

            width = image.Width;
            height = image.Height;

            return;
        }

        Bitmap tmpImage;

        //aplicando escala de cinza
        tmpImage = grayscaleFilter.Apply(image);


        if (++counter == 2)
        {
            counter = 0;

            // move o fundo na direção do novo frame
            moveTowardsFilter.OverlayImage = tmpImage;
            Bitmap tmp = moveTowardsFilter.Apply(backgroundFrame);

            backgroundFrame.Dispose();
```

```
                    backgroundFrame = tmp;
                }

                // set backgroud frame as an overlay for difference filter
                differenceFilter.OverlayImage = backgroundFrame;

                Bitmap tmpImage2 = processingFilter1.Apply(tmpImage);
                tmpImage.Dispose();

                pixelsChanged = (calculateMotionLevel) ?
                    CalculateWhitePixels(tmpImage2) : 0;

                Bitmap tmpImage2b = processingFilter2.Apply(tmpImage2);
                tmpImage2.Dispose();

                Bitmap redChannel = extrachChannel.Apply(image);

                mergeFilter.OverlayImage = tmpImage2b;
                Bitmap tmpImage3 = mergeFilter.Apply(redChannel);
                redChannel.Dispose();
                tmpImage2b.Dispose();

                ReplaceChannel replaceChannel = new ReplaceChannel(RGB.R, tmpImage3);
                Bitmap tmpImage4 = replaceChannel.Apply(image);
                tmpImage3.Dispose();

                image.Dispose();
                image = tmpImage4;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao processar frame", ex);
            }
               }

               private int CalculateWhitePixels( Bitmap image )
               {
            try
            {
                int count = 0;

                BitmapData data = image.LockBits(new Rectangle(0, 0, width, height),
                    ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);

                int offset = data.Stride - width;


                //perigo
                unsafe
                {
                    byte* ptr = (byte*)data.Scan0.ToPointer();

                    for (int y = 0; y < height; y++)
                    {
                        for (int x = 0; x < width; x++, ptr++)
                        {
                            count += ((*ptr) >> 7);
                        }
                        ptr += offset;
                    }
                }
                image.UnlockBits(data);

                return count;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao calcular pixels brancos", ex);
            }
               }
        }
}
```

MotionDetector3.cs

```
namespace SentryGun.Motion
```

```
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;

        using AForge.Imaging;
        using AForge.Imaging.Filters;

        public class MotionDetector3 : IMotionDetector
        {
               private IFilter       grayscaleFilter = new GrayscaleBT709( );
               private IFilter       pixellateFilter = new Pixellate( );
               private Difference differenceFilter = new Difference( );
               private IFilter thresholdFilter = new Threshold( 15 );
               private Dilatation dilatationFilter = new Dilatation( );
               private IFilter edgesFilter = new Edges( );
               private Merge mergeFilter = new Merge( );
               private IFilter extrachChannel = new ExtractChannel( RGB.R );
         private MoveTowards moveTowardsFilter = new MoveTowards( );
               private FiltersSequence      processingFilter1 = new FiltersSequence( );
               private FiltersSequence      processingFilter2 = new FiltersSequence( );
               private Bitmap backgroundFrame;
               private int counter = 0;
               private bool calculateMotionLevel = false;
               private int width;
               private int height;
               private int pixelsChanged;
         private int m_MinimumVolume;
         private Rectangle[] m_rects;

         public Rectangle[] Rectangles
         {
             get { return m_rects; }
             set { m_rects = value; }
         }

         public int MinimumVolume
         {
             get { return m_MinimumVolume; }
             set { m_MinimumVolume = value; }
         }

               public bool MotionLevelCalculation
               {
                      get { return calculateMotionLevel; }
                      set { calculateMotionLevel = value; }
               }

               public double MotionLevel
               {
            get
            {
                try
                {
                    return (double)pixelsChanged / (width * height);
                }
                catch (Exception ex)
                {
                    throw new Exception("Erro ao obter propriedade", ex);
                }
            }
               }

               public MotionDetector3( )
               {
            try
            {
                processingFilter1.Add(grayscaleFilter);
                processingFilter1.Add(pixellateFilter);

                processingFilter2.Add(differenceFilter);
                processingFilter2.Add(thresholdFilter);
                processingFilter2.Add(dilatationFilter);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar objeto", ex);
```

```
}
   }

   public void Reset( )
   {
try
{
    if (backgroundFrame != null)
    {
        backgroundFrame.Dispose();
        backgroundFrame = null;
    }
    counter = 0;
}
catch (Exception ex)
{
    throw new Exception("Erro ao resetar detector", ex);
}
   }

   public void ProcessFrame( ref Bitmap image )
   {
try
{
    if (backgroundFrame == null)
    {
        backgroundFrame = processingFilter1.Apply(image);

        width = image.Width;
        height = image.Height;

        return;
    }

    Bitmap tmpImage;

    tmpImage = processingFilter1.Apply(image);

    if (++counter == 2)
    {
        counter = 0;

        moveTowardsFilter.OverlayImage = tmpImage;
        Bitmap tmp = moveTowardsFilter.Apply(backgroundFrame);

        backgroundFrame.Dispose();
        backgroundFrame = tmp;
    }

    differenceFilter.OverlayImage = backgroundFrame;

    Bitmap tmpImage2 = processingFilter2.Apply(tmpImage);
    tmpImage.Dispose();

    pixelsChanged = (calculateMotionLevel) ?
        CalculateWhitePixels(tmpImage2) : 0;

    // encontrar limites
    Bitmap tmpImage2b = edgesFilter.Apply(tmpImage2);
    tmpImage2.Dispose();

    Bitmap redChannel = extrachChannel.Apply(image);

    mergeFilter.OverlayImage = tmpImage2b;
    Bitmap tmpImage3 = mergeFilter.Apply(redChannel);
    redChannel.Dispose();
    tmpImage2b.Dispose();

    ReplaceChannel replaceChannel = new ReplaceChannel(RGB.R, tmpImage3);
    Bitmap tmpImage4 = replaceChannel.Apply(image);
    tmpImage3.Dispose();

    image.Dispose();
    image = tmpImage4;
}
catch (Exception ex)
{
```

```
                throw new Exception("Erro ao processar frame", ex);
            }
               }

               private int CalculateWhitePixels( Bitmap image )
               {
            try
            {
                int count = 0;

                BitmapData data = image.LockBits(new Rectangle(0, 0, width, height),
                    ImageLockMode.ReadOnly, PixelFormat.Format8bppIndexed);

                int offset = data.Stride - width;

                //perigo
                unsafe
                {
                    byte* ptr = (byte*)data.Scan0.ToPointer();

                    for (int y = 0; y < height; y++)
                    {
                        for (int x = 0; x < width; x++, ptr++)
                        {
                            count += ((*ptr) >> 7);
                        }
                        ptr += offset;
                    }
                }

                image.UnlockBits(data);

                return count;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao calcuar pixels brancos", ex);
            }
               }
        }
}
```

MotionDetector3Optimized.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;
        using System.Drawing.Imaging;

        using AForge.Imaging;
        using AForge.Imaging.Filters;

        public class MotionDetector3Optimized : IMotionDetector
        {
               private byte[] backgroundFrame = null;
               private byte[] currentFrame = null;
               private byte[] currentFrameDilatated = null;
               private int counter = 0;
               private bool calculateMotionLevel = false;
               private int width;
               private int height;
               private int pixelsChanged;
         private int m_MinimumVolume;
         private Rectangle[] m_rects;

         public Rectangle[] Rectangles
         {
             get { return m_rects; }
             set { m_rects = value; }
         }

         public int MinimumVolume
         {
             get { return m_MinimumVolume; }
```

```
    set { m_MinimumVolume = value; }
}

       public bool MotionLevelCalculation
       {
              get { return calculateMotionLevel; }
              set { calculateMotionLevel = value; }
       }

       public double MotionLevel
       {
              get
    {
     try
     {
         return (double)pixelsChanged / (width * height);
     }
     catch (Exception ex)
     {
         throw new Exception("Erro ao obter propriedade", ex);
     }
    }
       }

       public MotionDetector3Optimized( )
       {
       }

       public void Reset( )
       {
    try
    {
     backgroundFrame = null;
     currentFrame = null;
     currentFrameDilatated = null;
     counter = 0;
    }
    catch (Exception ex)
    {
     throw new Exception("Erro ao resetar detector", ex);
    }
       }

       public void ProcessFrame( ref Bitmap image )
       {
    try
    {
     width = image.Width;
     height = image.Height;

     int fW = (((width - 1) / 8) + 1);
     int fH = (((height - 1) / 8) + 1);
     int len = fW * fH;

     if (backgroundFrame == null)
     {
         backgroundFrame = new byte[len];
         currentFrame = new byte[len];
         currentFrameDilatated = new byte[len];

         BitmapData imgData = image.LockBits(
             new Rectangle(0, 0, width, height),
             ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

         // cria fundo inicial
         PreprocessInputImage(imgData, width, height, backgroundFrame);

         image.UnlockBits(imgData);

         return;
     }

     BitmapData data = image.LockBits(
         new Rectangle(0, 0, width, height),
         ImageLockMode.ReadOnly, PixelFormat.Format24bppRgb);

     //pré-processamento da imagem (mais rápido!!!)
```

```
PreprocessInputImage(data, width, height, currentFrame);

if (++counter == 2)
{
    counter = 0;

    // move o fundo pra o frame novo (na mão dessa vez)
    for (int i = 0; i < len; i++)
    {
        int t = currentFrame[i] - backgroundFrame[i];
        if (t > 0)
            backgroundFrame[i]++;
        else if (t < 0)
            backgroundFrame[i]--;
    }
}

// diferença e limiar
pixelsChanged = 0;
for (int i = 0; i < len; i++)
{
    int t = currentFrame[i] - backgroundFrame[i];
    if (t < 0)
        t = -t;

    if (t >= 15)
    {
        pixelsChanged++;
        currentFrame[i] = (byte)255;
    }
    else
    {
        currentFrame[i] = (byte)0;
    }
}
if (calculateMotionLevel)
    pixelsChanged *= 64;
else
    pixelsChanged = 0;

// dilatação igual para extender as bordas
for (int i = 0; i < fH; i++)
{
    for (int j = 0; j < fW; j++)
    {
        int k = i * fW + j;
        int v = currentFrame[k];

        // pixels da esquerda
        if (j > 0)
        {
            v += currentFrame[k - 1];

            if (i > 0)
            {
                v += currentFrame[k - fW - 1];
            }
            if (i < fH - 1)
            {
                v += currentFrame[k + fW - 1];
            }
        }
        // pixels da direita
        if (j < fW - 1)
        {
            v += currentFrame[k + 1];

            if (i > 0)
            {
                v += currentFrame[k - fW + 1];
            }
            if (i < fH - 1)
            {
                v += currentFrame[k + fW + 1];
            }
        }
        // pixels do topo
```

```
                        if (i > 0)
                        {
                            v += currentFrame[k - fW];
                        }
                        // pixels da base
                        if (i < fH - 1)
                        {
                            v += currentFrame[k + fW];
                        }

                        currentFrameDilatated[k] = (v != 0) ? (byte)255 : (byte)0;
                    }
                }

                // pós processamento
                PostprocessInputImage(data, width, height, currentFrameDilatated);

                image.UnlockBits(data);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao processar frame", ex);
            }
               }

               private void PreprocessInputImage( BitmapData data, int width, int height,
byte[] buf )
               {
            try
            {
                int stride = data.Stride;
                int offset = stride - width * 3;
                int len = (int)((width - 1) / 8) + 1;
                int rem = ((width - 1) % 8) + 1;
                int[] tmp = new int[len];
                int i, j, t1, t2, k = 0;

                unsafe
                {
                    byte* src = (byte*)data.Scan0.ToPointer();

                    for (int y = 0; y < height; )
                    {
                        // pegar pixels
                        Array.Clear(tmp, 0, len);

                        // calcular brilhos
                        for (i = 0; (i < 8) && (y < height); i++, y++)
                        {
                            // para cada pixel
                            for (int x = 0; x < width; x++, src += 3)
                            {
                                //aplica escala de cinza BT709
                                tmp[(int)(x / 8)] += (int)(0.2125f * src[RGB.R] + 0.7154f *
src[RGB.G] + 0.0721f * src[RGB.B]);
                            }
                            src += offset;
                        }

                        // calcula a média
                        t1 = i * 8;
                        t2 = i * rem;

                        for (j = 0; j < len - 1; j++, k++)
                            buf[k] = (byte)(tmp[j] / t1);
                        buf[k++] = (byte)(tmp[j] / t2);
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao pré-processar imagem de entrada", ex);
            }
               }

               private void PostprocessInputImage( BitmapData data, int width, int height,
byte[] buf )
```

```
            {
            try
            {
                int stride = data.Stride;
                int offset = stride - width * 3;
                int len = (int)((width - 1) / 8) + 1;
                int lenWM1 = len - 1;
                int lenHM1 = (int)((height - 1) / 8);
                int rem = ((width - 1) % 8) + 1;

                int i, j, k;

                unsafe
                {
                    byte* src = (byte*)data.Scan0.ToPointer();

                    // para cada linha
                    for (int y = 0; y < height; y++)
                    {
                        i = (y / 8);

                        // para cada pixel
                        for (int x = 0; x < width; x++, src += 3)
                        {
                            j = x / 8;
                            k = i * len + j;

                            // verifica se é necessário destacar o objeto em movimento
                            if (buf[k] == 255)
                            {
                                // verificar bordas
                                if (
                                    ((x % 8 == 0) && ((j == 0) || (buf[k - 1] == 0))) ||
                                    ((x % 8 == 7) && ((j == lenWM1) || (buf[k + 1] == 0))) ||
                                    ((y % 8 == 0) && ((i == 0) || (buf[k - len] == 0))) ||
                                    ((y % 8 == 7) && ((i == lenHM1) || (buf[k + len] == 0)))
                                    )
                                {
                                    src[RGB.R] = 255;
                                }
                            }
                        }
                        src += offset;
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao pós-processar imagem de entrada", ex);
            }
               }
        }
}
```

MotionDetector4.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;
        using System.Reflection;

        using AForge.Imaging;
        using AForge.Imaging.Filters;

        public class MotionDetector4 : IMotionDetector
        {
                private IFilter grayscaleFilter = new GrayscaleBT709( );
                private IFilter pixellateFilter = new Pixellate( );
                private Difference differenceFilter = new Difference( );
                private IFilter thresholdFilter = new Threshold( 15 );
                private MoveTowards moveTowardsFilter = new MoveTowards( );
                private FiltersSequence processingFilter1 = new FiltersSequence( );
                private FiltersSequence processingFilter2 = new FiltersSequence( );
                private Bitmap backgroundFrame;
                private int counter = 0;
```

```
       private Bitmap[] numbersBitmaps = new Bitmap[9];
       private bool calculateMotionLevel = false;
       private int width;
       private int height;
       private int pixelsChanged;
private int m_MinimumVolume;
private Rectangle[] m_rects;

public Rectangle[] Rectangles
{
    get { return m_rects; }
    set { m_rects = value; }
}

public int MinimumVolume
{
    get { return m_MinimumVolume; }
    set { m_MinimumVolume = value; }
}

       public bool MotionLevelCalculation
       {
              get { return calculateMotionLevel; }
              set { calculateMotionLevel = value; }
       }

       public double MotionLevel
       {
    get
    {
        try
        {
            return (double)pixelsChanged / (width * height);
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao obter propriedade", ex);
        }
    }
       }

       public MotionDetector4( )
       {
    try
    {
        processingFilter1.Add(grayscaleFilter);
        processingFilter1.Add(pixellateFilter);

        processingFilter2.Add(differenceFilter);
        processingFilter2.Add(thresholdFilter);

        // carrega os bmp's com os números para colocar no canto dos alvos
        Assembly assembly = this.GetType().Assembly;

        for (int i = 1; i <= 9; i++)
        {
            numbersBitmaps[i - 1] = new Bitmap(assembly.GetManifestResourceStream(
                string.Format("SentryGun.Motion.Resources.{0}.gif", i)));
        }
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao instanciar objeto", ex);
    }
       }

       public void Reset( )
       {
    try
    {
        if (backgroundFrame != null)
        {
            backgroundFrame.Dispose();
            backgroundFrame = null;
        }
        counter = 0;
    }
```

```
catch (Exception ex)
{
    throw new Exception("Erro ao resetar detector", ex);
}
   }

   public void ProcessFrame( ref Bitmap image )
   {
try
{
    if (backgroundFrame == null)
    {
        backgroundFrame = processingFilter1.Apply(image);

        width = image.Width;
        height = image.Height;

        return;
    }

    Bitmap tmpImage;

    tmpImage = processingFilter1.Apply(image);

    if (++counter == 2)
    {
        counter = 0;

        moveTowardsFilter.OverlayImage = tmpImage;
        Bitmap tmp = moveTowardsFilter.Apply(backgroundFrame);

        backgroundFrame.Dispose();
        backgroundFrame = tmp;
    }

    differenceFilter.OverlayImage = backgroundFrame;

    Bitmap tmpImage2 = processingFilter2.Apply(tmpImage);
    tmpImage.Dispose();

    // obter os retângulos
    BlobCounter blobCounter = new BlobCounter(tmpImage2);

    Rectangle[] rects = blobCounter.GetObjectRectangles();
    tmpImage2.Dispose();

    pixelsChanged = 0;

    if (rects.Length != 0)
    {
        // cria os desenhos dos alvos
        Graphics g = Graphics.FromImage(image);

        using (Pen pen = new Pen(Color.Red, 1))
        {
            int n = 0;

            foreach (Rectangle rc in rects)
            {
                g.DrawRectangle(pen, rc);

                if ((n < 10) && (rc.Width > 15) && (rc.Height > 15))
                {
                    g.DrawImage(numbersBitmaps[n], rc.Left, rc.Top, 7, 9);
                    n++;
                }

                if (calculateMotionLevel)
                    pixelsChanged += rc.Width * rc.Height;
            }
        }
        g.Dispose();
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao processar frame", ex);
```

```
            }
               }
        }
}
```

MotionDetector5.cs

```
namespace SentryGun.Motion
{
        using System;
        using System.Drawing;
        using System.Reflection;

        using AForge.Imaging;
        using AForge.Imaging.Filters;

        public class MotionDetector5 : IMotionDetector
        {
               private IFilter       grayscaleFilter = new GrayscaleBT709( );
               private IFilter       pixellateFilter = new Pixellate( );
               private Difference differenceFilter = new Difference( );
               private IFilter thresholdFilter = new Threshold( 15 );
               private MoveTowards moveTowardsFilter = new MoveTowards( );
         private FiltersSequence      processingFilter1 = new FiltersSequence( );
               private FiltersSequence      processingFilter2 = new FiltersSequence( );
         private Bitmap backgroundFrame;
               private int    counter = 0;
         private Bitmap[] numbersBitmaps = new Bitmap[9];
         private bool calculateMotionLevel = false;
               private int width;
               private int height;
               private int pixelsChanged;
         private int m_MinimumVolume;
         private Rectangle[] m_rects;

         public Rectangle[] Rectangles
         {
             get { return m_rects; }
             set { m_rects = value; }
         }

         public int MinimumVolume
         {
             get { return m_MinimumVolume; }
             set { m_MinimumVolume = value; }
         }

               public bool MotionLevelCalculation
               {
                      get { return calculateMotionLevel; }
                      set { calculateMotionLevel = value; }
               }

               public double MotionLevel
               {
             get
             {
                 try
                 {
                     return (double)pixelsChanged / (width * height);
                 }
                 catch (Exception ex)
                 {
                     throw new Exception("Erro ao obter propriedade", ex);
                 }
             }
               }

               public MotionDetector5( )
               {
             try
             {
                 processingFilter1.Add(grayscaleFilter);
                 processingFilter1.Add(pixellateFilter);
```

```
    processingFilter2.Add(differenceFilter);
    processingFilter2.Add(thresholdFilter);

    Assembly assembly = this.GetType().Assembly;

    for (int i = 1; i <= 9; i++)
    {
        numbersBitmaps[i - 1] = new Bitmap(assembly.GetManifestResourceStream(
            string.Format("SentryGun.Motion.Resources.{0}.gif", i)));
    }
}
catch (Exception ex)
{
    throw new Exception("Erro ao instanciar objeto", ex);
}
   }

   public void Reset( )
   {
try
{
    if (backgroundFrame != null)
    {
        backgroundFrame.Dispose();
        backgroundFrame = null;
    }
    counter = 0;
}
catch (Exception ex)
{
    throw new Exception("Erro ao resetar detector", ex);
}
   }

   public void ProcessFrame( ref Bitmap image )
   {
try
{
    if (backgroundFrame == null)
    {
        backgroundFrame = processingFilter1.Apply(image);

        width = image.Width;
        height = image.Height;

        return;
    }

    Bitmap tmpImage;

    tmpImage = processingFilter1.Apply(image);

    //if (++counter == 2)
    //{
    //    counter = 0;

    //    moveTowardsFilter.OverlayImage = tmpImage;
    //    Bitmap tmp = moveTowardsFilter.Apply(backgroundFrame);

    //    backgroundFrame.Dispose();
    //    backgroundFrame = tmp;
    //}

    differenceFilter.OverlayImage = backgroundFrame;

    Bitmap tmpImage2 = processingFilter2.Apply(tmpImage);
    tmpImage.Dispose();

    BlobCounter counter = new BlobCounter(tmpImage2);

    m_rects = counter.GetObjectRectangles();
    tmpImage2.Dispose();

    pixelsChanged = 0;

    if (m_rects.Length != 0)
```

```
                {
                    Graphics g = Graphics.FromImage(image);

                    using (Pen pen = new Pen(Color.Red, 1))
                    {
                        int n = 0;

                        foreach (Rectangle rc in m_rects)
                        {
                            //verifica se o tamanho do alvo está dentro da tolerância
                            if ((rc.Width * rc.Height) > m_MinimumVolume)
                            {
                                g.DrawRectangle(pen, rc);

                                if ((n < 9))
                                {
                                    g.DrawImage(numbersBitmaps[n], rc.Left, rc.Top, 7, 9);
                                    n++;
                                }

                                if (calculateMotionLevel)
                                    pixelsChanged += rc.Width * rc.Height;
                            }
                        }
                    }
                    g.Dispose();
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao processar frame", ex);
            }
               }
        }
}
```

Pacote SentryGun.SentryControl

Configuration.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SentryGun.SentryControl
{
    public static class Configuration
    {
        private static int m_ServoInitialPosition = 300;
        private static int m_ServoFinalPosition = 1200;
        private static int m_CamWidth = 320;
        private static int m_CamHeight = 240;
        private static string m_BoardMessageStartOk = "Aguardando Posicao Inicial";
        private static int m_BoardCommunicationSpeed = 10;
        private static int m_BoardRepeatNumber = 50;
        private static int m_MovementPercentRejection = 3;
        private static int m_ServoXCaliberLeft = 650;
        private static int m_ServoXCaliberRight = 850;
        private static int m_ServoYCaliberTop = 870;
        private static int m_ServoYCaliberBottom = 720;
        private static int m_MinimumVolume = 0;

        private static EnumServo m_servoX = EnumServo.Servo1;
        private static EnumServo m_servoY = EnumServo.Servo2;
        private static EnumServo m_servoTrigger = EnumServo.Servo3;

        private static int m_TriggerReleasedPosition = 300;
        private static int m_TriggerPressedPosition = 400;

        private static int m_CameraFrameRejectCount = 28;

        public static int TriggerReleasedPosition
        {
```

```
    get { return Configuration.m_TriggerReleasedPosition; }
    set { Configuration.m_TriggerReleasedPosition = value; }
}

public static int TriggerPressedPosition
{
    get { return Configuration.m_TriggerPressedPosition; }
    set { Configuration.m_TriggerPressedPosition = value; }
}

public static int MovementPercentRejection
{
    get { return Configuration.m_MovementPercentRejection; }
    set { Configuration.m_MovementPercentRejection = value; }
}

public static int ServoXCaliberLeft
{
    get { return Configuration.m_ServoXCaliberLeft; }
    set { Configuration.m_ServoXCaliberLeft = value; }
}
public static int ServoXCaliberRight
{
    get { return Configuration.m_ServoXCaliberRight; }
    set { Configuration.m_ServoXCaliberRight = value; }
}
public static int MinimunVolume
{
    get { return Configuration.m_MinimumVolume; }
    set { Configuration.m_MinimumVolume = value; }
}
public static int ServoYCaliberTop
{
    get { return Configuration.m_ServoYCaliberTop; }
    set { Configuration.m_ServoYCaliberTop = value; }
}
public static int ServoYCaliberBottom
{
    get { return Configuration.m_ServoYCaliberBottom; }
    set { Configuration.m_ServoYCaliberBottom = value; }
}

public static EnumServo ServoX
{
    get { return Configuration.m_servoX; }
    set { Configuration.m_servoX = value; }
}

public static EnumServo ServoY
{
    get { return Configuration.m_servoY; }
    set { Configuration.m_servoY = value; }
}

public static EnumServo ServoTrigger
{
    get { return Configuration.m_servoTrigger; }
    set { Configuration.m_servoTrigger = value; }
}

public static int ServoInitialPosition
{
    get { return Configuration.m_ServoInitialPosition; }
    set { Configuration.m_ServoInitialPosition = value; }
}

public static int ServoFinalPosition
{
    get { return Configuration.m_ServoFinalPosition; }
    set { Configuration.m_ServoFinalPosition = value; }
}

public static int CamWidth
{
    get { return Configuration.m_CamWidth; }
    set { Configuration.m_CamWidth = value; }
}
```

```
        public static int CamHeight
        {
            get { return Configuration.m_CamHeight; }
            set { Configuration.m_CamHeight = value; }
        }

        public static string BoardMessageStartOk
        {
            get { return Configuration.m_BoardMessageStartOk; }
            set { Configuration.m_BoardMessageStartOk = value; }
        }

        public static int BoardCommunicationSpeed
        {
            get { return Configuration.m_BoardCommunicationSpeed; }
            set { Configuration.m_BoardCommunicationSpeed = value; }
        }

        public static int BoardRepeatNumber
        {
            get { return Configuration.m_BoardRepeatNumber; }
            set { Configuration.m_BoardRepeatNumber = value; }
        }

        public static int GetRelativeServoX(int CameraX)
        {
            double CamFactor = (Convert.ToDouble(CameraX) / Convert.ToDouble(m_CamWidth)) + 1;
            int ServoSize = m_ServoXCaliberRight - m_ServoXCaliberLeft;

            double increment = (ServoSize * CamFactor) - ServoSize;
            int result = Convert.ToInt32(m_ServoXCaliberLeft + increment);

            return result;
        }

        public static int GetRelativeServoY(int CameraY)
        {

            double CamFactor = (Convert.ToDouble(CameraY) / Convert.ToDouble(m_CamHeight)) +
1;
            int ServoSize = m_ServoYCaliberBottom - m_ServoYCaliberTop;

            double increment = (ServoSize * CamFactor) - ServoSize;
            int result = Convert.ToInt32(m_ServoYCaliberTop + increment);

            return result;
        }

    }
}
```

LockFreeStack.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SentryGun.SentryControl
{
    public class LockFreeStack<T>
    {
        private SingleLinkNode<T> head;
        private int m_Count;

        public int Count
        {
            get { return m_Count; }
            set { m_Count = value; }
        }

        public LockFreeStack()
        {
            head = new SingleLinkNode<T>();
```

```
        }

        public void Push(T item)
        {
            SingleLinkNode<T> newNode = new SingleLinkNode<T>();
            newNode.Item = item;

            do
            {
                newNode.Next = head.Next;
            }   while   (!SyncMethods.CAS<SingleLinkNode<T>>(ref   head.Next,   newNode.Next,
newNode));

            m_Count++;
        }

        private bool Pop(out T item)
        {
            SingleLinkNode<T> node;

            do
            {
                node = head.Next;
                if (node == null)
                {
                    item = default(T);
                    return false;
                }
            } while (!SyncMethods.CAS<SingleLinkNode<T>>(ref head.Next, node, node.Next));

            item = node.Item;
            return true;
        }

        public T Pop()
        {
            T result;
            Pop(out result);

            m_Count--;

            if (m_Count < 0)
                m_Count = 0;

            return result;
        }
    }
}
```

ServoController.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.IO.Ports;
using System.Threading;

using SentryGun.ErrorHandler;

namespace SentryGun.SentryControl
{
    public enum EnumServo
    {
        Servo1 = 1,
        Servo2,
        Servo3,
        Servo4,
        Servo5,
        Servo6,
        Servo7,
        Servo8
    }

    public static class ServoController
    {
        private static LockFreeStack<ServoScript> m_ScriptStack;
```

```
        private static string m_Buffer;

        public static string Buffer
        {
            get
            {
                return m_Buffer;
            }
            set
            {
                m_Buffer = value;
            }
        }

        public static LockFreeStack<ServoScript> ScriptStack
        {
            get
            {
                if (m_ScriptStack == null)
                    m_ScriptStack = new LockFreeStack<ServoScript>();

                return m_ScriptStack;
            }
            set
            {
                m_ScriptStack = new LockFreeStack<ServoScript>();
            }
        }

        public static ServoScript AddServoScript(EnumServo servo, int InitialPosition, int
FinalPosition, int Interval, bool Reverse)
        {
            try
            {
                return AddScriptGeneral(servo, InitialPosition, FinalPosition, Interval,
Reverse);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro adicionar script de movimentação", ex);
            }
        }

        public static void AddTargetScript(int TargetCameraX, int TargetCameraY)
        {
            try
            {
                int ServoCordinateX = Configuration.GetRelativeServoX(TargetCameraX);
                int ServoCordinateY = Configuration.GetRelativeServoY(TargetCameraY);

                // X
                ServoScript newScriptX = new ServoScript();
                newScriptX.Servo = Configuration.ServoX;
                newScriptX.InitialPosition = ServoCordinateX;
                newScriptX.FinalPosition = ServoCordinateX;
                newScriptX.Reverse = false;
                newScriptX.Interval = 0;

                // Y
                ServoScript newScriptY = new ServoScript();
                newScriptY.Servo = Configuration.ServoY;
                newScriptY.InitialPosition = ServoCordinateY;
                newScriptY.FinalPosition = ServoCordinateY;
                newScriptY.Reverse = false;
                newScriptY.Interval = 0;

                if (m_ScriptStack == null)
                    m_ScriptStack = new LockFreeStack<ServoScript>();

                //verificar limite de 5 pares de comandos
                if (m_ScriptStack.Count >= 10)
                {
                    do
                        m_ScriptStack.Pop();
                    while (m_ScriptStack.Count != 8);
                }
```

```
                //empilhar
                m_ScriptStack.Push(newScriptX);
                m_ScriptStack.Push(newScriptY);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro adicionar script de movimentação", ex);
            }
        }

        private static ServoScript AddScriptGeneral(EnumServo servo, int InitialPosition, int
FinalPosition, int Interval, bool Reverse)
        {
            try
            {
                if (m_ScriptStack == null)
                    m_ScriptStack = new LockFreeStack<ServoScript>();

                ServoScript newScript = new ServoScript();
                newScript.Servo = servo;
                newScript.InitialPosition = InitialPosition;
                newScript.FinalPosition = FinalPosition;
                newScript.Reverse = Reverse;
                newScript.Interval = Interval;

                m_ScriptStack.Push(newScript);

                return newScript;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro adicionar script", ex);
            }
        }

        public static void StartControllerBoard(SerialPort port, int InitialServosPosition,
int CommunicationSpeed)
        {

            try
            {
                m_Buffer = "";

                //não é necessário acrescentar CR no final dos comandos
                //pois esta propriedade já faz isso...
                port.NewLine = Environment.NewLine;

                if (port.IsOpen)
                    port.Close();

                port.Open();

                //comando de inicialização
                string startCommand = "SCStart" + CommunicationSpeed.ToString("00") +
InitialServosPosition.ToString("0000");

                port.WriteLine(startCommand);

            }
            catch (Exception ex)
            {
                throw new Exception("Erro iniciar placa controladora", ex);
            }
        }
    }
}
```

ServoScript.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Xml.Serialization;

using SentryGun.ErrorHandler;
```

```
namespace SentryGun.SentryControl
{
    [XmlRoot()]
    public class ServoScript
    {
        private int m_InitialPosition;
        private int m_FinalPosition;
        private bool m_Reverse;
        private int m_Interval;

        private EnumServo m_Servo;

        public EnumServo Servo
        {
            get
            {
                return m_Servo;
            }
            set
            {
                m_Servo = value;
            }
        }

        public int InitialPosition
        {
            get { return m_InitialPosition; }
            set { m_InitialPosition = value; }
        }

        public int FinalPosition
        {
            get { return m_FinalPosition; }
            set { m_FinalPosition = value; }
        }

        public bool Reverse
        {
            get { return m_Reverse; }
            set { m_Reverse = value; }
        }

        public int Interval
        {
            get { return m_Interval; }
            set { m_Interval = value; }
        }

        [XmlIgnore()]
        public string ExecutionCommand
        {
            get
            {
                try
                {
                    //SP + Numero do Servo (1 a 8) + Pos1 (50 a 1250) + Pos2 (50 a 1250) +
Inversao (0 ou 1) + Carriege Return

                    string command =
                        "SP" +
                        ((int)m_Servo).ToString("0") +
                        m_InitialPosition.ToString("0000") +
                        m_FinalPosition.ToString("0000") +
                        (m_Reverse == true ? "1":"0");

                    return command;
                }
                catch (Exception ex)
                {
                    throw new Exception("Erro ao executar script de movimentação", ex);
                }
            }
        }
    }
}
```

SingleLinkNode.cs

```
using System;
using System.Collections.Generic;
using System.Text;

namespace SentryGun.SentryControl
{
    internal class SingleLinkNode<T>
    {
        public SingleLinkNode<T> Next;
        public T Item;
    }
}
```

SyncMethods.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Threading;

namespace SentryGun.SentryControl
{
    public static class SyncMethods
    {
        public static bool CAS<T>(ref T location, T comparand, T newValue) where T : class
        {
            return     (object)   comparand   ==   (object)   Interlocked.CompareExchange<T>(ref
location, newValue, comparand);
        }
    }
}
```

Target.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

namespace SentryGun.SentryControl
{
    public class Target
    {

        private Rectangle m_CurrentRectangle;
        private bool m_Shooted;
        private bool m_Moving;
        private int m_Volume;

        public Target()
        {
        }

        public Rectangle CurrentRectangle
        {
            get { return m_CurrentRectangle; }
            set { m_CurrentRectangle = value; }
        }

        public bool Shooted
        {
            get { return m_Shooted; }
            set { m_Shooted = value; }
        }

        public int Volume
        {
            get { return m_Volume; }
            set { m_Volume = value; }
```

```
        }

        public bool Moving
        {
            get { return m_Moving; }
            set { m_Moving = value; }
        }
    }
}
```

TargetCollection.cs

```
using System;
using System.Collections;

namespace SentryGun.SentryControl
{

    public class TargetCollection : CollectionBase
    {

        #region Constructors

        public TargetCollection()
        {
        }

        public TargetCollection(TargetCollection value)
        {
            this.AddRange(value);
        }

        public TargetCollection(Target[] value)
        {
            this.AddRange(value);
        }

        #endregion

        #region Properties

        public Target this[int index]
        {
            get
            {
                return ((Target)(List[index]));
            }
            set
            {
                List[index] = value;
            }
        }
        #endregion

        #region Public Methods

        public int Add(Target value)
        {
            return List.Add(value);
        }

        public void AddRange(Target[] value)
        {
            for (int i = 0; (i < value.Length); i = (i + 1))
            {
                this.Add(value[i]);
            }
        }

        public void AddRange(TargetCollection value)
        {
            for (int i = 0; (i < value.Count); i = (i + 1))
            {
                this.Add(value[i]);
            }
```

```
}

public bool Contains(Target value)
{
    return List.Contains(value);
}

public void CopyTo(Target[] array, int index)
{
    List.CopyTo(array, index);
}

public int IndexOf(Target value)
{
    return List.IndexOf(value);
}

public void Insert(int index, Target value)
{
    List.Insert(index, value);
}

public new TargetEnumerator GetEnumerator()
{
    return new TargetEnumerator(this);
}

public void Remove(Target value)
{
    List.Remove(value);
}

#endregion

#region class TargetEnumerator

public class TargetEnumerator : object, IEnumerator
{

    private IEnumerator baseEnumerator;

    private IEnumerable temp;

    public TargetEnumerator(TargetCollection mappings)
    {
        this.temp = ((IEnumerable)(mappings));
        this.baseEnumerator = temp.GetEnumerator();
    }

    public Target Current
    {
        get
        {
            return ((Target)(baseEnumerator.Current));
        }
    }

    object IEnumerator.Current
    {
        get
        {
            return baseEnumerator.Current;
        }
    }

    public bool MoveNext()
    {
        return baseEnumerator.MoveNext();
    }

    bool IEnumerator.MoveNext()
    {
        return baseEnumerator.MoveNext();
    }

    public void Reset()
    {
```

```
                baseEnumerator.Reset();
            }

            void IEnumerator.Reset()
            {
                baseEnumerator.Reset();
            }
        }

        #endregion

    }
}
```

TargetControl.cs

```
using System;
using System.Collections.Generic;
using System.Text;
using System.Drawing;

using SentryGun.Motion;

namespace SentryGun.SentryControl
{
    public static class TargetControl
    {
        public     static     TargetCollection     ProcessTargets(IMotionDetector     detector,
TargetCollection inTargets)
        {
            try
            {
                TargetCollection outTargets = null;

                if (detector != null)
                {
                    if (detector.Rectangles != null && detector.Rectangles.GetUpperBound(0) !=
-1)
                    {
                        Console.WriteLine("Processando Alvos");
                        outTargets = RefreshTargets(detector.Rectangles, inTargets);
                        ShootTarget(outTargets);
                    }
                }

                return outTargets;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao processar alvos", ex);
            }
        }

        private     static     TargetCollection     RefreshTargets(Rectangle[]     rectangles,
TargetCollection inTargets)
        {
            try
            {
                int match = 0;
                TargetCollection outTargets = new TargetCollection();

                if (inTargets == null)
                    inTargets = new TargetCollection();

                foreach (Rectangle rect in rectangles)
                {
                    //verifica se o tamanho do alvo está dentro da tolerância
                    if ((rect.Width * rect.Height) > Configuration.MinimunVolume)
                    {
                        bool find = false;
                        foreach (Target oldTarget in inTargets)
                        {
                            if (oldTarget.CurrentRectangle.IntersectsWith(rect))
                            {
                                find = true;
                                match++;
```

```
                            oldTarget.Moving = TargetIsMoving(rect, oldTarget);
                            oldTarget.CurrentRectangle = rect;

                            outTargets.Add(oldTarget);
                            inTargets.Remove(oldTarget);
                            break;
                        }
                    }
                    if (!find)
                    {
                        Target newtarget = new Target();
                        newtarget.CurrentRectangle = rect;
                        newtarget.Moving = true;
                        outTargets.Add(newtarget);
                    }
                }
            }

            return outTargets;
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao atualizar alvos", ex);
        }
    }

    private static bool TargetIsMoving(Rectangle rect, Target oldTarget)
    {

        bool moving = false;

        //Cálculo de tolerância de alteração do centro
        double middleX_New = (rect.Bottom - rect.Top) / 2;
        double middleY_New = (rect.Right - rect.Left) / 2;
        double       middleX_Old       =       (oldTarget.CurrentRectangle.Bottom       -
oldTarget.CurrentRectangle.Top) / 2;
        double       middleY_Old       =       (oldTarget.CurrentRectangle.Right       -
oldTarget.CurrentRectangle.Left) / 2;
        double opposite = middleY_Old - middleY_New;
        double adjacent = middleX_Old - middleX_New;

        double distance = Math.Sqrt(Math.Pow(adjacent, 2) + Math.Pow(opposite, 2));
        double percent = Convert.ToDouble(Configuration.MovementPercentRejection) / 100;
        double tolerance = Configuration.CamWidth * percent;

        if (distance > tolerance)
            moving= true;

        //cálculo de variação de volume
        int         volumeOLD         =         oldTarget.CurrentRectangle.Width         *
oldTarget.CurrentRectangle.Height;
        int volumeNEW = rect.Width * rect.Height;
        int volumeDIFF = volumeNEW - volumeOLD;

        if (volumeDIFF > tolerance)
            moving = true;

        return moving;
    }

    private static void ShootTarget(TargetCollection targets)
    {
        try
        {
            //primeiro os que estão se movendo
            foreach (Target target in targets)
            {
                if (target.Moving || target.Shooted == false)
                {
                    Console.WriteLine("Atirando alvo");
                    ServoController.AddTargetScript(
                        (target.CurrentRectangle.Width / 2) + target.CurrentRectangle.X,
                        (target.CurrentRectangle.Height / 2) + target.CurrentRectangle.Y);

                    target.Shooted = true;
                    break;
```

```
                    }
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao atirar no alvo", ex);
            }
        }
    }
}
```

Pacote SentryGun.SentryControl

AboutDialog.cs

```
using System;
using System.Drawing;
using System.Windows.Forms;
using SentryGun.ErrorHandler;

namespace SentryGun.WinForms
{

        public partial class AboutDialog : System.Windows.Forms.Form
        {
                public AboutDialog()
                {
            try
            {
                InitializeComponent();
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar objeto.", ex);
            }
                }

                private           void           linkLabel1_LinkClicked(object           sender,
System.Windows.Forms.LinkLabelLinkClickedEventArgs e)
                {
            try
            {
                System.Diagnostics.Process.Start("IExplore", " http://" + linkLabel1.Text);
            }
            catch(Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
                }

        private void btnLogin_Click(object sender, EventArgs e)
        {
            try
            {
                this.Close();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }
        }
}
```

CaptureDeviceForm.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

using SentryGun.Dshow;
using SentryGun.Dshow.Core;
using SentryGun.ErrorHandler;

namespace SentryGun.WinForms
{
	public partial class CaptureDeviceForm : System.Windows.Forms.Form
	{
		private FilterCollection filters;
	 private string device;

		public string Device
		{
			get { return device; }
		}

		public CaptureDeviceForm()
		{
            try
            {
                InitializeComponent();

                try
                {
                    filters = new FilterCollection(FilterCategory.VideoInputDevice);

                    if (filters.Count == 0)
                        throw new ApplicationException();

                    foreach (Filter filter in filters)
                        deviceCombo.Items.Add(filter.Name);
                }
                catch (ApplicationException)
                {
                    deviceCombo.Items.Add("Não foram encontrado dispositivos locais");
                    deviceCombo.Enabled = false;
                    okButton.Enabled = false;
                }

                deviceCombo.SelectedIndex = 0;
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar objeto", ex);
            }
		}

		private void okButton_Click(object sender, System.EventArgs e)
		{
            try
            {
                device = filters[deviceCombo.SelectedIndex].MonikerString;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
		}
	}
}
```

frmMain.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;
using System.Data;
```

```
using System.Threading;

using SentryGun.Motion;
using SentryGun.Motion.VideoSource;
using SentryGun.AVI;
using SentryGun.SentryControl;
using SentryGun.ErrorHandler;

namespace SentryGun.WinForms
{
        public partial class frmMain : System.Windows.Forms.Form
    {

        #region Local Variables

        private const int m_StatLength = 15;
        private bool m_SentryStarted = false;
        private int m_StatIndex = 0, m_StatReady = 0;
               private int m_ServoX = 0;
        private int m_ServoY = 0;
        private int[] m_StatCount = new int[m_StatLength];
        private IMotionDetector m_Detector = null;
        private TargetCollection m_Targets = null;
        private Camera m_Camera = null;
        private int m_CountWastedTime = 0;
        private EnumTriggerControl m_TriggerControl = EnumTriggerControl.Idle;
        private bool m_SentryCentered = false;
        private enum EnumTriggerControl
        {
            Idle=0,
            Shoot,
            Shooting,
            Stop,
            Stopped
        }

        #endregion

        public frmMain ()
               {
            try
            {
                InitializeComponent();
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar objeto", ex);
            }
        }

        #region Events

        private void frmMain_Load(object sender, EventArgs e)
        {
            try
            {
                fillComboPorts();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void frmMain_Closing(object sender, System.ComponentModel.CancelEventArgs e)
               {
            try
            {
                CloseFile();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
               }

        private void btnStartBoard_Click(object sender, EventArgs e)
```

```
        {
            try
            {
                if(serialPort.IsOpen == false)
                    serialPort.PortName = (string)cboPorts.SelectedItem;

                //Inicializa a placa
                ServoController.StartControllerBoard(serialPort,
Configuration.ServoInitialPosition, Configuration.BoardCommunicationSpeed);

                //Take a nap...
                System.Threading.Thread.Sleep(1000);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

           private void exitFileItem_Click(object sender, System.EventArgs e)
           {
            try
            {
                this.Close();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
           }

           private void openFileItem_Click(object sender, System.EventArgs e)
           {
            try
            {
                if (openFileDialog.ShowDialog() == DialogResult.OK)
                {
                    VideoFileSource fileSource = new VideoFileSource();
                    fileSource.VideoSource = openFileDialog.FileName;

                    OpenVideoSource(fileSource);
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
           }

           private void openURLFileItem_Click(object sender, System.EventArgs e)
           {
            try
            {
                frmURL form = new frmURL();

                form.Description = "Informe a URL que forneça imagens a partir de uma
webcam:";
                form.URLs = new string[]
                            {
                                "http://194.18.89.220/axis-
cgi/jpg/image.cgi?resolution=320x240",
                                "http://iris.not.iac.es/axis-
cgi/jpg/image.cgi?resolution=320x240",
                                "http://webcam.mmhk.cz/axis-
cgi/jpg/image.cgi?resolution=320x240",
                                "http://80.232.34.33/axis-
cgi/jpg/image.cgi?resolution=352x240",
                                "http://129.186.47.239/axis-
cgi/jpg/image.cgi?resolution=352x240"
                            };

                if (form.ShowDialog(this) == DialogResult.OK)
                {
                    JPEGStream jpegSource = new JPEGStream();
                    jpegSource.VideoSource = form.URL;

                    OpenVideoSource(jpegSource);
```

```
            }
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
           }

           private void openMJEPGFileItem_Click(object sender, System.EventArgs e)
           {
        try
        {
            frmURL form = new frmURL();

            form.Description = "Informe a URL contendo um streaming de video:";
            form.URLs = new string[]
                        {
                                    "http://webcam-rade.ville-ge.ch/axis-
cgi/mjpg/video.cgi?resolution=320x240",
                                    "http://peeper.axisinc.com/nph-manupdate.cgi",
                                    "http://212.98.46.120/axis-
cgi/mjpg/video.cgi?resolution=352x288",
                                    "http://158.39.11.250/axis-
cgi/mjpg/video.cgi?resolution=352x240"
                        };

            if (form.ShowDialog(this) == DialogResult.OK)
            {
                MJPEGStream mjpegSource = new MJPEGStream();
                mjpegSource.VideoSource = form.URL;

                OpenVideoSource(mjpegSource);
            }
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
           }

           private void openLocalFileItem_Click(object sender, System.EventArgs e)
           {
        try
        {
            OpenLocalDevice();
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void timer_Elapsed(object sender, System.Timers.ElapsedEventArgs e)
    {
        try
        {
            Camera camera = cameraWindow.Camera;

            if (camera != null)
            {
                // obtém o número de frames recebidos no último segundo
                m_StatCount[m_StatIndex] = camera.FramesReceived;

                if (++m_StatIndex >= m_StatLength)
                    m_StatIndex = 0;
                if (m_StatReady < m_StatLength)
                    m_StatReady++;

                float fps = 0;

                // calcula média de frames
                for (int i = 0; i < m_StatReady; i++)
                {
                    fps += m_StatCount[i];
                }
                fps /= m_StatReady;
```

```
                    m_StatCount[m_StatIndex] = 0;

                    int frameCount = camera.FrameRejectNumber;
                    if (frameCount == 0)
                        frameCount++;

                    fpsPanel.Text = fps.ToString("F2") + " fps" + "   -   Tratando 1 Frame a
cada " + frameCount.ToString();
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void camera_NewFrame(object sender, System.EventArgs e)
        {
            try
            {
                if (m_SentryStarted)
                {
                    Console.WriteLine("Vai processar alvos");
                    m_Targets = TargetControl.ProcessTargets(m_Detector, m_Targets);
                }

            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector1_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if    (optDetector1.Checked    &&    cameraWindow.Camera    !=    null    &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector2_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if    (optDetector2.Checked    &&    cameraWindow.Camera    !=    null    &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector3_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if    (optDetector3.Checked    &&    cameraWindow.Camera    !=    null    &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector3Optimized_CheckedChanged(object sender, EventArgs e)
        {
```

```
            try
            {
                if   (optDetector3Optimized.Checked   &&   cameraWindow.Camera   !=   null   &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector4_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if    (optDetector4.Checked    &&    cameraWindow.Camera    !=    null    &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optNone_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if (optNone.Checked && cameraWindow.Camera.Running)
                {
                    m_Detector = null;
                    SetMotionDetector();
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void optDetector5_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                if    (optDetector5.Checked    &&    cameraWindow.Camera    !=    null    &&
cameraWindow.Camera.Running)
                    SetMotionDetector();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void sobreToolStripMenuItem_Click(object sender, EventArgs e)
        {
            try
            {
                AboutDialog frm = new AboutDialog();
                frm.ShowDialog();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnOpenCam_Click(object sender, EventArgs e)
        {
            try
            {
                OpenLocalDevice();
            }
            catch (Exception ex)
            {
```

```
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void btnServoTest_Click(object sender, EventArgs e)
    {
        try
        {
            frmServoTest frm = new frmServoTest();
            frm.Show();
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void btnStart_Click(object sender, EventArgs e)
    {
        try
        {
            if (m_SentryStarted)
                return;

            btnStart.Enabled = false;
            btnStop.Enabled = true;

            m_SentryStarted = true;

            backWorker.RunWorkerAsync();
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void btnStop_Click(object sender, EventArgs e)
    {
        try
        {
            backWorker.CancelAsync();
            m_SentryStarted = false;
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void abrirTestadorToolStripMenuItem_Click(object sender, EventArgs e)
    {
        try
        {
            frmServoTest frm = new frmServoTest();
            frm.Show();
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void btnStartCamera_Click(object sender, EventArgs e)
    {
        try
        {
            SetMotionDetector();

            m_Camera.FrameRejectNumber = Convert.ToInt32(txtFrameReject.Text);

            m_Camera.Start();
            timer.Start();
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
```

```
            }
        }

        private void btnStartCaliber_Click(object sender, EventArgs e)
        {
            try
            {
                //Posição Central
                m_ServoX = Convert.ToInt32(txtInitiaPositionX.Text);
                m_ServoY = Convert.ToInt32(txtInitiaPositionY.Text);

                txtServoXLeft.Text = Configuration.ServoXCaliberLeft.ToString();
                txtServoXRight.Text = Configuration.ServoXCaliberRight.ToString();
                txtServoYTop.Text = Configuration.ServoYCaliberTop.ToString();
                txtServoYBottom.Text = Configuration.ServoYCaliberBottom.ToString();

                ServoController.AddServoScript(Configuration.ServoX,  m_ServoX,  m_ServoX,  0,
false);
                ServoController.AddServoScript(Configuration.ServoY,  m_ServoY,  m_ServoY,  0,
false);

                backWorkerCaliber.RunWorkerAsync();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnStopCaliber_Click(object sender, EventArgs e)
        {
            try
            {
                backWorkerCaliber.CancelAsync();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnServoYTop_Click(object sender, EventArgs e)
        {
            try
            {
                //ServoController.ScriptStack = new LockFreeStack<ServoScript>();
                m_ServoY -= 5;
                ServoController.AddServoScript(Configuration.ServoY,  m_ServoY,  m_ServoY,  0,
false);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnServoXRight_Click(object sender, EventArgs e)
        {
            try
            {
                //ServoController.ScriptStack = new LockFreeStack<ServoScript>();
                m_ServoX += 5;
                ServoController.AddServoScript(Configuration.ServoX,  m_ServoX,  m_ServoX,  0,
false);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnServoYBottom_Click(object sender, EventArgs e)
        {
            try
            {
                //ServoController.ScriptStack = new LockFreeStack<ServoScript>();
                m_ServoY += 5;
```

```
                ServoController.AddServoScript(Configuration.ServoY,  m_ServoY,  m_ServoY,  0,
false);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnServoXLeft_Click(object sender, EventArgs e)
        {
            try
            {
                //ServoController.ScriptStack = new LockFreeStack<ServoScript>();
                m_ServoX -= 5;
                ServoController.AddServoScript(Configuration.ServoX,  m_ServoX,  m_ServoX,  0,
false);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnSetServoXRight_Click(object sender, EventArgs e)
        {
            try
            {
                Configuration.ServoXCaliberRight = m_ServoX;
                txtServoXRight.Text = m_ServoX.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnSetServoYBottom_Click(object sender, EventArgs e)
        {
            try
            {
                Configuration.ServoYCaliberBottom = m_ServoY;
                txtServoYBottom.Text = m_ServoY.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnSetServoXLeft_Click(object sender, EventArgs e)
        {
            try
            {
                Configuration.ServoXCaliberLeft = m_ServoX;
                txtServoXLeft.Text = m_ServoX.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnSetServoYTop_Click(object sender, EventArgs e)
        {
            try
            {
                Configuration.ServoYCaliberTop = m_ServoY;
                txtServoYTop.Text = m_ServoY.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtServoXLeft_KeyPress(object sender, KeyPressEventArgs e)
```

```
        {
            try
            {
                if (e.KeyChar == (char)Keys.Enter)
                    Configuration.ServoXCaliberLeft = Convert.ToInt32(txtServoXLeft.Text);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtServoXRight_KeyPress(object sender, KeyPressEventArgs e)
        {
            try
            {
                if (e.KeyChar == (char)Keys.Enter)
                    Configuration.ServoXCaliberRight = Convert.ToInt32(txtServoXRight.Text);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtServoYTop_KeyPress(object sender, KeyPressEventArgs e)
        {
            try
            {
                if (e.KeyChar == (char)Keys.Enter)
                    Configuration.ServoYCaliberTop = Convert.ToInt32(txtServoYTop.Text);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtServoYBottom_KeyPress(object sender, KeyPressEventArgs e)
        {
            try
            {
                if (e.KeyChar == (char)Keys.Enter)
                    Configuration.ServoYCaliberBottom = Convert.ToInt32(txtServoYBottom.Text);
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnServoCenter_Click(object sender, EventArgs e)
        {
            try
            {
                centerPosition();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        #endregion

        #region Functions

        private void centerPosition()
        {
            try
            {
                int         centerX         =         Configuration.ServoXCaliberLeft         +
((Configuration.ServoXCaliberRight - Configuration.ServoXCaliberLeft) / 2);
                int         centerY         =         Configuration.ServoYCaliberTop         +
((Configuration.ServoYCaliberBottom - Configuration.ServoYCaliberTop) / 2);
                ServoController.AddServoScript(Configuration.ServoX,   centerX,   centerX,   0,
false);
```

```
                ServoController.AddServoScript(Configuration.ServoY,   centerY,   centerY,   0,
false);
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao centralizar arma", ex);
            }
        }

        private void fillComboPorts()
        {

            cboPorts.Items.Clear();

            string[] ports = System.IO.Ports.SerialPort.GetPortNames();
            foreach (string port in ports)
            {
                cboPorts.Items.Add(port);
            }

            if (cboPorts.Items.Count > 0)
                cboPorts.SelectedIndex = 0;
        }

        private void OpenLocalDevice()
        {
            try
            {
                CaptureDeviceForm form = new CaptureDeviceForm();

                if (form.ShowDialog(this) == DialogResult.OK)
                {
                    CaptureDevice localSource = new CaptureDevice();
                    localSource.VideoSource = form.Device;

                    OpenVideoSource(localSource);
                }
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao abrir dispositivo local", ex);
            }
        }

           private void OpenVideoSource(IVideoSource source)
           {
            try
            {
                this.Cursor = Cursors.WaitCursor;

                CloseFile();

                m_Camera = new Camera(source, m_Detector);
                cameraWindow.Camera = m_Camera;
                m_StatIndex = m_StatReady = 0;
                m_Camera.NewFrame += new EventHandler(camera_NewFrame);

                this.Cursor = Cursors.Default;
            }
            catch (Exception ex)
            {
                this.Cursor = Cursors.Default;
                throw new Exception("Erro ao abrir VideoSource", ex);
            }
           }

           private void CloseFile()
           {
            try
            {
                Camera camera = cameraWindow.Camera;

                if (camera != null)
                {
                    cameraWindow.Camera = null;
                    camera.SignalToStop();
                    camera.WaitForStop();
```

```
                camera = null;

                if (m_Detector != null)
                    m_Detector.Reset();
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao fechar arquivo", ex);
        }
    }

    private void SetMotionDetector()
    {
        try
        {
            if (optDetector1.Checked == true)
                m_Detector = new MotionDetector1();

            if (optDetector2.Checked == true)
                m_Detector = new MotionDetector2();

            if (optDetector3.Checked == true)
                m_Detector = new MotionDetector3();

            if (optDetector3Optimized.Checked == true)
                m_Detector = new MotionDetector3Optimized();

            if (optDetector4.Checked == true)
                m_Detector = new MotionDetector4();

            if (optDetector5.Checked == true)
            {
                m_Detector = new MotionDetector5();
                double percent = Convert.ToDouble(txtTolerance.Text);
                double volume = cameraWindow.Width * cameraWindow.Height * (percent /
100);
                Configuration.MinimunVolume = Convert.ToInt32(volume);
                m_Detector.MinimumVolume = Convert.ToInt32(volume);
            }

            Camera camera = cameraWindow.Camera;

            if (camera != null)
            {
                camera.Lock();
                camera.MotionDetector = m_Detector;

                m_StatIndex = m_StatReady = 0;
                camera.Unlock();
            }
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao definir algoritmo de detecção", ex);
        }
    }

    #endregion

    #region BackWorker

    private void backWorker_DoWork(object sender, DoWorkEventArgs e)
    {
        try
        {
            int countWastedTime=0;
            EnumTriggerControl triggerControl = EnumTriggerControl.Idle;
            bool centered = false;
            do
            {
                ServoScript script1 = null;
                ServoScript script2 = null;

                script1 = ServoController.ScriptStack.Pop();
                script2 = ServoController.ScriptStack.Pop();
```

```
                    //Idle test
                    if (script1 == null && script2 == null)
                    {
                        countWastedTime++;

                        if (countWastedTime > 100)
                        {
                            if (!centered)
                            {
                                int     centerX     =     Configuration.ServoXCaliberLeft     +
((Configuration.ServoXCaliberRight - Configuration.ServoXCaliberLeft) / 2);
                                int     centerY     =     Configuration.ServoYCaliberTop     +
((Configuration.ServoYCaliberBottom - Configuration.ServoYCaliberTop) / 2);

                                string cmd1 = "SP" +
                                   ((int)Configuration.ServoX).ToString("0") +
                                   centerX.ToString("0000") +
                                   centerX.ToString("0000") +
                                   "0";

                                for (int icont = Configuration.BoardRepeatNumber; icont > 0;
icont--)
                                    serialPort.WriteLine(cmd1);

                                string cmd2 = "SP" +
                                   ((int)Configuration.ServoY).ToString("0") +
                                   centerY.ToString("0000") +
                                   centerY.ToString("0000") +
                                   "0";

                                for (int icont = Configuration.BoardRepeatNumber; icont > 0;
icont--)
                                    serialPort.WriteLine(cmd2);

                                centered = true;
                            }
                            countWastedTime = 0;
                        }

                        if (triggerControl != EnumTriggerControl.Stopped)
                            triggerControl = EnumTriggerControl.Stop;
                    }

                    //run script 1
                    if (script1 != null)
                    {
                        centered = false;
                        if (triggerControl != EnumTriggerControl.Shooting)
                            triggerControl = EnumTriggerControl.Shoot;

                        string cmd = script1.ExecutionCommand;

                        for (int icont = Configuration.BoardRepeatNumber; icont > 0; icont--)
                            serialPort.WriteLine(cmd);
                    }

                    //run script 2
                    if (script2 != null)
                    {
                        centered = false;
                        if (triggerControl != EnumTriggerControl.Shooting)
                            triggerControl = EnumTriggerControl.Shoot;

                        string cmd = script2.ExecutionCommand;

                        for (int icont = Configuration.BoardRepeatNumber; icont > 0; icont--)
                            serialPort.WriteLine(cmd);
                    }

                    //trigger behavior
                    if (triggerControl == EnumTriggerControl.Shoot)
                    {
                        string cmd = "SP" +
                            ((int)Configuration.ServoTrigger).ToString("0") +
                            Configuration.TriggerReleasedPosition.ToString("0000") +
                            Configuration.TriggerPressedPosition.ToString("0000") +
```

```
                        "0";

                    for (int icont = Configuration.BoardRepeatNumber; icont > 0; icont--)
                        serialPort.WriteLine(cmd);

                    triggerControl = EnumTriggerControl.Shooting;
                    centered = false;
                }

                if (triggerControl == EnumTriggerControl.Stop)
                {
                    string cmd = "SP" +
                        ((int)Configuration.ServoTrigger).ToString("0") +
                        Configuration.TriggerReleasedPosition.ToString("0000") +
                        Configuration.TriggerReleasedPosition.ToString("0000") +
                        "0";

                    for (int icont = Configuration.BoardRepeatNumber; icont > 0; icont--)
                        serialPort.WriteLine(cmd);

                    triggerControl = EnumTriggerControl.Stopped;
                    centered = false;
                }

            } while (backWorker.CancellationPending == false);
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private void backWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs
e)
    {
        try
        {
            Console.WriteLine("Fim de trabalho");
            btnStart.Enabled = true;
            btnStop.Enabled = false;

            m_SentryStarted = false;
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    #endregion

    #region BackWorkerCaliber

    private void backWorkerCaliber_DoWork(object sender, DoWorkEventArgs e)
    {
        try
        {
            do
            {
                ServoScript script1 = null;

                try
                {
                    script1 = ServoController.ScriptStack.Pop();
                }
                catch (InvalidOperationException)
                {
                    //não faz nada, pilha vazia
                }

                if (script1 != null)
                {
                    string cmd = script1.ExecutionCommand;

                    for (int icont = Configuration.BoardRepeatNumber; icont > 0; icont--)
                        serialPort.WriteLine(cmd);
```

```
                    backWorker.ReportProgress(10, script1);
                }

            } while (backWorkerCaliber.CancellationPending == false);
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }

    }

    private void backWorkerCaliber_ProgressChanged(object sender, ProgressChangedEventArgs
e)
    {
        try
        {

        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    private        void        backWorkerCaliber_RunWorkerCompleted(object        sender,
RunWorkerCompletedEventArgs e)
    {
        try
        {
            Console.WriteLine("Fim de trabalho");
            btnStart.Enabled = true;
            btnStop.Enabled = false;

            m_SentryStarted = false;
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }

    }

    #endregion

}
}
```

frmTestServo.cs

```
using System;
using System.Collections.Generic;
using System.ComponentModel;
using System.Data;
using System.Drawing;
using System.Text;
using System.Windows.Forms;
using System.IO.Ports;
using System.IO;
using System.Xml;
using System.Xml.Serialization;
using System.Threading;

using SentryGun.ErrorHandler;
using SentryGun.SentryControl;

namespace SentryGun.WinForms
{
    public partial class frmServoTest : Form
    {
        private string m_InputData = String.Empty;
        private int m_listIndex;

        delegate void SetTextCallback(string text);

        public frmServoTest()
```

```
        {
            try
            {
                InitializeComponent();
            }
            catch (Exception ex)
            {
                throw new Exception("Erro ao instanciar formulário", ex);
            }
        }

        #region Events

        private void frmServoTest_Load(object sender, EventArgs e)
        {
            try
            {
                fillComboPorts();

                PreConfigTab1();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void frmServoTest_FormClosing(object sender, FormClosingEventArgs e)
        {
            try
            {
                if (serialPort.IsOpen)
                    serialPort.Close();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void cboPortSelect_SelectedIndexChanged(object sender, EventArgs e)
        {
            try
            {
                if (serialPort.IsOpen)
                    serialPort.Close();

                propertyGrid1.Enabled = true;
                serialPort.PortName = cboPortSelect.SelectedItem.ToString();
                propertyGrid1.Refresh();

                toolStripStatusLabel.Text = serialPort.PortName + "--> Tx. Transf.: " +
serialPort.BaudRate + " bps, Bits de Dados: " + serialPort.DataBits + ", Paridade: " +
serialPort.Parity.ToString() + ", Bits de Parada: " + serialPort.StopBits;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void trackInitialPosition_ValueChanged(object sender, EventArgs e)
        {
            try
            {
                txtInitialPosition.Text = trackInitialPosition.Value.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void trackCommSpeed_ValueChanged(object sender, EventArgs e)
        {
            try
            {
```

```
                txtCommSpeed.Text = trackCommSpeed.Value.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void trackPos1_ValueChanged(object sender, EventArgs e)
        {
            try
            {
                txtPos1.Text = trackPos1.Value.ToString();

                if (chkStationary.Checked)
                    txtPos2.Text = txtPos1.Text;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void trackPos2_ValueChanged(object sender, EventArgs e)
        {
            try
            {
                txtPos2.Text = trackPos2.Value.ToString();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private           void           propertyGrid1_PropertyValueChanged(object           s,
PropertyValueChangedEventArgs e)
        {
            try
            {
                toolStripStatusLabel.Text =
                    serialPort.PortName +
                    "--> Tx. Transf.: " + serialPort.BaudRate +
                    " bps, Bits de Dados: " + serialPort.DataBits +
                    ", Paridade: " + serialPort.Parity.ToString() +
                    ", Bits de Parada: " + serialPort.StopBits;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnConfig_Click(object sender, EventArgs e)
        {
            try
            {
                StartControllerBoard();

                grpCommSpeed.Enabled = false;
                grpConfig.Enabled = false;
                grpInitialPosition.Enabled = false;
                grpServos.Enabled = false;
                btnConfig.Enabled = false;

                PreConfigTab2();

                tabControl1.SelectedTab = tabPage2;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnNewServoScript_Click(object sender, EventArgs e)
        {
```

```
            try
            {
                ClearScript();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnAddServoScript_Click(object sender, EventArgs e)
        {
            try
            {
                AddScript();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnDelServoScript_Click(object sender, EventArgs e)
        {
            try
            {
                if (lstScripts.SelectedItems.Count > 0)
                    DelScript();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnStart_Click(object sender, EventArgs e)
        {
            try
            {
                if (lstScripts.Items.Count == 0)
                {
                    MessageBox.Show(this,   "Lista   de   comandos   está   vazia!",   "Erro!",
MessageBoxButtons.OK, MessageBoxIcon.Error);
                    return;
                }

                //fillListServos();

                btnStart.Enabled = false;
                btnStop.Enabled = true;
                grpScripts.Enabled = false;
                btnSave.Enabled = false;
                btnClear.Enabled = false;
                btnOpen.Enabled = false;

                //locking stack
                //Monitor.Enter(ServoController.ScriptStack);

                //ServoController.ScriptStack = new LockFreeStack<ServoScript>();

                foreach (ListViewItem li in lstScripts.Items)
                    ServoController.ScriptStack.Push((ServoScript)li.Tag);

                //unlocking stack
                //Monitor.Exit(ServoController.ScriptStack);

                m_listIndex = 0;

                backWorker.RunWorkerAsync();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnStop_Click(object sender, EventArgs e)
```

```
{
    try
    {
        backWorker.CancelAsync();
    }
    catch (Exception ex)
    {
        ErrorHandlerControl.ShowErrorDialog(ref ex);
    }
}

private void btnClosePort_Click(object sender, EventArgs e)
{
    try
    {
        if (serialPort.IsOpen)
            serialPort.Close();
    }
    catch (Exception ex)
    {
        ErrorHandlerControl.ShowErrorDialog(ref ex);
    }
}

private void lstScripts_ItemActivate(object sender, EventArgs e)
{
    try
    {
        if (lstScripts.SelectedItems.Count == 0)
            return;

        ServoScript script = (ServoScript)lstScripts.SelectedItems[0].Tag;

        foreach (EnumServo servo in cboServo.Items)
        {
            if (servo == script.Servo)
            {
                cboServo.SelectedItem = servo;
                break;
            }
        }

        //txtRepeatNumber.Text = Configuration.BoardRepeatNumber.ToString();
        chkInverse.Checked = script.Reverse;
        txtInterval.Text = script.Interval.ToString();
        trackPos1.Value = script.InitialPosition;
        trackPos2.Value = script.FinalPosition;

        if (script.InitialPosition == script.FinalPosition)
            chkStationary.Checked = true;

        lstScripts.Items.Remove(lstScripts.SelectedItems[0]);
    }
    catch (Exception ex)
    {
        ErrorHandlerControl.ShowErrorDialog(ref ex);
    }
}

private void btnClear_Click(object sender, EventArgs e)
{
    try
    {
        lstScripts.Items.Clear();
        ClearScript();
    }
    catch (Exception ex)
    {
        ErrorHandlerControl.ShowErrorDialog(ref ex);
    }
}

private void txtPos2_KeyPress(object sender, KeyPressEventArgs e)
{
    try
    {
        if (e.KeyChar == (char)Keys.Enter)
```

```
                AddScript();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnSave_Click(object sender, EventArgs e)
        {
            try
            {
                saveFileDlg.FileName = "script " + DateTime.Now.ToString("yyyyMMdd hhmmss") +
".xml";
                DialogResult result = saveFileDlg.ShowDialog(this);

                if (result == DialogResult.Cancel || saveFileDlg.FileName == "")
                    return;

                List<ServoScript> list = new List<ServoScript>();

                ServoScript script = null;
                foreach (ListViewItem li in lstScripts.Items)
                {
                    script = (ServoScript)li.Tag;
                    list.Add(script);
                }

                FileStream f = new FileStream(saveFileDlg.FileName, FileMode.Create);
                XmlSerializer xmlSr = new XmlSerializer(list.GetType());
                xmlSr.Serialize(f, list);
                f.Close();
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void btnOpen_Click(object sender, EventArgs e)
        {
            try
            {

                DialogResult result = openFileDlg.ShowDialog(this);

                if (result == DialogResult.Cancel || openFileDlg.FileName == "")
                    return;

                List<ServoScript> list = new List<ServoScript>();
                FileStream f = new FileStream(openFileDlg.FileName, FileMode.Open);
                XmlSerializer xmlSr = new XmlSerializer(list.GetType());
                list = (List<ServoScript>)xmlSr.Deserialize(f);
                f.Close();

                //if(ServoController.ScriptStack != null)
                //    ServoController.ScriptStack = new LockFreeStack<ServoScript>();

                foreach (ServoScript script in list)
                {
                    ListViewItem li = new ListViewItem(script.Servo.ToString());
                    li.Tag = script;
                    li.SubItems.Add(script.InitialPosition.ToString("0000"));
                    li.SubItems.Add(script.FinalPosition.ToString("0000"));
                    li.SubItems.Add(chkInverse.Checked == true ? "Sim" : "Não");
                    li.SubItems.Add(script.Interval.ToString());
                    li.SubItems.Add("Aguardando");

                    lstScripts.Items.Add(li);
                }

                //fillListServos();

            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
```

```
            }
        }

        private void chkStationary_CheckedChanged(object sender, EventArgs e)
        {
            try
            {
                trackPos2.Enabled = !chkStationary.Checked;
                txtPos2.Enabled = !chkStationary.Checked;

                if (chkStationary.Checked)
                {
                    trackPos2.Value = trackPos1.Value;
                    txtPos2.Text = txtPos1.Text;
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        #endregion

        #region Validating

        private void txtInitialPosition_Validating(object sender, CancelEventArgs e)
        {
            try
            {
                int value = 0;
                try
                {
                    value = Convert.ToInt32(txtInitialPosition.Text);
                }
                catch (Exception ex)
                {
                    errorProvider1.SetError(txtInitialPosition, "Informe um valor entre 50 e
1250");
                    errorProvider1.SetIconAlignment(txtInitialPosition,
ErrorIconAlignment.MiddleLeft);
                    e.Cancel = true;
                    return;
                }

                if (value >= 50 && value <= 1250)
                {
                    errorProvider1.SetError(txtInitialPosition, "");
                    e.Cancel = false;
                    trackInitialPosition.Value = value;
                }
                else
                {
                    errorProvider1.SetError(txtInitialPosition, "Informe um valor entre 50 e
1250");
                    errorProvider1.SetIconAlignment(txtInitialPosition,
ErrorIconAlignment.MiddleLeft);
                    e.Cancel = true;
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtCommSpeed_Validating(object sender, CancelEventArgs e)
        {
            try
            {
                int value = 0;
                try
                {
                    value = Convert.ToInt32(txtCommSpeed.Text);
                }
                catch (Exception ex)
                {
```

```
                    errorProvider1.SetError(txtCommSpeed, "Informe um valor entre 0 e 99");
                    errorProvider1.SetIconAlignment(txtCommSpeed,
ErrorIconAlignment.MiddleLeft);
                    e.Cancel = true;
                }

                if (value >= 0 && value <= 99)
                {
                    errorProvider1.SetError(txtCommSpeed, "");
                    e.Cancel = false;
                    trackCommSpeed.Value = value;
                }
                else
                {
                    errorProvider1.SetError(txtCommSpeed, "Informe um valor entre 0 e 99");
                    errorProvider1.SetIconAlignment(txtCommSpeed,
ErrorIconAlignment.MiddleLeft);
                    e.Cancel = true;
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtPos1_Validating(object sender, CancelEventArgs e)
        {
            try
            {
                int value = 0;
                try
                {
                    value = Convert.ToInt32(txtPos1.Text);
                }
                catch (Exception ex)
                {
                    errorProvider1.SetError(txtPos1, "Informe um valor entre 50 e 1250");
                    errorProvider1.SetIconAlignment(txtPos1, ErrorIconAlignment.MiddleLeft);
                    e.Cancel = true;
                }

                if (value >= 50 && value <= 1250)
                {
                    errorProvider1.SetError(txtPos1, "");
                    e.Cancel = false;
                    trackPos1.Value = value;

                    if(chkStationary.Checked)
                        trackPos2.Value = value;
                }
                else
                {
                    errorProvider1.SetError(txtPos1, "Informe um valor entre 50 e 1250");
                    errorProvider1.SetIconAlignment(txtPos1, ErrorIconAlignment.BottomRight);
                    e.Cancel = true;
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void txtPos2_Validating(object sender, CancelEventArgs e)
        {
            try
            {
                int value = 0;

                try
                {
                    value = Convert.ToInt32(txtPos2.Text);
                }
                catch (Exception ex)
                {
                    errorProvider1.SetError(txtPos2, "Informe um valor entre 50 e 1250");
```

```
                errorProvider1.SetIconAlignment(txtPos2, ErrorIconAlignment.MiddleLeft);
                e.Cancel = true;
            }

            if (value >= 50 && value <= 1250)
            {
                errorProvider1.SetError(txtPos2, "");
                e.Cancel = false;
                trackPos2.Value = value;
            }
            else
            {
                errorProvider1.SetError(txtPos2, "Informe um valor entre 50 e 1250");
                errorProvider1.SetIconAlignment(txtPos2, ErrorIconAlignment.BottomRight);
                e.Cancel = true;
            }
        }
        catch (Exception ex)
        {
            ErrorHandlerControl.ShowErrorDialog(ref ex);
        }
    }

    #endregion

    #region Functions

    private void StartControllerBoard()
    {
        try
        {
            int InitialPosition = Convert.ToInt32(txtInitialPosition.Text);
            int CommunicationSpeed = Convert.ToInt32(txtCommSpeed.Text);

            ServoController.StartControllerBoard(serialPort,              InitialPosition,
CommunicationSpeed);

            cboServo.Items.Clear();
            if (chkServo1.Checked)
                cboServo.Items.Add(EnumServo.Servo1);
            if (chkServo2.Checked)
                cboServo.Items.Add(EnumServo.Servo2);
            if (chkServo3.Checked)
                cboServo.Items.Add(EnumServo.Servo3);
            if (chkServo4.Checked)
                cboServo.Items.Add(EnumServo.Servo4);
            if (chkServo5.Checked)
                cboServo.Items.Add(EnumServo.Servo5);
            if (chkServo6.Checked)
                cboServo.Items.Add(EnumServo.Servo6);
            if (chkServo7.Checked)
                cboServo.Items.Add(EnumServo.Servo7);
            if (chkServo8.Checked)
                cboServo.Items.Add(EnumServo.Servo8);

            //fillComboServos();
        }
        catch (Exception ex)
        {
            throw new Exception("Erro ao iniciar placa controladora", ex);
        }
    }

    private void ClearScript()
    {
        try
        {
            if(cboServo.Items.Count >0)
                cboServo.SelectedIndex = 0;

            //txtRepeatNumber.Text = "70";
            chkInverse.Checked = false;
            trackPos1.Value = 50;
            trackPos2.Value= 50;
            txtInterval.Text = "1000";
            chkStationary.Checked = false;
        }
```

```
    catch (Exception ex)
    {
        throw new Exception("Erro ao limpar campos", ex);
    }
}

private void AddScript()
{
    try
    {
        ServoScript script = new ServoScript();
        script.Servo = (EnumServo)cboServo.SelectedItem;
        script.InitialPosition = Convert.ToInt32(txtPos1.Text);
        script.FinalPosition = Convert.ToInt32(txtPos2.Text);
        script.Interval = Convert.ToInt32(txtInterval.Text);
        script.Reverse = chkInverse.Checked;

        ListViewItem li = new ListViewItem(script.Servo.ToString());
        li.Tag = script;
        li.SubItems.Add(script.InitialPosition.ToString("0000"));
        li.SubItems.Add(script.FinalPosition.ToString("0000"));
        li.SubItems.Add(chkInverse.Checked == true ? "Sim" : "Não");
        li.SubItems.Add(script.Interval.ToString());
        li.SubItems.Add("Aguardando");

        lstScripts.Items.Add(li);

        string p2 = txtPos2.Text;
        string interval = txtInterval.Text;

        ClearScript();

        txtPos1.Text = p2;
        txtInterval.Text = interval;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao adicionar script", ex);
    }
}

private void DelScript()
{
    try
    {
        lstScripts.Items.Remove(lstScripts.SelectedItems[0]);
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao remover script", ex);
    }
}

private void PreConfigTab1()
{
    try
    {
        cboPortSelect.SelectedIndex = 0;
        chkServo1.Checked = true;
        trackCommSpeed.Value = 10;
        trackInitialPosition.Value = 50;
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao configurar formulário", ex);
    }
}

private void PreConfigTab2()
{
    try
    {
        trackPos1.Value = 50;
        trackPos2.Value = 1250;
        //txtRepeatNumber.Text = "70";
        txtInterval.Text = "1000";
        btnStop.Enabled = false;
```

```
    }
    catch (Exception ex)
    {
        throw new Exception("Erro ao configurar formulário", ex);
    }
}

private void fillComboPorts()
{
    string[] ports = SerialPort.GetPortNames();

    foreach (string port in ports)
        cboPortSelect.Items.Add(port);
}


#endregion

#region BackGroundWorker

private void backWorker_DoWork(object sender, DoWorkEventArgs e)
{
    try
    {
        do
        {
            int icont = Configuration.BoardRepeatNumber;
            ServoScript script = null;

            //Lock na pilha
            //Monitor.Enter(ServoController.ScriptStack);

            try
            {
                script = ServoController.ScriptStack.Pop();
            }
            catch(InvalidOperationException)
            {
                //não faz se a fila estiver vazia
                //Monitor.Exit(ServoController.ScriptStack);
                break;
            }

            //UnLock na pilha
            //Monitor.Exit(ServoController.ScriptStack);

            if (script != null)
            {
                string cmd = script.ExecutionCommand;

                for (; icont > 0; icont--)
                    serialPort.WriteLine(cmd);

                backWorker.ReportProgress(10, script);

                if (script.Interval > 0)
                    System.Threading.Thread.Sleep(script.Interval);
            }

        } while (backWorker.CancellationPending == false);
    }
    catch (Exception ex)
    {
        ErrorHandlerControl.ShowErrorDialog(ref ex);
    }
}

private void backWorker_ProgressChanged(object sender, ProgressChangedEventArgs e)
{
    try
    {
        ListViewItem li = lstScripts.Items[m_listIndex];
        li.SubItems[5].Text = "Executando";

        m_listIndex++;
    }
    catch (Exception ex)
```

```
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void backWorker_RunWorkerCompleted(object sender, RunWorkerCompletedEventArgs
e)
        {
            try
            {
                btnStart.Enabled = true;
                btnStop.Enabled = false;
                grpScripts.Enabled = true;
                btnSave.Enabled = true;
                btnClear.Enabled = true;
                btnOpen.Enabled = true;
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        #endregion

        #region Buffer

        private void serialPort_DataReceived(object sender, SerialDataReceivedEventArgs e)
        {
            try
            {
                m_InputData = serialPort.ReadExisting();
                if (m_InputData != String.Empty)
                {
                    SetText(m_InputData);
                }
            }
            catch (Exception ex)
            {
                ErrorHandlerControl.ShowErrorDialog(ref ex);
            }
        }

        private void SetText(string text)
        {
            if (this.txtBuffer.InvokeRequired)
            {
                SetTextCallback d = new SetTextCallback(SetText);
                this.Invoke(d, new object[] { text });
            }
            else
                this.txtBuffer.Text += text;
        }

        #endregion

    }
}
```

frmURL.cs

```
using System;
using System.Drawing;
using System.Collections;
using System.ComponentModel;
using System.Windows.Forms;

namespace SentryGun.WinForms
{
       public partial class frmURL : System.Windows.Forms.Form
       {
              private string url;

              public string URL
              {
```

```
                get { return url; }
            }

            public string Description
            {
                set
                {
                    label1.Text = value;
                }
            }

            public string[] URLs
            {
                set
                {
                    urlCombo.Items.AddRange(value);
                }
            }

            public frmURL()
            {
                InitializeComponent();
            }

            private void okButton_Click(object sender, System.EventArgs e)
            {
                url = urlCombo.Text;
            }
        }
}
```

Program.cs

```
using System;
using System.Collections.Generic;
using System.Windows.Forms;

namespace SentryGun.WinForms
{
    static class Program
    {
        [STAThread]
        static void Main()
        {
            Application.EnableVisualStyles();
            Application.SetCompatibleTextRenderingDefault(false);
            Application.Run(new frmMain());
        }
    }
}
```

APÊNDICE B – FOTOS REAIS DO DISPOSITIVO

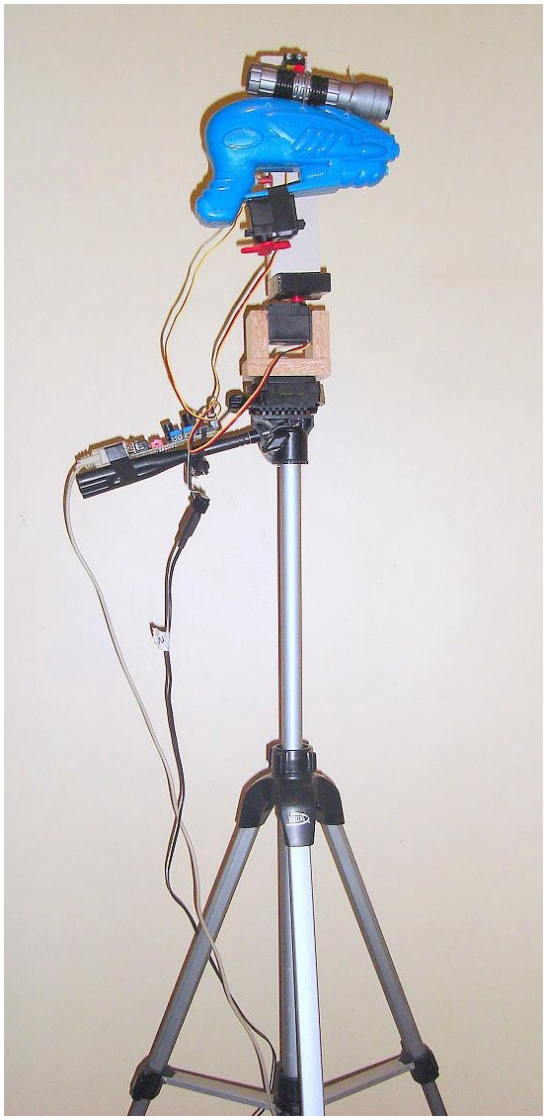


Figura B.1 – Foto da arma: vista geral

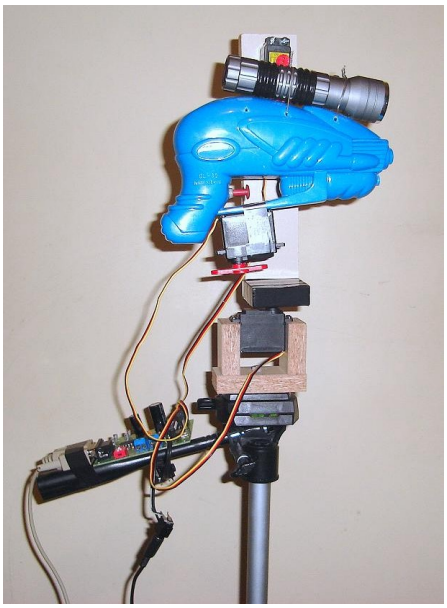


Figura B.2 – Foto da arma: vista lateral

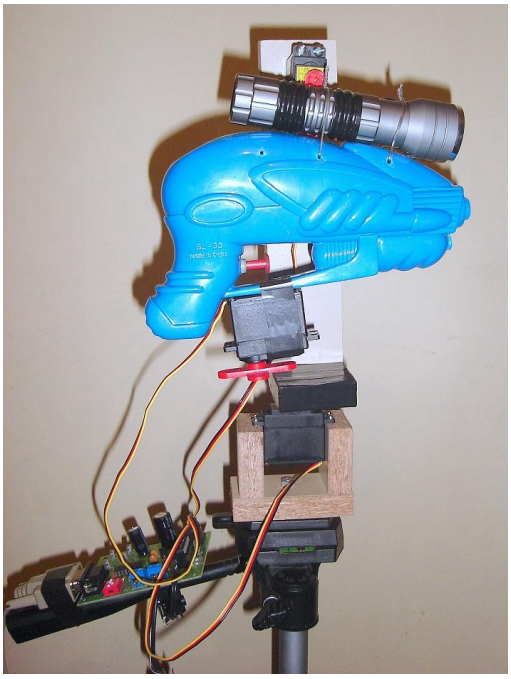


Figura B.3 – Foto da arma: vista lateral

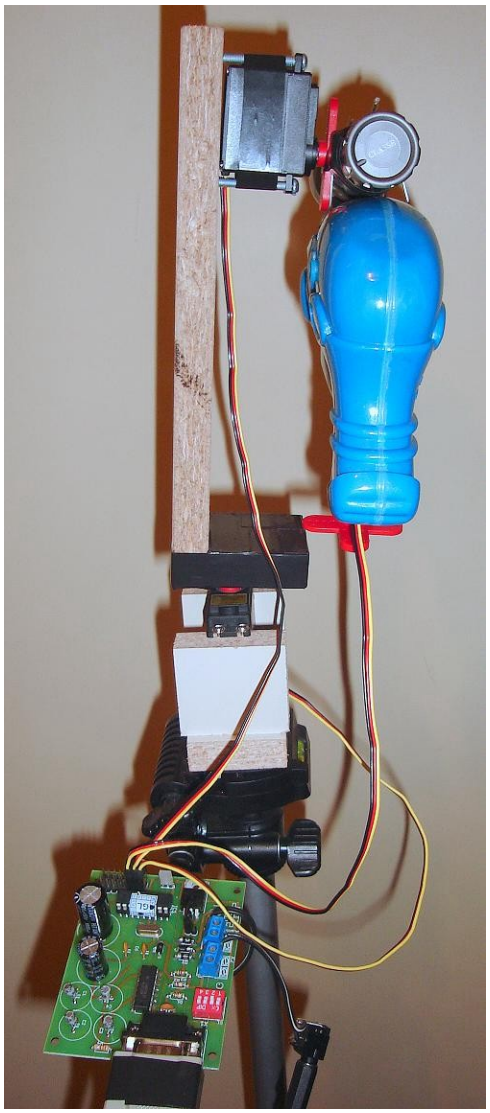


Figura B.4 – Foto da arma: vista posterior

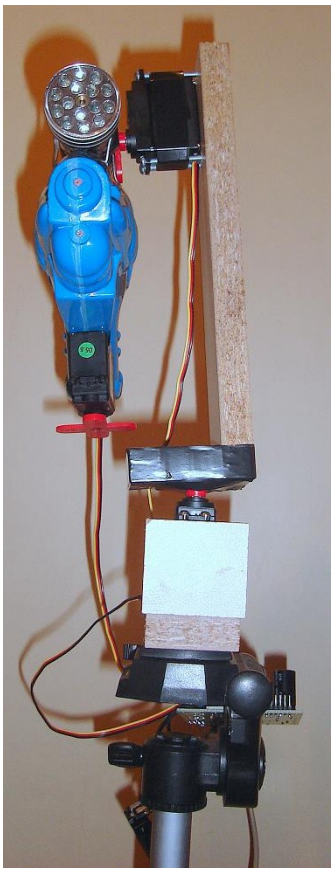


Figura B.5 – Foto da arma: vista frontal

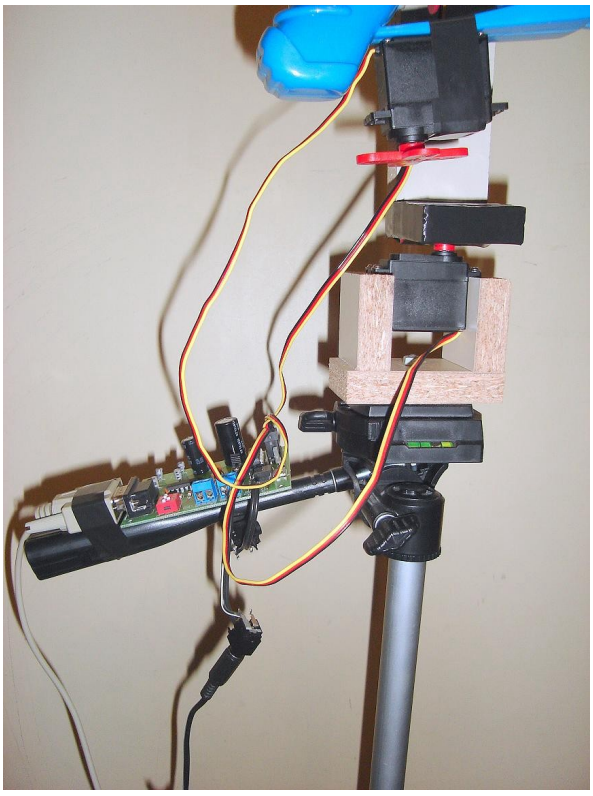


Figura B.6 – Foto da arma: vista da base

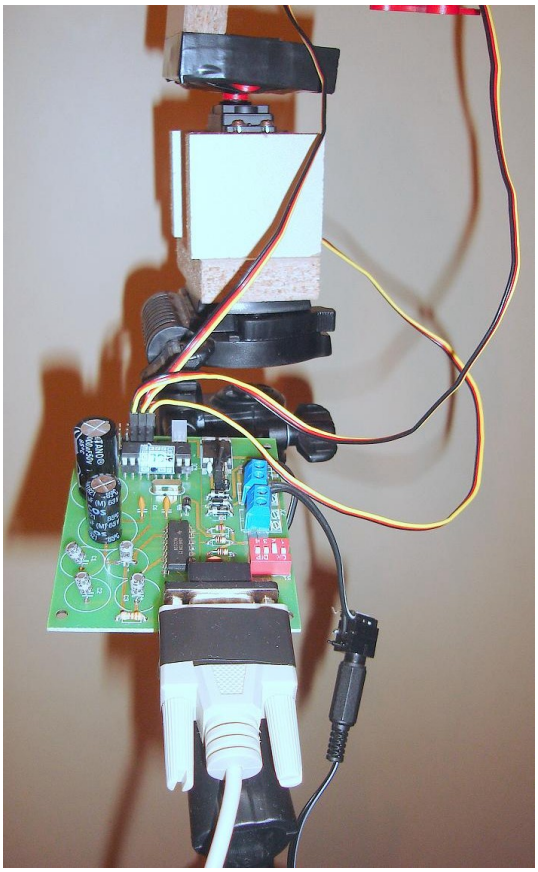


Figura B.7 – Foto da arma: placa controladora

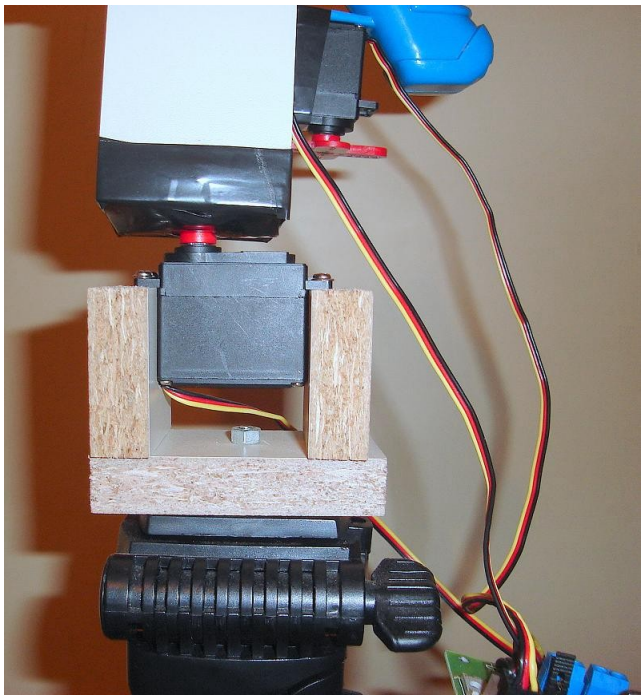


Figura B.8 – Foto da arma: servo do eixo X

This document was created with Win2PDF available at http://www.win2pdf.com.
The unregistered version of Win2PDF is for evaluation or non-commercial use only.
This page will not be added after purchasing Win2PDF.