ISABELA FERNANDES LAULETTA
TIAGO DE SOUZA MODESTO

RECONHECIMENTO DE PADRÕES EM IMAGENS UTILIZANDO MÉTODOS DE VISÃO COMPUTACIONAL PARA APLICAÇÕES EM REALIDADE AUMENTADA

MONOGRAFIA DO PROJETO DE CONCLUSÃO DE CURSO

SÃO PAULO
2010

ISABELA FERNANDES LAULETTA
TIAGO DE SOUZA MODESTO

RECONHECIMENTO DE PADRÕES EM IMAGENS UTILIZANDO MÉTODOS DE VISÃO COMPUTACIONAL PARA APLICAÇÕES EM REALIDADE AUMENTADA

Monografia apresentada à Escola Politécnica da Universidade de São Paulo para obtenção do diploma de graduação

Área de Concentração:
Engenharia Mecatrônica

Orientador:
Prof. Dr. Fabio Gagliardi Cozman

SÃO PAULO
2010

FICHA CATALOGRÁFICA

Lauletta, Isabela Fernandes
Reconhecimento de padrões em imagens utilizando métodos de visão computacional para aplicações em realidade aumentada / I.F. Lauletta, T.S. Modesto. -- São Paulo, 2010.
57 p.

Trabalho de Formatura - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia Mecatrônica e de Sistemas Mecânicos.

1. Visão computacional 2. Realidade virtual 3. Cinemática 4. Robôs I. Modesto, Tiago de Souza II.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia Mecatrô-nica e de Sistemas Mecânicos III. t.

RESUMO

Este trabalho tem por finalidade o estudo e a programação de algoritmos de visão computacional (para detecção de borda, segmentação, reconhecimento de padrões e localização de objetos de interesse em imagens) bem como sua utilização em realidade aumentada.

O usuário do programa utiliza um objeto (marcador) com um padrão pré determinado (como é de costume em realidade aumentada) que é filmado através de uma webcam. As imagens registradas são processadas pelos algoritmos desenvolvidos que efetuarão a localização da posição e da direção do marcador utilizado. Com isso é possível realizar a interpretação dos movimentos realizados pelo usuário.

Os dados de posição e rotação do marcador são utilizados para substituir a imagem do marcador pela imagem de outros objetos utilizados na aplicação de realidade aumentada.

A aplicação de realidade aumentada desenvolvida neste trabalho visa um projeto educativo na área de reciclagem que pode ser utilizado em eventos, feiras e convenções. A aplicação consiste de um robô virtual em duas dimensões (2D), com dois graus de liberdade (duas juntas rotativas), utilizado para coletar o lixo virtual que aparece no lugar do marcador do usuário e depositá-lo no cesto de descarte adequado (coleta seletiva).

O software final produzido neste trabalho foi programado em Java com o auxílio do ambiente integrado de desenvolvimento (IDE) Netbeans e utilizando as bibliotecas do Processing.

Palavras-chave: realidade aumentada, localização de padrões, detecção de borda, reconhecimento de padrões, visão computacional, cinemática inversa, robô RR

Sumário

1 INTRODUÇÃO

Realidade Aumentada (RA), ou em inglês Augmented Reality (AR), é uma área que está crescendo em pesquisas de realidade virtual. O mundo a nossa volta produz informações que são difíceis de copiar no computador, o que fica evidente em ambientes virtuais. Estes são normalmente bem simples, como os criados para entretenimento e jogos (Figura 1), mas também podem ser mais realistas como os simuladores de vôo.

Figura 1 Imagem de um ambiente virtual.

Um sistema de realidade aumentada gera uma vista composta do mundo real e do mundo virtual para o usuário. É uma combinação da cena real vista pelo usuário e de uma cena virtual criada pelo computador que "aumenta" a cena com informações adicionais.

As áreas de aplicação podem ser, por exemplo médica, de entretenimento, de treinamento militar, de projetos em engenharia, de propaganda, na robótica. Isso mostra as diversas formas de atuação dessa tecnologia. Em todas essas aplicações o destaque é a percepção que as pessoas têm do mundo ao seu redor e como agem sobre ele. Sendo assim, o objetivo final é se criar um sistema no qual o usuário não sabe onde termina a realidade e onde começa o espaço virtual; ele teria a impressão de

estar olhando para uma cena real. A Figura 2 mostra imagens da aplicação de realidade aumentada em algumas das áreas mencionadas. [1]

a) b)

Figura 2 Exemplos de aplicação da realidade no entretenimento (a) e na propaganda (b).

Alguns pesquisadores definem realidade aumentada como sendo um sistema que possui as seguintes propriedades: [2, 3, 4]

- Combina o ambiente real com objetos virtuais;
- Possui interatividade em tempo real;
- Alinha objetos reais e virtuais (em 3D).

A interatividade da RA permite que elementos virtuais acompanhem o posicionamento e os movimentos dos objetos reais. Há uma continuidade entre os dois mundos em tempo real. Para que o sistema identifique a localização, é necessário o uso de sensores eletromagnéticos ou de símbolos gráficos que são capitados pela câmera. O conceito de Realidade Aumentada pode também ser estendido para os outros sentidos: audição, tato e olfato. Além disso, outros dispositivos, como o Headed-Mounted Display (HMD), que é uma tecnologia que permite que o usuário veja a cena através de um dispositivo que coloca na cabeça), podem ser usados para se obter diferentes formas de realidades. Alguns dos tipos mais conhecidos de RA são:

- Optical See-through: projeta imagens e informações em um para-brisa ou outra interface transparente à frente da pessoa.
- Vídeo See-through: por meio de um capacete ou óculos pode-se ver imagens reais com informações virtuais.
- Espacial: a informação virtual é projetada sobre objetos da cena.
- Indireta: objetos virtuais são projetados no monitor e seguem movimentos do mundo real.

Sendo assim, nem sempre óculos especiais ou outro tipo de interface são necessários. Os elementos virtuais podem ser projetados diretamente nos objetos reais, tanto que o tipo de realidade aumentada mais barata e mais adotada atualmente é o indireto, o qual mescla o mundo real e o virtual na tela do PC.

Os celulares com câmera também servem como ferramenta para se explorar a realidade aumentada. Assim, um ambiente real pode ser filmado e, com a ajuda da tecnologia de processamento de imagens em tempo real e do reconhecimento automático de objetos, informações sobre o que se está sendo observado podem ser dispostas na imagem do display do celular junto com a realidade, por exemplo.

Uma das formas mais comuns de interagir com a realidade aumentada são os símbolos gráficos (como o da Figura 3) que atualmente estão presentes em impressões nas páginas de livros, em embalagens e em mídia impressa. Esses ícones são captados pela câmera e identificados por processos de reconhecimento de imagem, similares aos usados para ler impressões digitais. Os programas de RA analisam a deformação da imagem provocada pela visão em perspectiva, para que os elementos virtuais fiquem distribuídos apropriadamente no espaço 3D. [5]

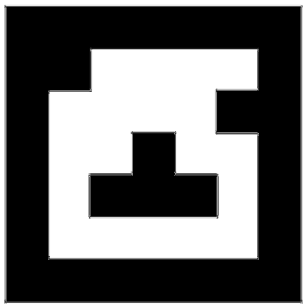


Figura 3 Exemplo de um símbolo gráfico utilizado em realidade aumentada.

Na aplicação desenvolvida neste trabalho o usuário interage com o mundo virtual através dos símbolos gráficos (exemplo apresentado na Figura 3, também chamado de marcador). Nela o usuário e o marcador que ele segura, são filmados por uma webcam. Porém apenas o usuário aparecerá na janela do programa e o marcador é substituído por um objeto virtual 2D (determinado pelo símbolo/código do marcador).

Como o tema escolhido para a aplicação é a reciclagem, os objetos que serão introduzidos pelo usuário na mesma serão resíduos de diferentes materiais. Uma vez que o resíduo é introduzido na aplicação ele continua sendo mostrado na mesma posição que o marcador segurado pelo usuário. Ou seja, se o usuário mover o marcador alterando a sua posição, o resíduo também tem sua posição alterada. Nota-se que para cada objeto, um padrão diferente deverá ser usado, e cabe ao programa de reconhecimento de padrão identificar à qual resíduo certo marcador está relacionado.

Quando outro marcador, chamado de marcador chave (não representa um objeto/resíduo), for mostrado para a webcam, um robô virtual 2D, com dois graus de liberdade, se movimenta até o resíduo e o removerá da mão do usuário. Depois ele se movimenta até o cesto de lixo adequado e descarta o resíduo de forma adequada, respeitando a coleta seletiva.

2 OBJETIVO E JUSTIFICATIVA

O objetivo deste trabalho de formatura é a elaboração de algoritmos de visão computacional para realizar a detecção de padrões em imagens, determinando a posição e direção dos mesmos e a utilização dessas informações em uma aplicação de realidade aumentada (software que permite a interação de um usuário no mundo real com uma aplicação virtual).

A motivação deste trabalho é a crescente utilização de realidade aumentada nas mais diversas aplicações.

Quanto à escolha do tema da aplicação desenvolvida, levou-se em consideração a contínua necessidade de investimento na conscientização da população em relação à reciclagem. Questão que apesar de ser amplamente discutida nos dias atuais ainda não é praticada em sua totalidade pelas pessoas.

3 FUNDAMENTOS TEÓRICOS

Nesta seção, serão descritos brevemente os principais métodos e conceitos teóricos estudados e utilizados na formulação dos algoritmos de processamento das imagens e do posicionamento do robô virtual da aplicação.

Normalmente antes de se analisar uma imagem, realiza-se um pré-processamento na mesma a fim de eliminar dados que não são de interesse e tenta-se ressaltar os que são importantes para a análise. Um pré-processamento muito utilizado é o de detecção de borda, porém a entrada para a realização deste processo é a imagem em sua escala de cinza. Portanto inicialmente será introduzida como esta transformação para a escala de cinza é realizada e em seguida será descrito o método de Sobel para detecção de borda. Por último será apresentado o método Hough Transform para localização de retas em imagens. O conceito de imagem digital também será apresentado.

Neste item de fundamentos também serão explicados os cálculos usados para se fazer a cinemática inversa e direta do robô virtual.(Seção 3.2)

3.1 Teorias relativas ao processamento de Imagens

3.1.1 Conceito de Imagem Digital

Uma imagem digital é a representação de uma imagem bidimensional usando números binários codificados de modo a permitir seu armazenamento, transferência, impressão ou reprodução, e seu processamento por meios eletrônicos.

Há dois tipos fundamentais de imagem digital. Uma é do tipo de rastreio (que é a usada neste projeto) e outra do tipo vetorial. Uma imagem digital do tipo raster, ou *bitmap*, ou ainda matricial, é aquela que em algum

momento apresenta uma correspondência bit-a-bit entre os pontos da imagem raster e os pontos da imagem reproduzida na tela de um monitor. A imagem vetorial não é reproduzida necessariamente por aproximação de pontos, antes era destinada a ser reproduzida por plotters de traçagem que reproduziam a imagem por deslocamento de canetas-tinteiro.

Tipicamente, as imagens raster são imagens fotográficas, e as imagens vetoriais são desenhos técnicos de engenharia. Os quadrinhos ilustrados se assemelham em qualidade a imagens raster, mas são impressos em *plotters* que passaram a imprimir à maneira das impressoras comuns por jato de tinta.[7]

3.1.2 Escala de Cinza

Uma imagem digital é uma imagem discretizada em pequenas unidades chamadas de pixel. Cada pixel de uma imagem digital corresponde a uma cor que pode ser codificada de diferentes formas. Uma codificação comum é a RGB, onde cada pixel corresponde a uma composição de intensidades de vermelho (R do inglês red), verde (G do inglês green) e azul (B do inglês blue). Nesta codificação quando os valores de intensidade de vermelho, verde e azul são iguais é caracterizada uma cor na escala cinza.

Existem problemas de visão computacional onde apenas estamos interessados na intensidade da cor e para simplificar a imagem que vamos analisar podemos transformá-la em escala de cinza (ou nível de cinza). Para isso devemos identificar as componentes de vermelho, verde e azul do pixel e aplicar uma transformação que represente sua intensidade. Uma transformação que é normalmente utilizada é de 30% da intensidade do vermelho, 59% da intensidade do verde e 11% do azul para a obtenção da intensidade de cinza. Ou seja, em cada pixel o valor da intensidade de vermelho é multiplicado por 0,30, o valor da intensidade de verde é multiplicado por 0,59 e o do azul por 0,11; esses valores são somados para se obter o valor da intensidade do cinza. Na codificação RGB citada acima para obter a cor cinza basta atribuir o valor da intensidade do cinza nas três

componentes de cores (vermelho, verde e azul). A Figura 4 mostra algumas intensidades de cor que a escala de cinza admite.

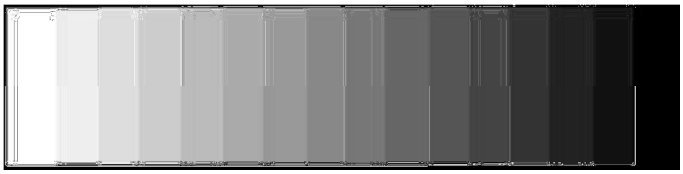


Figura 4 Algumas tonalidades da escala de cinza

3.1.3 Método de Sobel para Detecção de Borda

Detectar bordas em uma imagem é uma poderosa forma de selecionar informações úteis na mesma, e é um problema de fundamental importância no processamento de imagens. Pelo fato do custo computacional para analisar toda a imagem ser muito grande esta ferramenta é utilizada para diminuir a complexidade e a quantidade de dados que devem ser analisados, uma vez que na maioria das vezes são jogadas fora informações que não são de interesse, porém propriedades estruturais importantes das imagens são mantidas.

Bordas em imagens são áreas com forte intensidade de contraste, ou seja, um salto de intensidade de um pixel para o próximo. Existem muitas formas de realizar detecção de bordas. Entretanto, a maioria dos métodos podem ser divididos em dois grupos: análise do gradiente ou do laplaciano.

O método por gradiente detecta bordas olhando para o mínimo e o máximo da primeira derivada da imagem. Já por laplaciano procura-se por cruzamento de zeros na segunda derivada da imagem.

Existem vários métodos para aplicar detecção de bordas em imagens; os de Sobel e Canny são muito utilizados na área de visão computacional.[6]

O método de Sobel foi escolhido para ser utilizado na detecção de bordas das imagens que desejamos categorizar. Este método consiste em uma forma de calcular aproximadamente a derivada em uma imagem 2-D (equivalente a uma superfície). O operador de Sobel realiza a medida do gradiente espacial de uma imagem. Na verdade ele calcula uma

aproximação do gradiente, já que a imagem é discreta, e não uma função contínua.

Supondo que a imagem é uma função discreta I(x, y) de duas variáveis, onde x é direção horizontal e y é direção vertical, a aproximação do gradiente é dada por:

$$G_x(x,y) = -I(x-1,y-1) - 2 \times I(x-1,y) - I(x-1,y+1) + I(x+1,y-1) + 2 \times I(x+1,y) + I(x+1,y+1)$$

$$G_y(x,y) = -I(x-1,y-1) - 2 \times I(x,y-1) - I(x+1,y-1) + I(x-1,y+1) + 2 \times I(x,y+1) + I(x+1,y+1)$$

A magnitude do gradiente G(x, y) da imagem indica quão suavemente a imagem muda no ponto, e é dada por:

$$G(x,y) = \sqrt{{G_x}^2 + {G_y}^2}$$

Portanto, o Sobel é tipicamente usado para aproximar a magnitude absoluta do gradiente em cada ponto (pixel) de uma imagem em escala de cinza.

3.1.4 Método Hough Transform

A técnica da Transformada de Hough é consagrada em processamento de imagens para detectar elementos de uma imagem. Ela serve para detectar qualquer tipo de estrutura em uma imagem, mas neste trabalho será apenas analisado o caso mais simples que é o de detecção de linhas retas descritas pela equação:

$$y = mx + b$$

A transformada de Hough mapeia cada reta em um espaço de pares (m,b). Como uma reta é definida por dois parâmetros, uma imagem auxiliar é usada pela Transformada de Hough, ou seja, a imagem (x,y) é mapeada em uma imagem (m,b), chamada *acumulador*, às vezes também chamado de *espaço de parâmetros* ou *espaço de Hough.*

Implementações de detecção de linhas usando Transformada de Hough utilizam uma representação alternativa para retas, na forma de um par (r,Θ). O problema da parametrização (m,b) é que uma reta vertical não pode ser representada. Nesta nova parametrização, o parâmetro r representa a distância da reta até a origem e o parâmetro Θ representa o ângulo entre o eixo x e a reta entre origem e ponto mais próximo da origem, como mostrado na Figura 5.

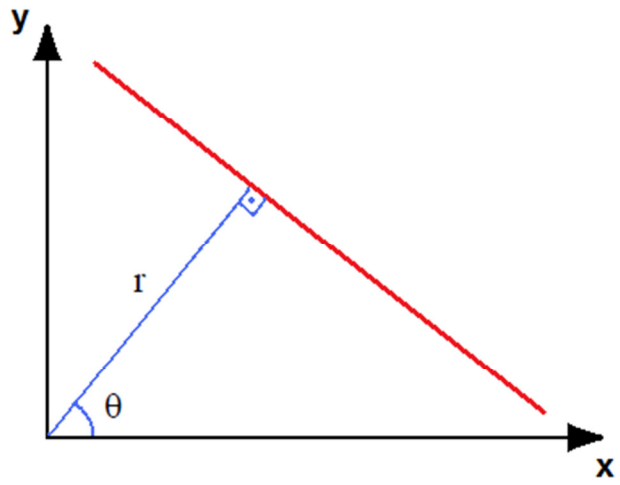


Figura 5 Gráfico indicando a representação (r, Θ) utilizada na Transformada Hough

A equação de uma reta é então dada por:

$$r = x\cos\theta + y\sin\theta$$

Se tivermos um ponto (x',y') que pertence a uma reta na imagem original, um número infinito de retas passa por esse ponto. Ou seja, um número infinito de pares (r, Θ) é compatível com (x',y'). A idéia da Transformada de Hough é adicionar 1 unidade a cada par (r, Θ) que seja compatível com um ponto (x',y') onde ocorre uma borda na imagem original.

Ou seja, dado um ponto (x',y') na imagem original, "desenhamos" a seguinte curva (soma de senóides) no acumulador:

$$r(\Theta) = x' \cos \Theta + y' \sin \Theta$$

Se uma determinada reta é desenhada na imagem, ocorrerá um acúmulo de pontos no acumulador, pois cada ponto da reta "vota" no mesmo ponto do acumulador. Se procurados os pontos de máximo local no acumulador, serão encontradas as retas na imagem original. Em termos de implementação, é preciso discretizar r e Θ usando o ângulo entre 0º e 180º em intervalos de 1º, por exemplo.

A algoritmo da Transformada de Hough apesar de apresentar uma possível solução para a detecção de objetos caracterizados por retas acabou não sendo utilizado na detecção do padrão uma vez que desenvolvemos outro algoritmo que apresentou um custo computacional inferior.

3.2 Teoria relativa à movimentação do robô

Nesta sesão serão descritos os métodos estudados e utilizados para se calcular a cinemática inversa e direta de um robô de dois graus de liberdade com duas juntas rotativas (RR).

3.2.1 Cinemática Inversa de um robô RR

Um braço robótico é composto por uma base, braço, antebraço e pulso, denominados elos. Para a conexão desses elos são usadas juntas, onde são acoplados os acionadores para realizarem os movimentos dos elos individualmente, com capacidade sensorial, e instruído por um sistema de controle.

No caso de um robô RR, as duas juntas são de revolução (dois graus de liberdade), e para o desenvolvimento dos cálculos chamaremos o braço de ligamento 1, o antebraço de ligamento 2. Na Figura 6 pode-se observar uma imagem esquemática de um robô desse tipo. [11]

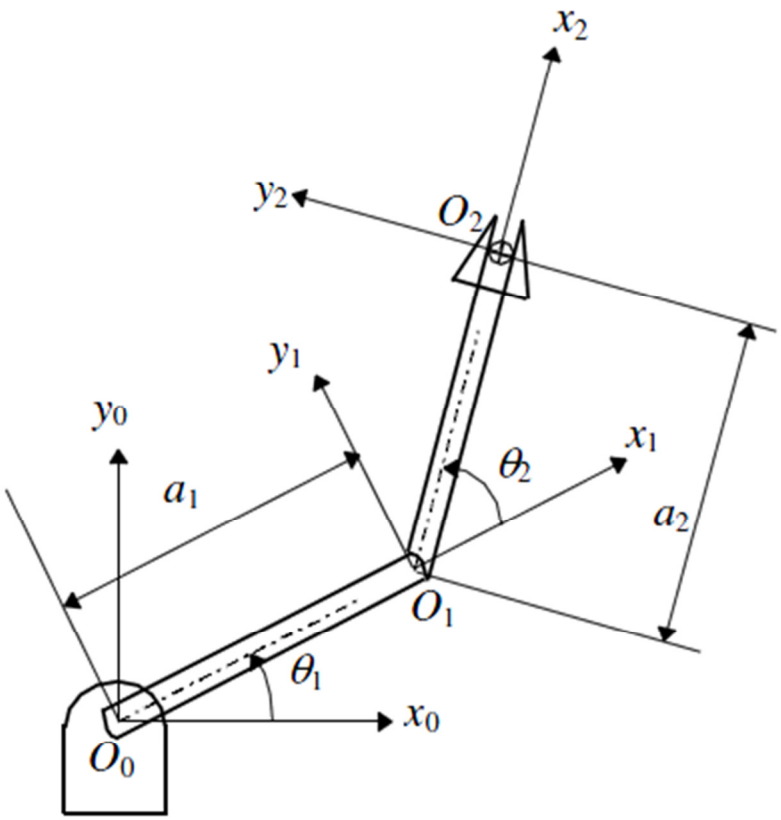


Figura 6 Esquema de um robô RR

Com o uso da notação de Denavit-Hartenberg, podemos montar as matrizes A^1_0 e A^2_1 e em seguida calcular a matriz $A^2_0 = A^1_0 A^2_1$ para seguirmos com as contas.

Tabela 1 Tabela de Denavit-Hartenberg para o robô RR

Ligamento i	ai	αi	di	θi
1	a1	0	0	θ1
2	a2	0	0	θ2

Utilizando a seguinte notação obteremos a fórmula geral para a formação das matrizes:

- ci = cos(θi);
- si = sen(θi);
- cij = cos(θi + θj) = ci*cj – si*sj;
- sij = sen(θi + θi) = si*cj + sj*ci;

$$A_{i-1}^{i} = \begin{bmatrix} C\theta_i & -S\theta_i C\alpha_i & S\theta_i S\alpha_i & a_i C\theta_i \\ S\theta_i & C\theta_i \cos\alpha_i & -C\theta_i S\alpha_i & a_i S\theta_i \\ 0 & S\alpha_i & C\alpha_i & d_i \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Sendo assim, temos que:

$$\mathbf{A}_0^1 = \begin{bmatrix} C_1 & -S_1 & 0 & a_1C_1 \\ S_1 & C_1 & 0 & a_1S_1 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad \mathbf{A}_1^2 = \begin{bmatrix} C_2 & -S_2 & 0 & a_2C_2 \\ S_2 & C_2 & 0 & a_2S_2 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

$$\mathbf{A}_0^2 = \mathbf{A}_0^1\mathbf{A}_1^2 = \begin{bmatrix} C_{12} & -S_{12} & 0 & a_1C_1 + a_2C_{12} \\ S_{12} & C_{12} & 0 & a_1S_1 + a_2S_{12} \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

No estudo da cinemática inversa, dado um ponto P (px, py), deve-se encontrar o valor dos ângulos θ_1 e θ_2. Sendo assim, pode-se escrever as seguintes equações:

$$\begin{cases} a_2 \times c_2 = px \times c_1 + py \times s_1 - a_1 \\ a_2 \times s_2 = -px \times s_1 + py \times c_1 \end{cases}$$

Manipulando-se essas duas equações, consegue-se escrever um equação para se determinar o valor de θ_1 e outra equação para se achar o valor de θ_2 (em função do θ_1 encontrado).

Por este ser um robô simples, de apenas dois graus de liberdade, geometricamente, também é possível encontrar o valor dos ângulos θ_1 e θ_2 para o caso desse robô RR. A cinemática inversa apresenta duas soluções possíveis, conforme mostra a Figura 7, com o cotovelo acima ($\theta_2 < 0$) e cotovelo abaixo ($\theta_2 > 0$).

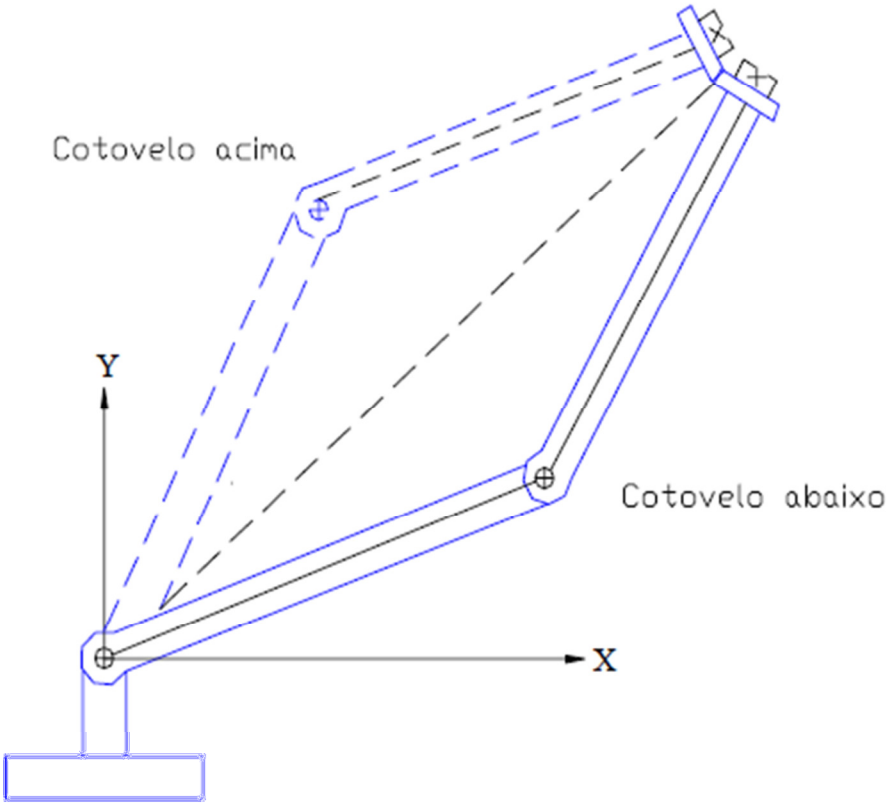


Figura 7 Soluções possíveis para a cinemática inversa do robô RR

Aplicando-se a lei dos co-senos ao triângulo formado pelo centro das duas juntas e pela posição do órgão terminal, obtem-se o valor do ângulo θ_2, dado por:

$$\theta_2 = \pm \arccos\left(\frac{x^2 + y^2 - l_1^2 - l_2^2}{2\ l_1\ l_2} \right)$$

onde a (x, y) é o pondo dado para onde o robô deverá se mover, e l_1 e l_2 são respectivamente os comprimentos do braço 1 e 2.

Por outro lado, da trigonometria tira-se que:

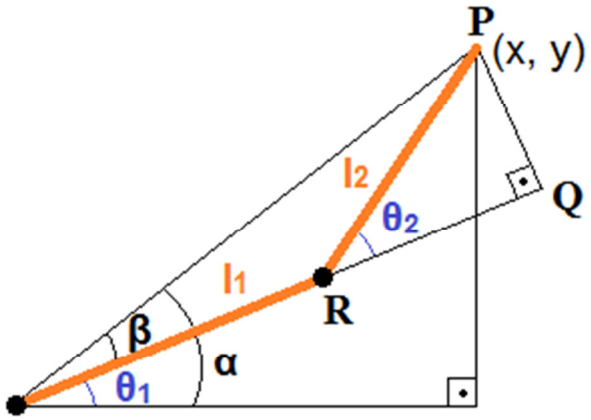


Figura 8 Cálculo geométrico da cinemática inversa do robô RR.

Da Figura 8 acima, conseguimos ver que:

$$\tan\beta = \frac{PQ}{l_1 + RQ} = \frac{l_2 \operatorname{sen}\theta_2}{l_1 + l_2\cos\theta_2} \qquad \text{e também que} \qquad \tan\alpha = \frac{y}{x}$$

Usando a seguinte identidade trigonométrica:

$$\tan\theta_1 = \tan(\alpha - \beta) = \frac{\tan\alpha - \tan\beta}{1 + \tan\alpha \ \ \tan\beta}$$

E uma vez que $\theta_1 = \alpha - \beta$, obtém-se:

$$\tan\theta_1 = \frac{y(l_1 + l_2\cos\theta_2) - xl_2\operatorname{sen}\theta_2}{x(l_1 + l_2\cos\theta_2) + yl_2\operatorname{sen}\theta_2}$$

Dessa forma, primeiro encontramos o valor de θ_2 e depois o de θ_1.

3.2.2 Cinemática Direta de um robô RR

O cálculo da cinemática direta de um robô RR é bem mais simples. Neste, os ângulos θ_1.e θ_2 são dados e o que desejamos descobrir é onde estará a ponta do efetuador, ou seja, qual é o ponto (x, y).

Seguindo o mesmo procedimento do item anterior, escrevemos a tabela de Denavit-Hartenberg, para montar as matrizes A^1_0 e A^2_1 e em seguida calcular a matriz $A^2_0 = A^1_0\, A^2_1$ e seguirmos com as contas.

Como já mostrado, a matriz A^2_0 fica então:

$$A_0^2 = \begin{bmatrix} c_{12} & -s_{12} & 0 & (l_1 c_1 + l_2 c_{12}) \\ s_{12} & c_{12} & 0 & (l_1 s_1 + l_2 s_{12}) \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix}$$

Olhando para a última coluna de A^2_0 podemos encontrar a relação entre os ângulos θ_1.e θ_2 e o ponto (x, y), como mostrado abaixo:

$$x = l_1 \cos\theta_1 + l_2 \cos(\theta_1 + \theta_2)$$
$$y = l_1 \sin\theta_1 + l_2 \sin(\theta_1 + \theta_2)$$

Assim, basta substituir os valores de .e θ_2 nas equações acima para se encontrar o valor das coordenadas do ponto (x, y).

4 FORMULAÇÃO DO PROJETO

O objetivo deste trabalho é desenvolver um programa de realidade aumentada. Sendo assim, neste item serão mostrados os parâmetros a serem determinados para que seja possível descrever totalmente a posição do objeto no espaço. Por último será apresentado à aplicação prática de realidade aumentada a ser desenvolvida.

4.1 Determinação de Parâmetros

4.1.1 Localização do Objeto

A localização do objeto de interesse é o principal parâmetro que deve ser determinado em um projeto que visa realizar uma aplicação de realidade aumentada. Isso porque com apenas este parâmetro já é possível realizar algumas aplicações simples. O parâmetro que desejamos determinar normalmente é a posição dos vértices do quadrado preto em volta do padrão quanto, pois através deles é possível determinar o valor do centro do mesmo.

4.1.2 Reconhecimento do Padrão

O reconhecimento de padrão é necessário em aplicações de realidade aumentada em que se deseja interagir com mais de um objeto, ou seja, são utilizados padrões diferentes para representar objetos diferentes no mundo virtual. Um exemplo de padrões diferentes pode ser encontrado na Figura 9.

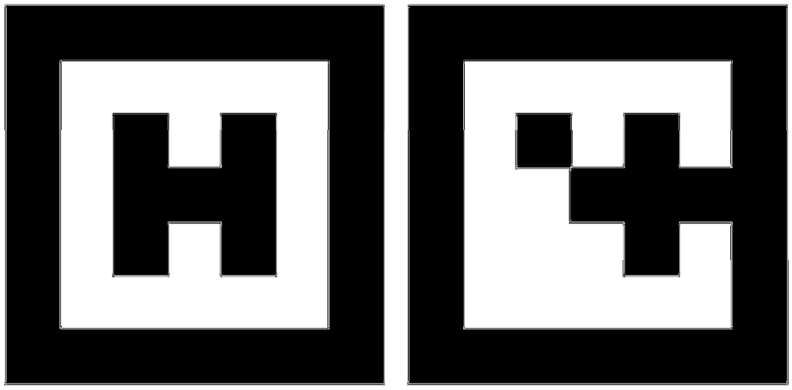


Figura 9 Exemplo de diferentes padrões para representar diferentes objetos

O algoritmo deve reconhecer os padrões determinados para se utilizar na realidade aumentada e o parâmetro a ser encontrado é o identificador referente ao padrão localizado.

4.1.3 Rotação do Objeto em Torno do Eixo z

A rotação em torno do eixo z é a rotação realizada em torno do eixo perpendicular ao plano do objeto que possui o padrão. A Figura 10 mostra uma imagem que representa este tipo de rotação.

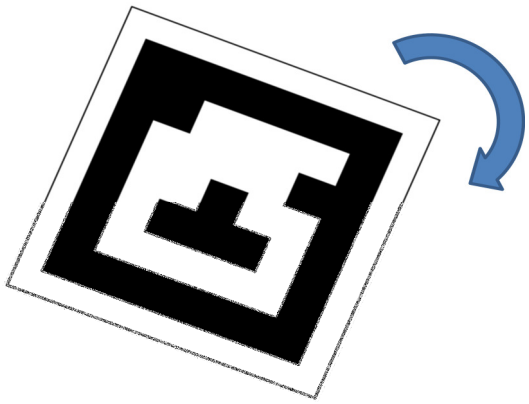


Figura 10 Rotação do objeto em torno do eixo z

O parâmetro a ser determinado é o ângulo que o objeto foi rotacionado; este ângulo pode ser absoluto ou incremental. Para permitir a

determinação do ângulo absoluto de rotação o padrão não pode ser totalmente simétrico.

Rotação do Objeto em Torno dos Eixos x ou y

Os eixos x e y são os eixos coplanares ao plano do objeto que possui o padrão. Essas rotações são limitadas, pois quando estes ângulos aumentam e tendem a 90 graus a visibilidade do padrão no objeto diminui (tende a zero). Os parâmetros a serem determinados são os ângulos do padrão que foi rotacionado em relação a cada eixo. Na Figura 11 podemos observar como essas rotações ocorrem.

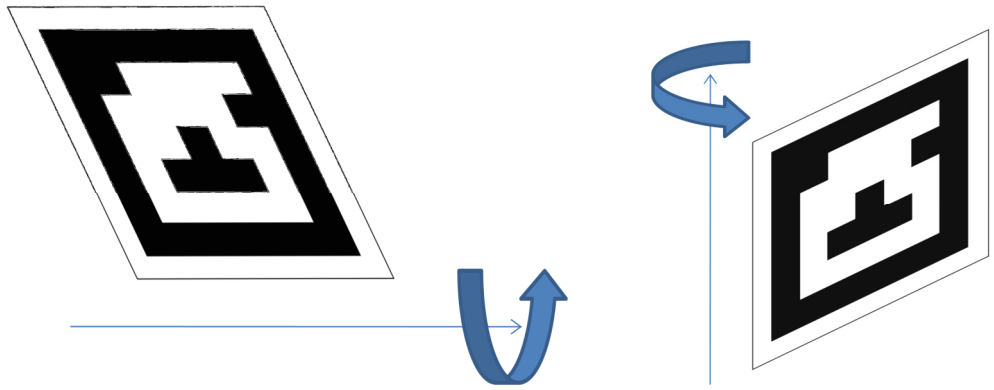


Figura 11 Rotação do objeto em torno do eixo x (a) e em torno do eixo y (b)

Como a aplicação de realidade aumentada desenvolvida neste projeto é uma aplicação em duas dimensões, não serão desenvolvidos algoritmos para encontrar esses parâmetros.

4.1.4 Rotações Múltiplas do Objeto

A rotação múltipla é o resultado da soma da rotação em torno do eixo x, y e z simultaneamente. Este efeito é mais difícil de ser analisado e os parâmetros que determinam a rotação múltipla são os próprios ângulos de rotação em torno do eixo x, y e z.

Como a aplicação de realidade aumentada desenvolvida neste projeto é uma aplicação em duas dimensões, não serão desenvolvidos algoritmos para encontrar esses parâmetros.

4.1.5 Profundidade do Objeto

A profundidade é um dos parâmetros mais difíceis de se determinar. Este parâmetro indica a distância entre a webcam e o objeto com o padrão. Porém como a aplicação de realidade aumentada desenvolvida neste projeto é uma aplicação em duas dimensões, não serão desenvolvidos algoritmos para encontrar esses parâmetros.

4.1.6 Desenvolvimento de algoritmos para a resolução do problema

Neste projeto foram desenvolvidos algoritmos capazes de determinar os parâmetros propostos. Porém o foco não foi chegar de imediato a uma solução ótima para tal tarefa. A intenção foi que ao longo do desenvolvimento desses algoritmos seja possível identificar as suas dificuldades e de alguma forma contorná-las. Porém não precisando ser necessariamente a mais precisa e a mais rápida existente.

São utilizados também algoritmos de visão computacionais já existentes que devem serão adaptados para ajudar na determinação dos parâmetros enunciados anteriormente.

Como objetivo final pretende se chegar a algoritmos que de uma maneira geral realizam a determinação desses parâmetros de forma mais precisa e com tempo de processamento de imagem menor.

4.2 Aplicação Prática

Por fim pretende-se utilizar todos os parâmetros adquiridos para a interação de uma pessoa com o mundo virtual em uma aplicação de realidade aumentada.

4.2.1 Desenvolvimento da Aplicação

Na aplicação, um braço robótico com duas juntas de rotação deve se movimentar até um objeto mostrado na tela, pegá-lo e colocá-lo em um determinado lugar. Com isso, métodos de cinemática direta e inversa devem ser aplicados para que o robô consiga desempenhar essas tarefas.

4.2.2 Etapas

Para o primeiro protótipo, dado um ponto no limite da tela o robô deverá conseguir rotacionar suas juntas de modo que a posição final do seu efetuador seja igual à do ponto passado como parâmetro. Para isso, um método de cinemática inversa deverá ser aplicado e testado.

Como segundo protótipo, o robô deverá partir do ponto anteriormente passado até uma posição já conhecida, o que equivale a levar o objeto para o cesto. Nesse caso, a posição em que se encontra o cesto é fixa, e para evitar cálculos computacionais excessivos, pode-se calcular previamente quais são os ângulos das juntas de rotação e depois passá-los diretamente como parâmetros, para o posicionamento do robô.

Por fim, deve-se unir os métodos de reconhecimento e localização de padrões com os métodos desenvolvidos para essa aplicação fazendo com que o parâmetro de entrada para o método da cinemática inversa do robô seja o parâmetro retornado pelo método de localização do padrão mostrado para webcam, e que o robô reconheça o padrão mostrado e coloque na cesta correta.

5 METODOLOGIA

A etapa de testes dos algoritmos de processamento de imagens foi realizada na linguagem C#. Portanto a metodologia se apoia nesta linguagem para explicar como os algoritmos foram implementados.

O motivo pelo qual a linguagem foi trocada no meio do projeto foi o baixo desempenho da biblioteca de captura de imagem utilizada inicialmente. Ela fornecia poucos quadros por segundo (por volta de 7 quadros por segundo, sem o processamento da imagem). Isso resultou em um baixo grau de aplicabilidade em aplicações de realidade aumentada.

Por isso resolveu-se desenvolver a aplicação em Java, pois uma biblioteca de captura de imagens que foi testada (biblioteca vídeo do Processing) mostrou bons resultados (fornecendo mais de 60 quadros por segundo, sem o processamento da imagem). Esse quesito é muito importante para esse tipo de aplicação, pois fora a captura, muitos outros métodos de processamento de imagem são utilizados e devem ser processados de forma rápida para que o impacto causado na aplicação (que é em tempo real) seja menor. Portanto, todo o código desenvolvido em C# foi adaptado para Java no desenvolvimento da aplicação. A seguir, encontram-se as metodologias utilizadas na linguagem C# (do item 5.1 ao 5.6) e a continuação do desenvolvimento do projeto que foi programado em Java (a partir do item 5.7).

5.1 Trabalhando com Imagem

Na linguagem C# imagens são armazenadas por um objeto da classe Image, porém para instanciar um objeto com uma imagem devemos utilizar um objeto da classe Bitmap que é filha da classe Image. Quando se deseja criar um objeto do tipo Bitmap através de uma imagem que já existe, basta passar o caminho da mesma como parâmetro do construtor dessa classe.

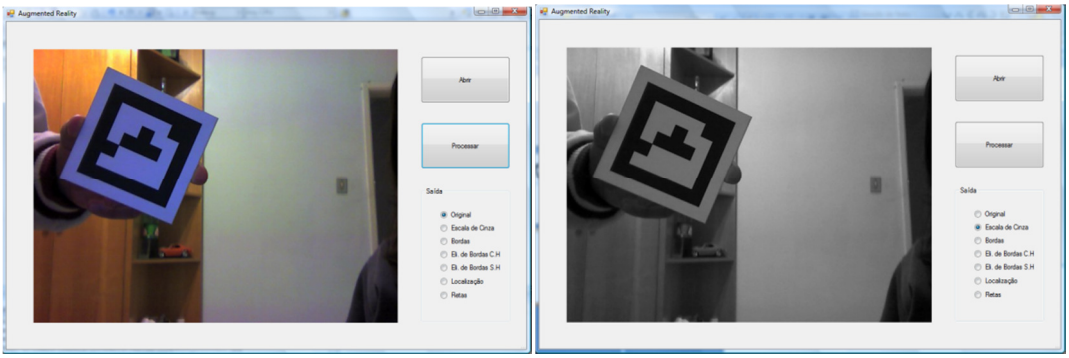
Através de um objeto do tipo Bitmap é possível ter acesso aos pixels da imagem, isso é possível devido aos métodos (Bitmap)b.GetPixel(x,y) e

(Bitmap)b.SetPixel(x,y,color). Quando utilizamos o método GetPixel(x,y) onde x e y são as posições do pixel na imagem e o retorno deste método é um objeto do tipo Color, onde para se obter o valor de cada cor no padrão RGB basta utilizar as propriedades (Color)c.R , (Color)c.G e (Color)c.B. Para alterar um pixel basta utilizar o método SetPixel(x,y,color) onde os parâmetros de entrada são as posições x e y do pixel e a cor do mesmo na forma de um objeto do tipo Color.

Para explicar como a imagem foi manipulada pelos processos que ela passou vale lembrar que uma imagem digital é normalmente representada por um vetor, onde o valor de cada elemento do vetor corresponde à cor de um pixel da imagem. Os padrões mais comuns para representar as cores são o ARGB e o RGB. Como a transparência não é uma característica que nos interessa analisar, vamos trabalhar apenas com o padrão RGB. Porém sabemos que na escala de cinza a intensidade de vermelho é igual ao de verde que é igual ao de azul, portanto para armazenar uma imagem em um vetor na escala de cinza precisamos apenas do que chamaremos de intensidade de cinza. Lembrando que apesar da escala de cinza ter valor igual para as três cores, quando estamos transformando uma imagem colorida em escala de cinza não basta pegar o valor da intensidade de uma delas e igualar com as outras, para se fazer a transformação deve se pegar as intensidades das três cores e somá-las em uma proporção que já foi mostrada na fundamentação teórica.

Ao transformar cada pixel na escala de cinza o seu valor nessa escala foi armazenado em um vetor de bytes. Isso porque no padrão RGB a intensidade das cores vão de 0 a 255 que é exatamente a capacidade de armazenamento do byte. Para mapear a posição dos pixels em um vetor devemos seguir a seguinte estrutura vetorImagem[largura * y + x] onde largura é um valor fixo que corresponde a largura da imagem original, e os valores x e y correspondem a posição do pixel na mesma.

Na Figura 12 podemos observar na imagem da direita a imagem da esquerda na escala de cinza.



a) b)

Figura 12 Imagem em escala de cinza (b) da imagem original (a).

Para a recuperação da imagem através do vetor na escala de cinza foi criado um objeto do tipo Bitmap, porém não através do caminho de uma imagem existente, mas sim através do construtor que recebe como parâmetro de entrada o tamanho da imagem (largura e altura) e o formato do pixel. O próximo passo é preencher a imagem com a cor dos pixels através do método já citado SetPixel(x,y,color). Para formar a cor do objeto do tipo Color pode se usar o método estático dessa classe que é o Color.FromArgb(valor, valor, valor) onde valor é a intensidade de cinza da imagem.

Para mostrar a Imagem no monitor foi implementado uma interface gráfica utilizando o Windows Form do Visual Studio. Uma instância da classe PictureBox é encarregada de apresentar a figura. Ela recebe a imagem através da propriedade (PictureBox)p.Image, como normalmente estamos trabalhando com objetos do tipo Bitmap a única coisa que temos que fazer é realizar um cast de um objeto do tipo Bitmap para um do tipo Image.

Na Figura 13 podemos observar a Interface gráfica implementada.

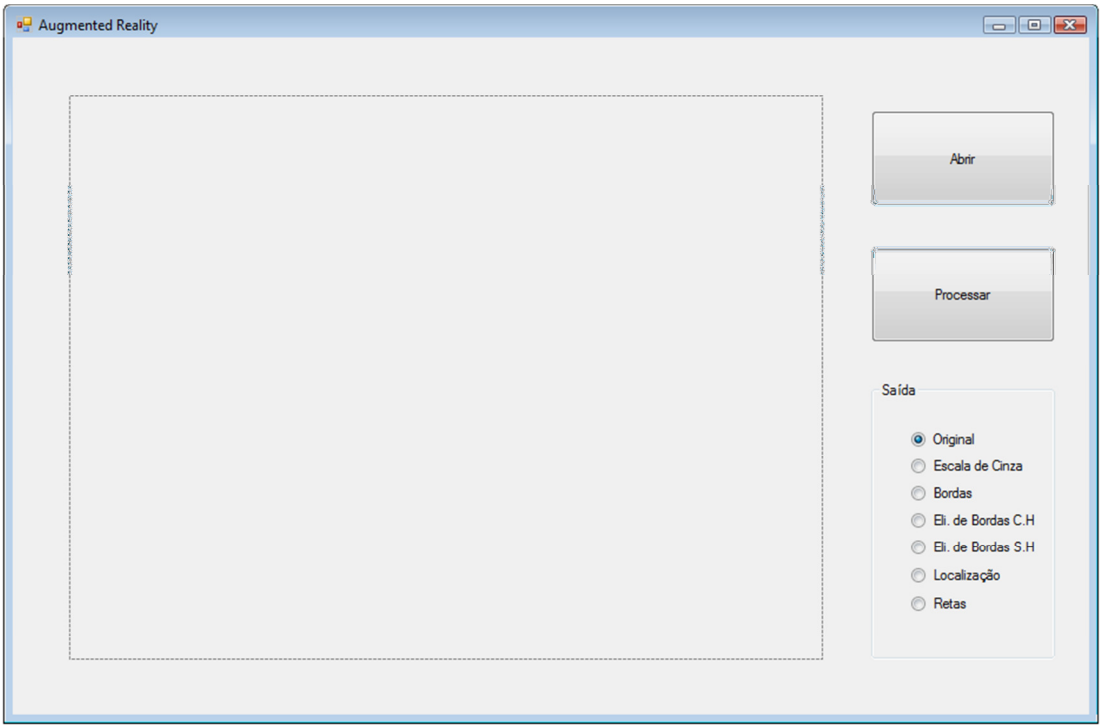


Figura 13 Interface gráfica para visualização das imagens

5.2 Detecção de Borda

O método de detecção de borda que foi implementado foi o método de Sobel. Este método recebe o vetor da imagem em escala de cinza e retorna a mesma também em escala de cinza, porém apenas com as bordas da imagem. Como neste método o gradiente pode ultrapassar o valor 255 foi utilizado um vetor de inteiros para receber o resultado da magnitude dos gradientes e após a normalização os valores foram novamente armazenados em um vetor de bytes.

A Figura 14 ilustra o resultado depois de aplicado o método de detecção de borda na Figura 12-b).

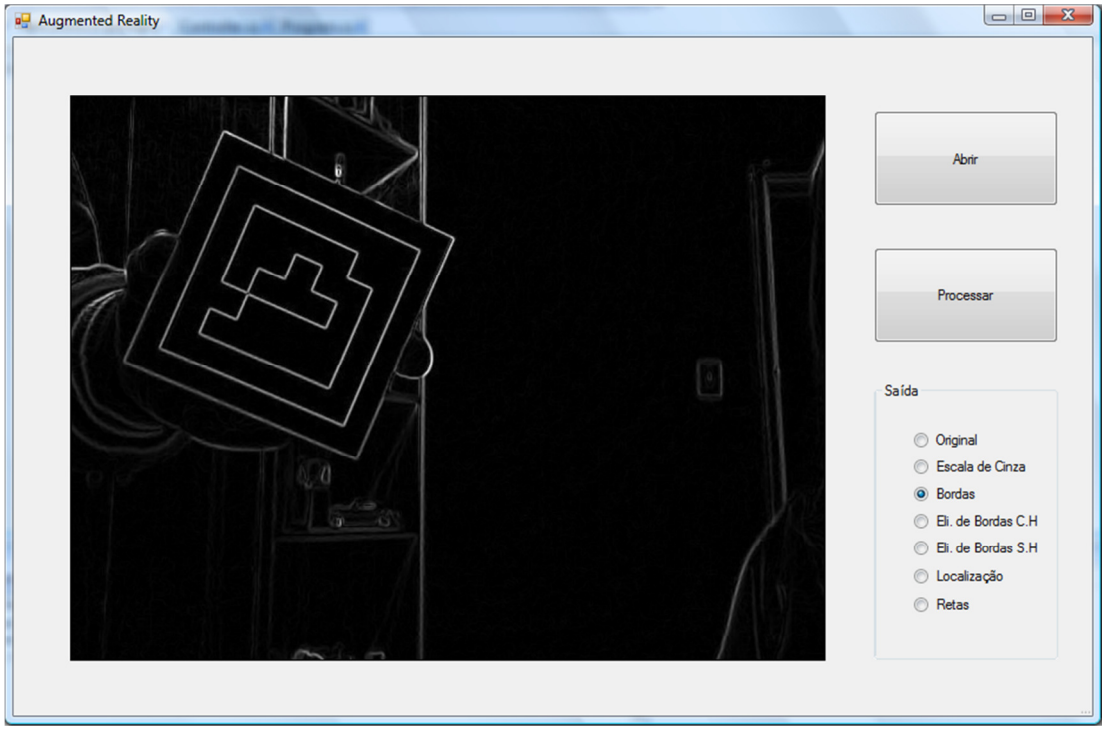
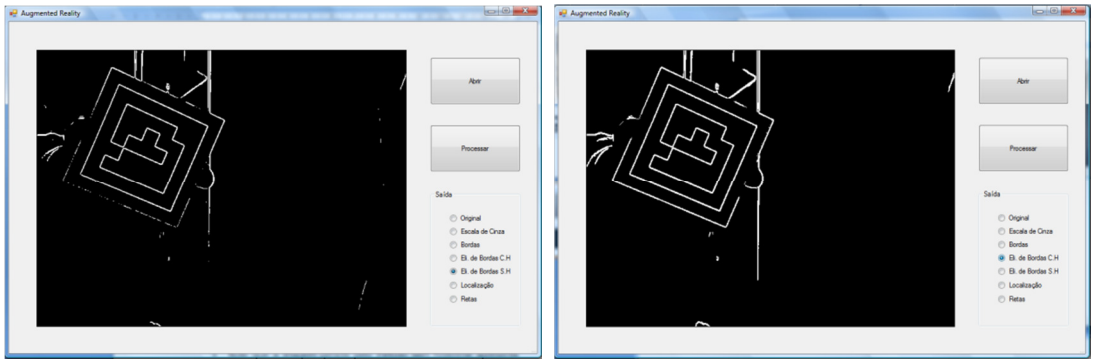


Figura 14 Imagem após aplicação do método de detecção de borda

5.3 Eliminação de Bordas Fracas

Dois métodos de eliminação de bordas foram aplicados, um com histerese e outro sem. Ambos recebem o vetor imagem na escala de cinza e retorna o mesmo em preto e branco, isso porque as bordas fortes recebem o valor máximo de 255 que corresponde ao branco e as fracas recebem o valor mínimo de 0 que corresponde ao preto.

Na Figura 15 podemos observar a esquerda o método de eliminação de bordas fracas sem histerese e a direita com histerese.



a) b)

Figura 15 Imagem resultante sem histerese (a) e com histerese (b)

Note que a imagem gerada pelo método sem histerese apresenta rupturas nas regiões que representam a mesma borda, porém isso não é um problema tão grande para o que desejamos observar na imagem, uma vez que a borda gerada pelo quadrado (objeto que se pretende localizar) é predominante na imagem.

5.4 Localização

O método de localização desenvolvido neste projeto foi elaborado com base na observação das imagens de saída do método de identificação de bordas com eliminação das bordas fracas. Notou-se que o objeto que se deseja localizar sobressai muito na imagem e é um dos poucos que possui bordas que são cíclicas, ou seja, que forma uma região fechada como pode se observar na Figura 16.

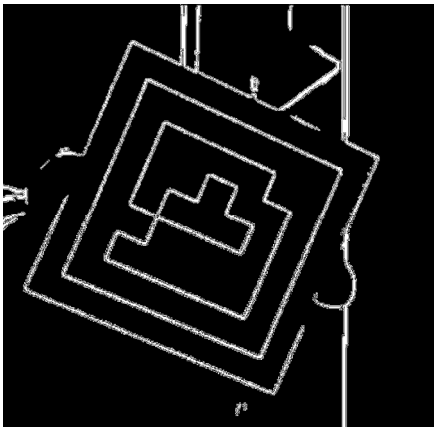


Figura 16 Destaque para borda do quadrado que forma uma região fechada

Para encontrar as bordas cíclicas varremos a imagem inteira atrás dos pixels brancos e a partir do pixel branco encontrado tentamos formar um ciclo através dos pixels branco adjacentes. Porém esse processo não precisa ser realizado para todos os pixels brancos, pelo fato de muitos já terem participado de testes anteriores. Portanto antes de começar a testar se a partir daquele pixel irá ser possível fechar um ciclo é feito a checagem se o mesmo não já participou de um teste anterior. Com isso eliminamos uma quantidade enorme de testes desnecessários ou que dariam o mesmo resultado.

O critério para eliminar o pixel do teste é simples: se ele possui um pixel branco na linha de cima próximo (checagem de 5 pixels na linha de cima) a ele, a chance dele já ter sido testado é grande e portanto ele é eliminado, isso também acontece quando um pixel possui um pixel branco antes e próximo a ele. Na Figura 17 podemos observar os casos em que o pixel é eliminado para o teste e quando ele é aceito.

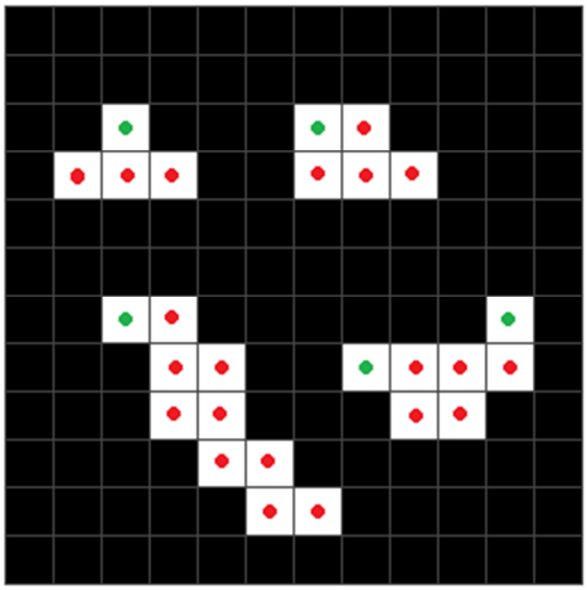


Figura 17 Bolinhas verdes representam pixels aceitos e bolinhas vermelhas representam pixels eliminados

Agora com os pixels que forem aceitos começaremos o processo em tentar achar uma figura com bordas fechadas. Porém a figura que queremos identificar possui características peculiares, facilitando esse processo: uma é que se trata de um quadrado, portanto as bordas que correspondem às arestas são retas. Assim como começamos do pixel superior que por sua vez representará um dos supostos vértices do quadrado, tentamos descer na imagem a fim de chegar ao próximo vértice. Na Figura 18 podemos observar o que foi explicado anteriormente.

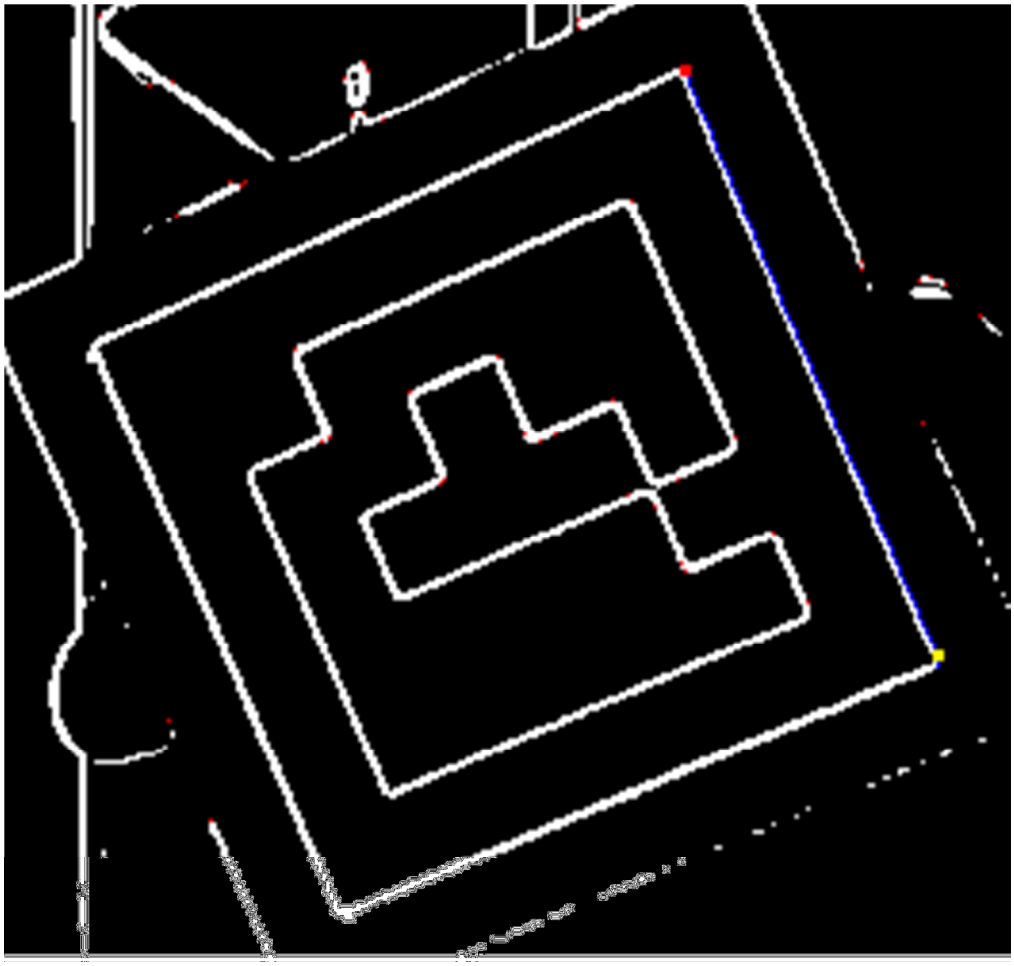


Figura 18 Os pontos vermelhos são os possíveis primeiros vértices e a linha azul representa o caminho percorrido pelo programa para chegar ao segundo vértice (em amarelo)

Na Figura 18 começamos a contornar a primeira aresta do quadrado no sentido horário. A lógica utilizada foi que se deve checar sempre se o pixel da direita é branco ou não. Quando for branco, passamos para o próximo pixel. Se for preto checamos qual a cor do pixel abaixo. Se for preto chegamos no fim desta aresta, porém se for branco continuamos a checar a cor do pixel a direita. Se for branco continuamos e a lógica volta a ser a inicial; porém, se for preto, deveríamos concluir que chegamos em um vértice e o código estaria terminado. Porém pelo fato da reta da aresta não ser perfeita as imperfeições podem ser contornadas fazendo novos testes para baixo e novamente para a direita. Esses testes são repedidos por três vezes se mesmo assim ele não achar um pixel a direita para continuar sua verificação ele volta até a última vez que não conseguiu ir mais para a direita e considera este ponto o vértice.

Para as próximas arestas a lógica é a mesma, mas a direção em que as checagens ocorrerão é diferente. Porém entre uma aresta e outra o processo só continua se a aresta anterior tiver um número mínimo de pixels, assim eliminamos regiões extremamente pequenas que provavelmente não é a que desejamos localizar (com exceção de quando o objeto que desejamos localizar estiver muito longe) e economizamos tempo de processamento.

Este método se mostrou muito eficiente em alguns testes realizados, porém foi encontrada uma posição do quadrado em que o mesmo falhou. Esta posição é apresentada na Figura 19.

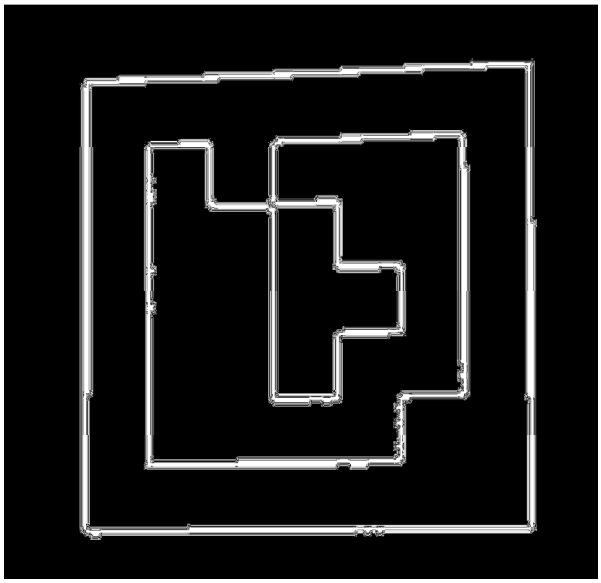


Figura 19 Posição em que ocorre falha na varredura no sentido horário

Este problema foi solucionado aplicando o método tanto no sentido horário quanto no sentido anti-horário. Quando em um sentido o algoritmo não encontra um caminho adequado para se propagar, ele tenta no outro sentido. Este método não garante que irá encontrar apenas a região do quadrado, outras regiões podem ser encontradas ao aplicar este método, por isso ele não é sozinho suficiente para identificar unicamente a região de interesse. No caso acima, apenas uma região de interesse foi encontrada, e com os vértices dessa região foram traçadas retas que ligam esses pontos na imagem original, como visto na Figura 20.

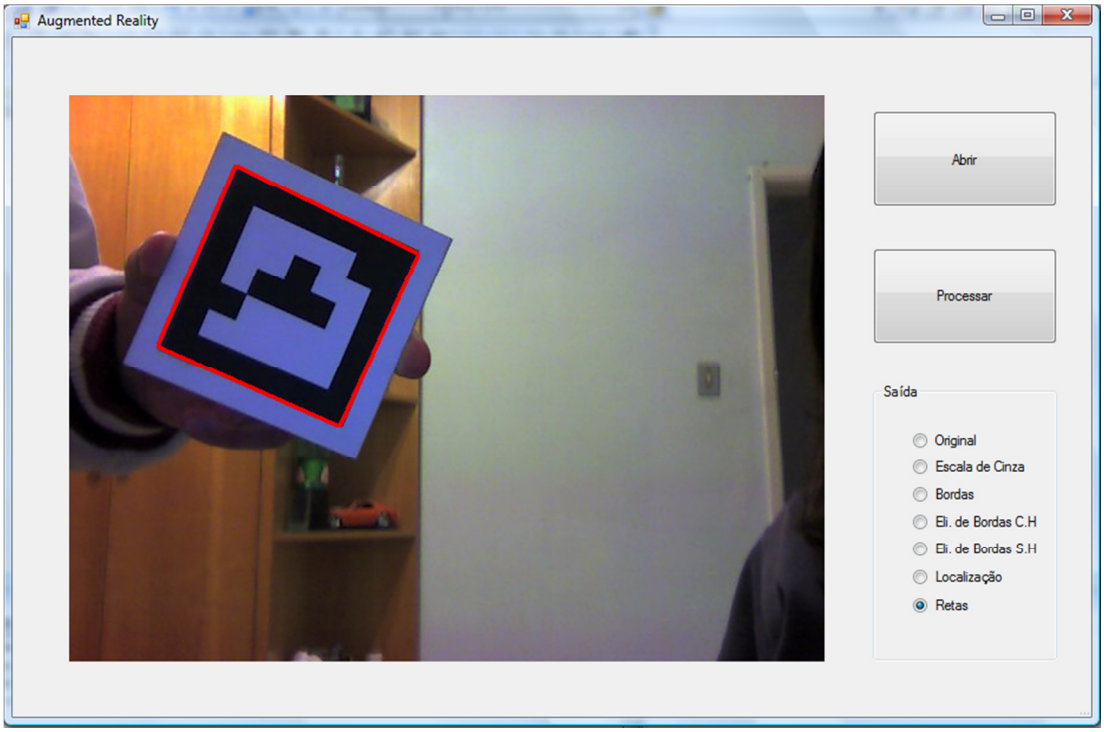


Figura 20 Imagem que representa a localização do objeto com o padrão

5.5 Captura de Imagens

Originalmente os métodos foram aplicados para imagens estáticas. Os mesmos foram adaptados para captura dinâmica de imagens através de uma webcam. Para realizar a captura de imagem foi utilizada uma classe chamada WebCam_Capture que realiza a conexão com a webcam e retorna um objeto do tipo Image de tempos em tempos através de um evento. Essa imagem é tratada e depois é retornada para a interface gráfica. A classe WebCam_Capture foi importada do Visual Basic 6 e pode ser encontrada na referência [8].

5.6 Realização de Testes de desempenho dos algoritmos

Para a realização dos testes foi criada uma classe chamada Cronometro que possui um método chamado Start(). Este método grava o

valor do relógio do computador em uma variável que é correspondente ao valor do início da cronometragem do tempo; o método Stop() por sua vez grava o valor do término. Pra saber o tempo cronometrado basta chamar o método TempoTotalMS().

5.7 Reconhecimento do padrão impresso no marcador

Após a localização das regiões de interesse realizado pelo processo descrito no item 5.4, deve-se eliminar as regiões que não contém um marcador e analisar as regiões que contém.

Para realizar essa tarefa utilizamos dois vetores (Vx e Vy) obtidos através dos vértices das regiões de interesse. Observe na Figura 21 a representação dos vetores Vx e Vy na imagem de um marcador.

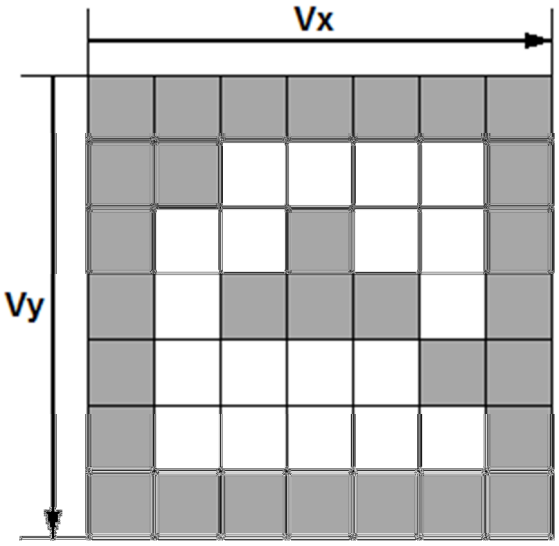
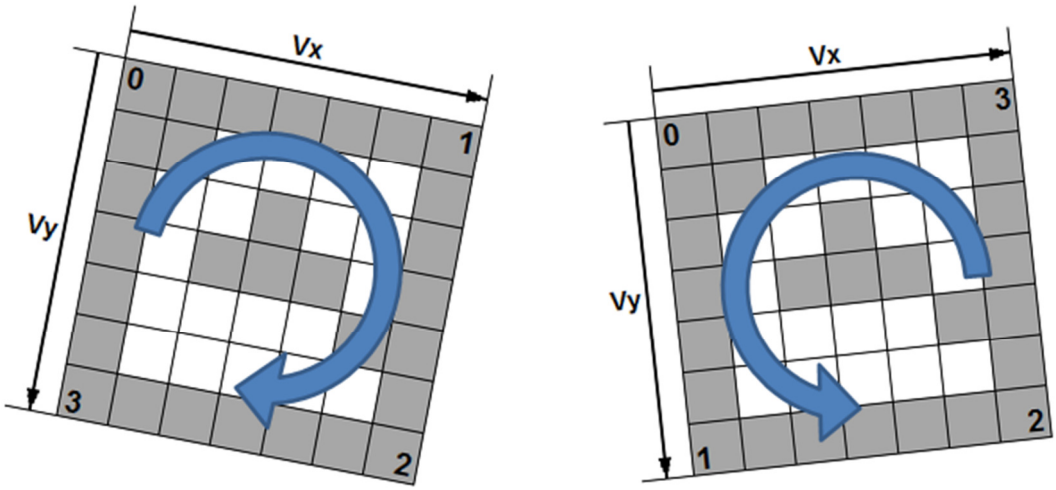


Figura 21 Representação dos vetores Vx e Vy no marcador

Quando a região de interesse for localizada, percorrendo-a no sentido horário os vértices que formam o vetor Vx são os vértices 0 e 1. Já os que formam o Vy são os vértices 3 e 0. Porém quando a região de interesse é localizada percorrendo a no sentido anti-horário os vértices que formam Vx e Vy são trocados. Isso pode ser observado na Figura 22.



a) b)

Figura 22 Formação do Vx e Vy através dos vértices quando a região é encontrada no sentido horário a) e no sentido anti-horário b)

O primeiro teste a ser feito é tentar identificar se apenas um dos cantos da região interna do marcador é preto (quadrado diferenciador). Esta operação pode ser observada na Figura 23 e consiste em combinar as proporções 3/14 e 11/14 dos vetores Vx e Vy. Se essa condição for verdadeira prosseguimos mapeando todos os quadrados internos do marcador, mas se ela for falsa, descartamos essa região.

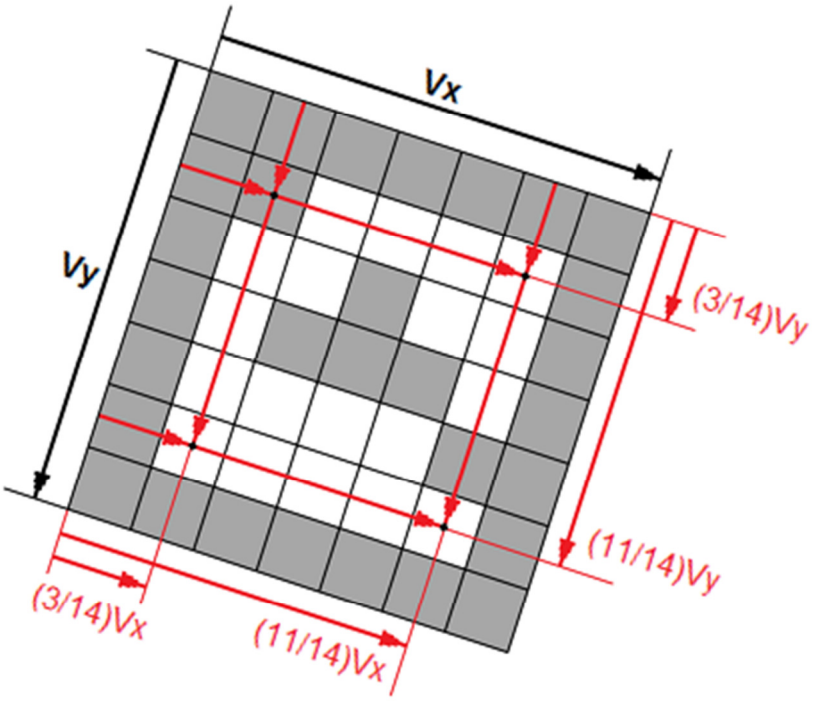


Figura 23 Procedimento para encontrar o quadrado diferenciador

O mapeamento do marcador é feito amostrando o ponto central de cada quadrado, combinando Vx e Vy para atingir esse objetivo. Esses valores são guardados em um array de variáveis booleanas. Este array é comparado com valores previamente registrados, se ele for igual á um desses registros, o marcador é considerado conhecido e recebe a identificação do mesmo.

Porém antes de fazer essa comparação, devemos rotacionar o array para que fique na mesma posição que os registros foram gravados, pois o marcador pode estar rotacionado de 90, 180 ou 270 graus. Assim não precisamos registrar 4 posições diferentes para cada marcador registrado.

Para saber quantas vezes devemos rotacionar o array para ficar igual ao registro, utiliza-se a informação da posição do quadrado diferenciador do marcador.

5.8 Localização do centro do Marcador

Depois que identificamos a região de interesse como sendo um marcador conhecido, achar a posição central do marcador é simples: Basta somar a posição do vértice 0 com 7*Vx/14 e 7*Vy/14. Isso pode ser observado na Figura 24

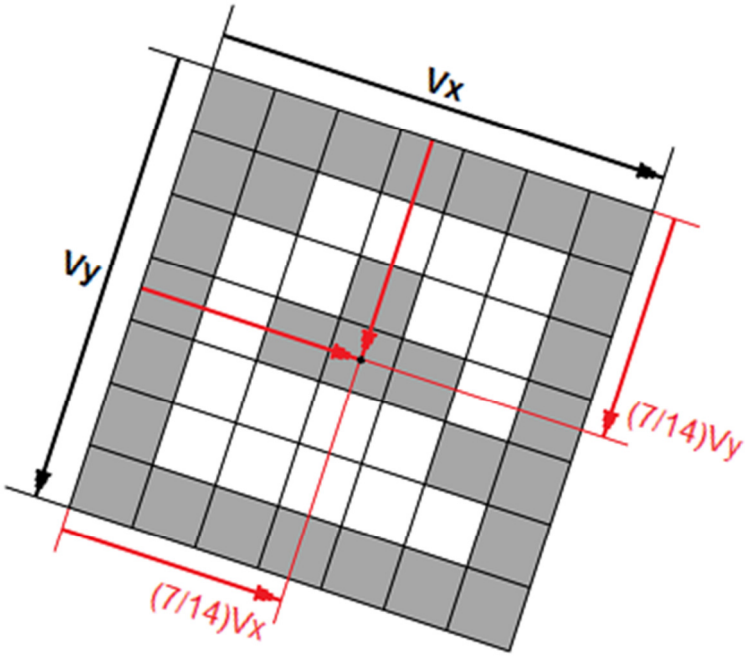


Figura 24 Localização do centro do marcador

5.9 Cálculo do ângulo do marcador em relação ao eixo Z

Para encontrar o ângulo de rotação do marcador em relação ao eixo Z utilizam-se duas informações, o ângulo entre Vx e a horizontal e a posição do quadrado diferenciador.

A posição do quadrado diferenciador indica se houve uma rotação completa de 90, 180 ou 270 graus. E o ângulo entre o Vx e a horizontal indica quantos graus devem ser adicionados a esta rotação.

5.10 Contornando problemas com a não localização

Quando movimentamos o marcador rapidamente a imagem do mesmo fica borrada e os algoritmos de localização não são capazes de localizá-los.

Para que a imagem do resíduo não ficasse piscando, ou seja, aparecendo e desaparecendo na tela toda vez que o marcador não fosse localizado, foi programado um sistema que aguarda um tempo até dizer que o marcador realmente não foi localizado.

Esse tempo pode ser modificado conforme a necessidade da aplicação.

5.11 Optimização do pré-processamento da imagem

Em uma tentativa de melhorar o desempenho (numero de quadros por segundo) da aplicação eliminamos alguns processos de tratamento de imagens (processos 5.2 e 5.3). Portanto a imagem é transformada em escala de cinza e depois diretamente em preto e branco. Por fim realiza-se a busca dos marcadores através do processo 5.4.

Isso resultou em uma eficácia maior na localização dos marcadores e um ganho elevado no desempenho geral da aplicação (aumento na quantidade de quadros por segundo).

5.12 Aplicação

5.12.1 Estrutura da aplicação

Como já foi dito a aplicação foi desenvolvida em Java utilizando as bibliotecas do Processing com o auxilio da IDE Netbeans.

O software ficou dividido em 6 classes que são:

- Processing
- ImageProcessor
- Robot
- Pattern
- TimeManager
- Vector2i

Apesar das bibliotecas do Java fornecer classes para trabalhar com vetores (Vector2, Vector3, etc), elas são voltadas para trabalhar com números do tipo float, como se desejava trabalhar com inteiros foi criado uma classe Vector2i.

A classe TimeManager foi criada com o intuito de gerenciar o tempo na aplicação e para a realização de testes de desempenho da mesma.

A classe Pattern possui todos os atributos para descrever os estados do marcador (posição, ângulo, visibilidade, vértices, id, etc).

Na classe Robot é realizado toda a parte do posicionamento do robô através do cálculo da cinemática inversa e o posicionamento do resíduo. Que quando é pego pelo robô é realizado através do cálculo da cinemática direta. Ela também contém o código responsável por desenhar tanto o robô quanto o resíduo na janela do aplicativo.

A classe ImageProcessor é responsável pela parte de aquisição das imagens e pelo processamento das mesmas para localizar os marcadores. Após encontrar os marcadores, ela retira todas as informações necessárias do mesmo, colocando as em um objeto do tipo pattern. Depois ela coloca esses objetos em uma lista que poderá ser utilizada por outras classes.

A classe Processing possui o main do projeto e é estruturada da mesma forma que no Processing (Programa). Nela chamamos os métodos das classes ImageProcessor e Robot responsáveis por processar as entradas e saídas da aplicação.

5.12.2 Proposta da Aplicação

Gostaríamos de acrescentar algo do curso de mecatrônica na aplicação desenvolvida, e foi por isso que a escolha de um braço robótico

virtual foi feita. A proposta também é educacional de certa forma, pois mostra de uma forma simples, divertida e interativa a maneira correta de se reciclar o lixo.

Sendo assim, a aplicação de RA feita neste trabalho consiste em um robô virtual que deve identificar o resíduo mostrado (através do padrão filmado pela webcam) e despejá-lo na lixeira correta. A Figura 25 mostra a relação dos marcadores com cada tipo de lixo e, conseqüentemente, com cada tipo de lixeira.

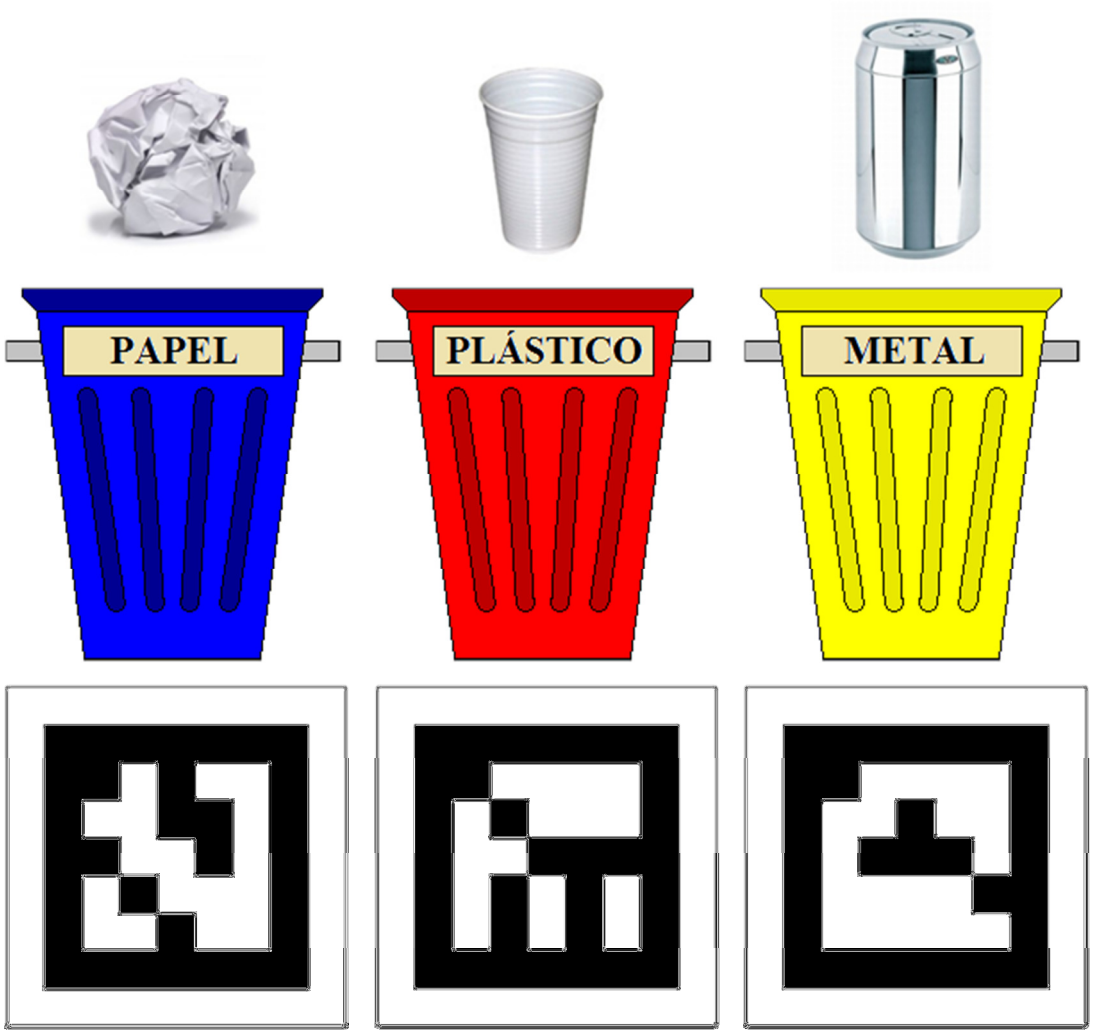


Figura 25 Os resíduos e seus respectivos padrões e lixeiras.

5.12.3 Desenvolvimento do Protótipo

O protótipo da aplicação foi desenvolvido em Java no programa de prototipagem Processing, para posteriormente ser passado para o ambiente de desenvolvimento Netbeans usando as bibliotecas do Processing.

Nesse protótipo usa-se cinemática inversa para que o robô RR atinja as coordenadas dadas pelo cursor do mouse. No anexo encontra-se o código fonte utilizado. Na Figura 26 pode-se observar o resultado visual desse protótipo.

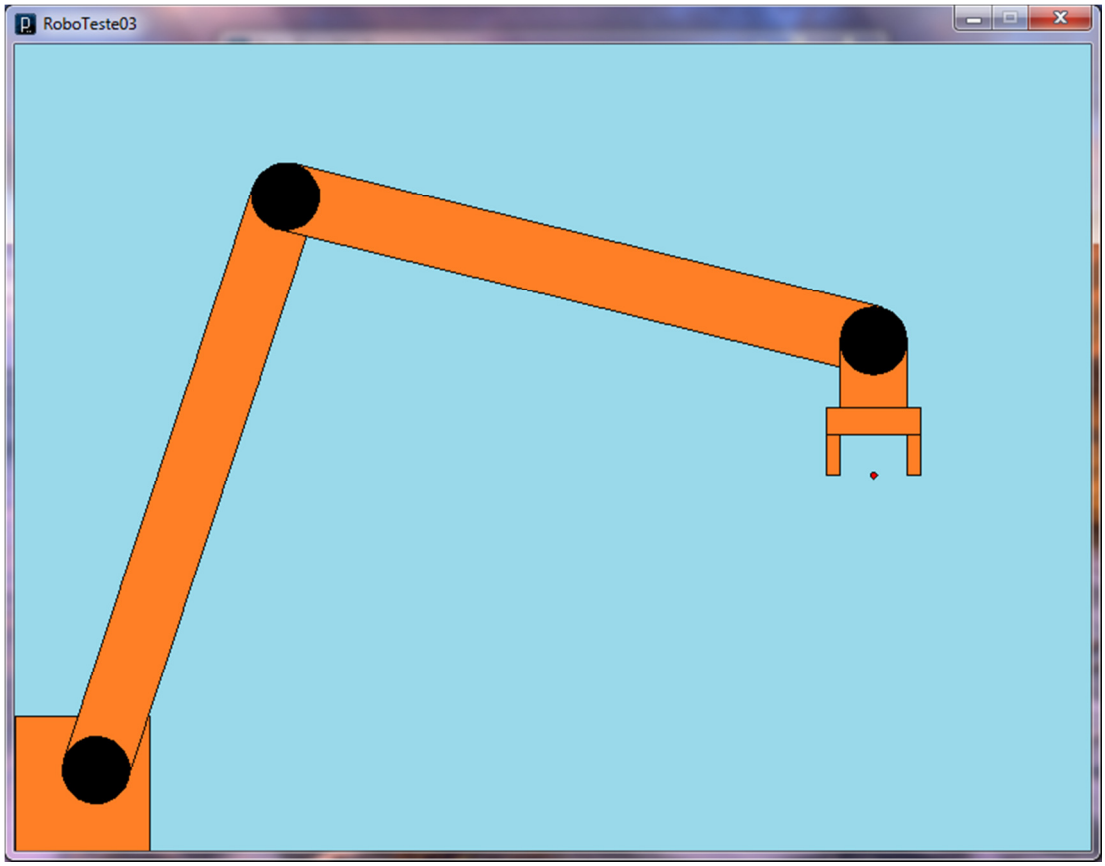


Figura 26 Imagem do protótipo da aplicação feito no Processing

A seguir foram adicionadas imagens mais realistas de um robô, para que a aplicação se tornasse mais interessante. A Figura 27 mostra o resultado dessa mudança. Também foi adicionado um círculo verde que marca o ponto do posicionamento de uma das lixeiras para onde o robô deverá se direcionar. Até então a aplicação funcionava da seguinte forma: o ponto vermelho marcava a próxima posição para onde o robô deveria se

movimentar. Este começava em sua posição inicial e assim que um ponto da tela fosse clicado ele se dirigia a esse ponto; em seguida se movia para o ponto representativo da lixeira e por fim retornava a sua posição inicial. A princípio esses movimentos eram instantâneos, mas depois um controle de trajetória foi adicionado, tornando a movimentação mais parecida com a realidade.

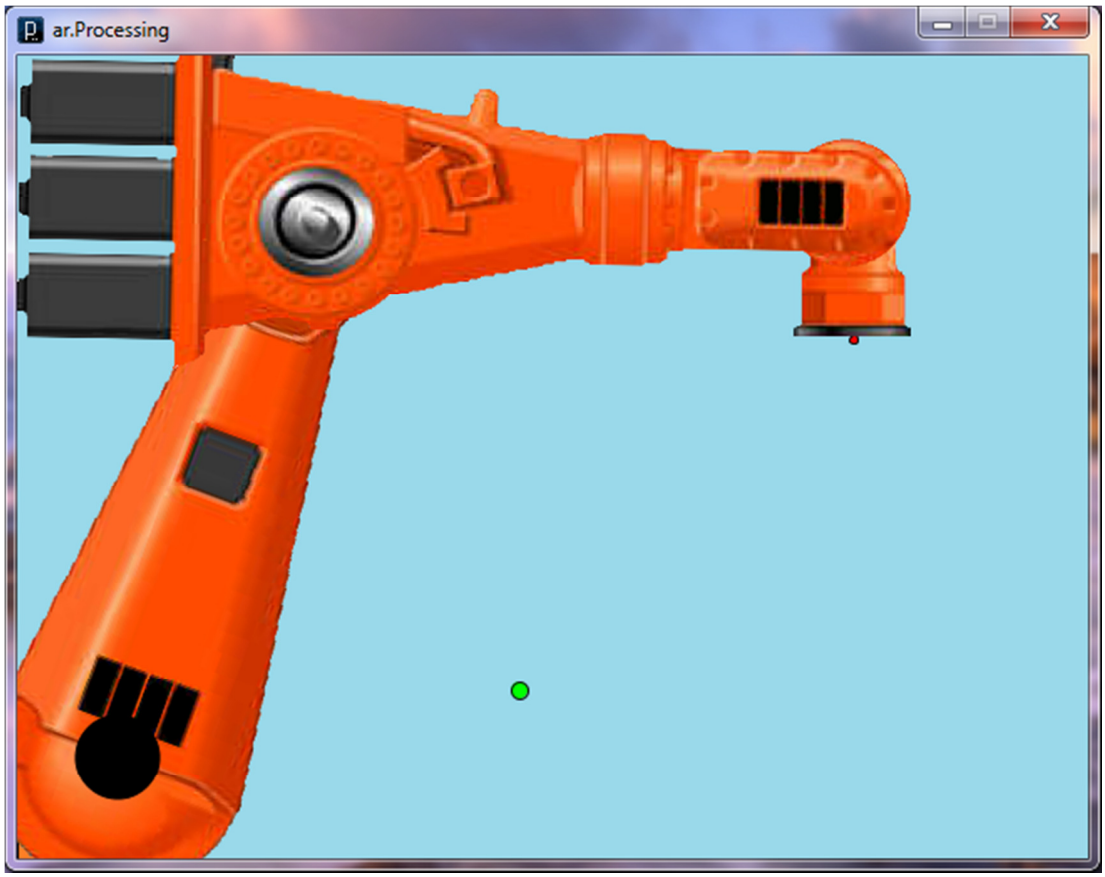


Figura 27 Imagem do protótipo com imagens mais realistas.

Nota-se que para se criar essa imagem, imagens de cada braço foram adicionadas em posições determinadas para se criar a impressão de que elas não estão simplesmente sobrepostas, mas sim conectadas por uma junta rotativa. Isso não é tão simples, pois para cada imagem deve-se mapear os pontos importantes para o correto posicionamento na aplicação Na Figura 28 pode-se observar uma vista explodida do robô com todas as suas partes mais significativas para essa aplicação. A Figura 29 mostra o resultado da montagem das peças.

Figura 28 Vista explodida que mostra as principais peças do robô para esta aplicação.

Figura 29 Imagem do robô montado.

O próximo passo foi a introdução de um quadrado verde (que representa o objeto que substituirá o padrão). Este se movimentava de acordo com a movimentação do mouse. Quando o botão esquerdo era apertado, o quadrado verde deveria permanecer parado e o robô começaria a se movimentar em sua direção. Chegando no objeto o robô "pegaria" este e o levaria até a lixeira correta, indicada por circunferências coloridas (como pode ser observado na Figura 30).

Neste protótipo, os pontos azul, vermelho e amarelo representam respectivamente as lixeiras de papel, plástico e metal. Sendo assim, definido no programa que o quadrado verde representaria uma latinha de refrigerante, por exemplo, o robô a pegaria e a levaria até o ponto amarelo. Ao chegar neste ponto, o quadrado verde se soltaria do robô e cairia em direção à lixeira, enquanto o robô retorna a sua posição inicial.

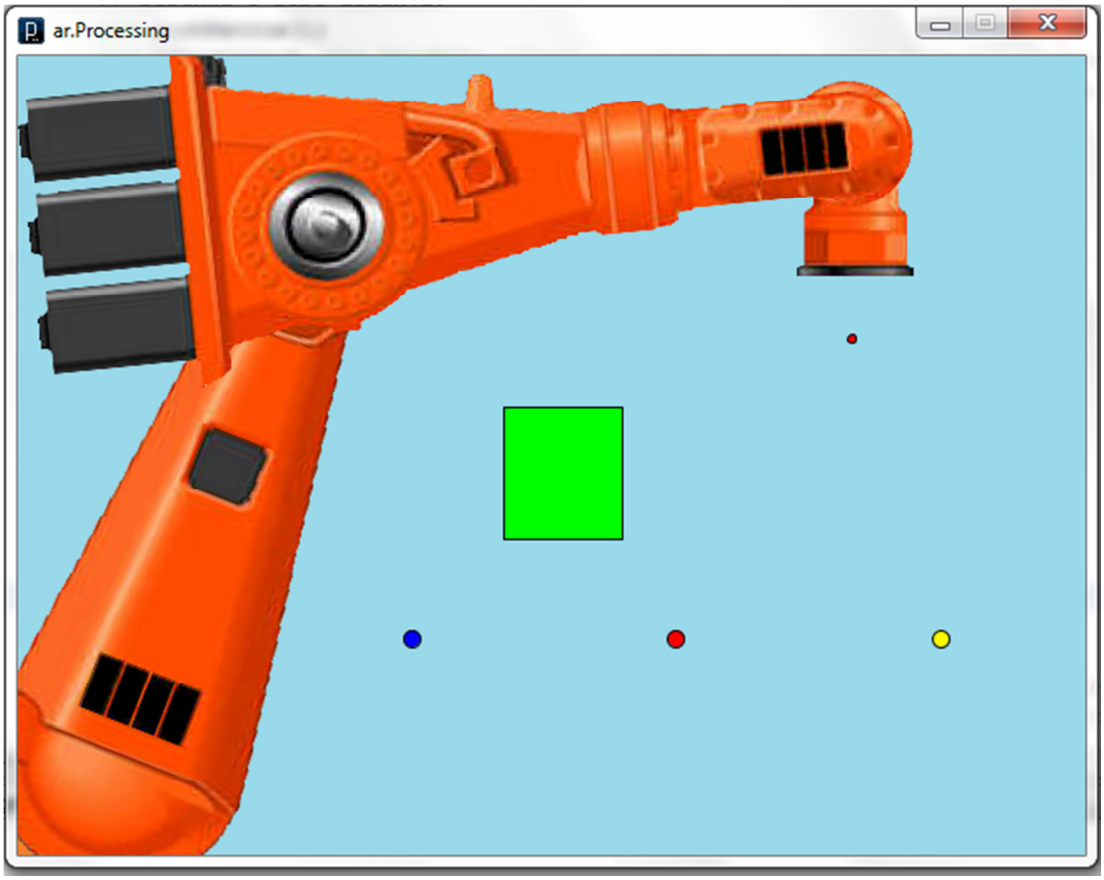


Figura 30 Imagem do protótipo com um objeto representado pelo quadrado verde.

O passo final do protótipo foi substituir o quadrado verde por figuras de resíduos e adicionar lixeiras no lugar dos pontos que as representavam (ver Figura 31). Do passo anterior para este, pequenas mudanças foram feitas, mas nenhuma que alterasse o funcionamento do protótipo.

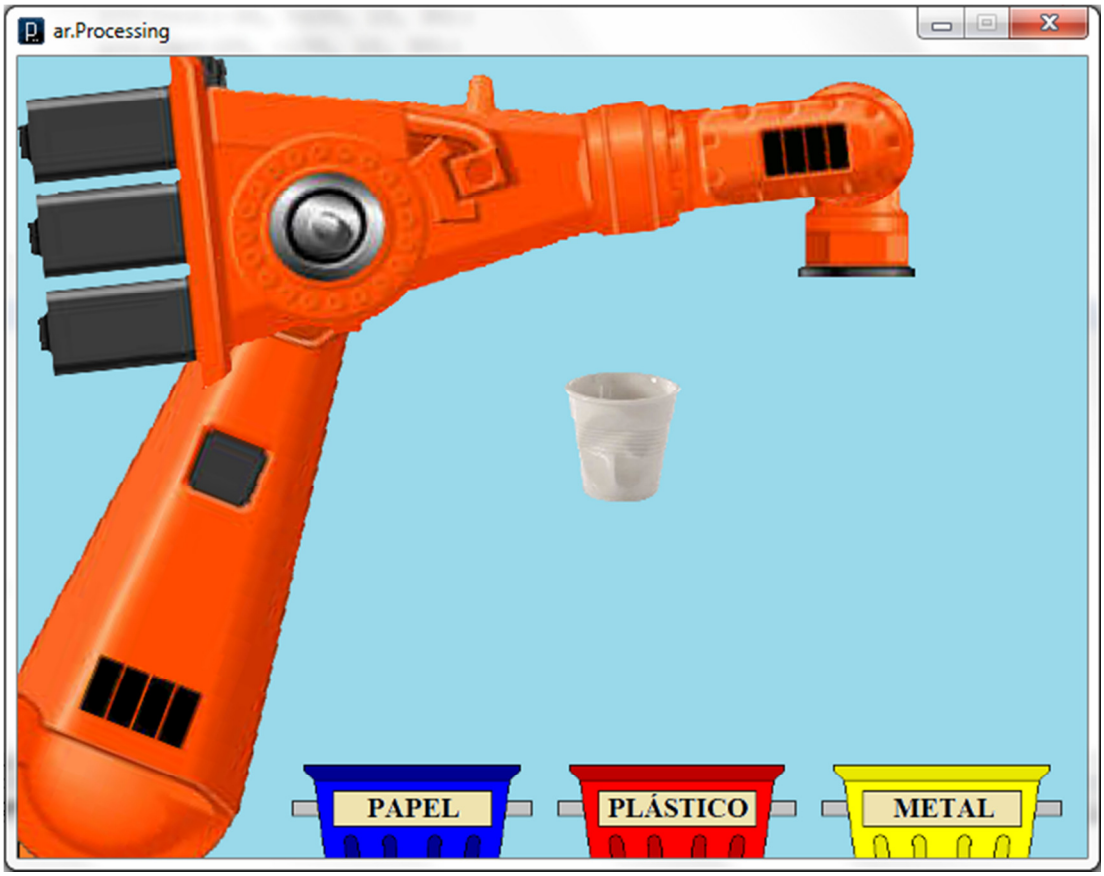


Figura 31 Imagem que mostra o formato final do protótipo.

Do protótipo para a aplicação em sua forma final, foram feitas as compatibilizações dos métodos de visão computacional com os algoritmos usados para desenvolver a parte virtual da aplicação. No item a seguir esse processo é descrito.

5.12.4 União do protótipo da aplicação e do programa de RA

Com o programa de RA desenvolvido é possível localizar e identificar um padrão. Passando-se o centro desse padrão como coordenada de destino para o robô do protótipo, pode-se aprimorar a aplicação de forma

que pareça que o robô vai pegar o objeto mostrado. Feito isso, o próximo passo foi fazer com que o robô deixasse o objeto (identificado pelo seu padrão) na cesta correta. A Figura 32 abaixo mostra a aplicação virtual desenvolvida, com o vídeo capturado pela webcam como background. A etapa final (substituição do padrão pela imagem do dejeto) é descrita no item a seguir.

Figura 32 União do protótipo com o programa de RA desenvolvido.

5.12.5 Finalização da aplicação

A etapa final é fazer com que o padrão mostrado para webcam seja substituído pela imagem de um tipo de material reciclável (bola de papel, latinha de refrigerante, copo plástico, etc...). Assim, o robô deverá perceber que um objeto está sendo mostrado na tela, se movimentar em direção a ele, agarrá-lo e despejá-lo na cesta de lixo correta (papel, metal, vidro, etc...).

A Figura 33 abaixo mostra a seqüência de eventos da interação entro o usuário no mundo real e o robô no mundo virtual.

Figura 33 Seqüência de fotos que mostram o funcionamento da aplicação desenvolvida.

Na forma final da aplicação, os eventos se passam de acordo com a seguinte ordem:

- O robô virtual começa em sua posição inicial, e continua lá até que o sinal de “ok” seja dado;
- Em seguida, o usuário mostra um dos padrões que será substituído pela respectiva imagem de resíduo (no exemplo é o marcador que corresponde aos metais). A imagem imita os movimentos de deslocamento e rotação em torno do eixo z (que “sai” da tela) feitos pelo usuário;
- Ao mostrar o marcador de “ok”, o robô começa a se direcionar ao resíduo para agarrá-lo. Ele continua fazendo perseguindo o objeto mesmo se durante sua trajetória o usuário tentar escapar dele;
- Quando o robô alcança o resíduo, ele o agarra e se dirige para a lixeira correta (que no caso do exemplo é a amarela);
- Ao chegar na lixeira correta, o robô solta o objeto para que este caia dentro do lixo;

- Por fim, depois de liberar o resíduo o robô retorna para sua posição inicial.

  O mesmo processo se passa para os diferentes tipos de resíduos.

6 Resultados

6.1 Resultados Quantitativos

Os resultados quantitativos foram sensíveis à mudança na resolução da captura de imagem, ou seja, quanto maior a resolução (maior número de pixels por imagem) maior é a quantidade de dados a ser tratado e maior o tempo levado para os algoritmos processarem as imagens. Portanto a critério de quantificação dos resultados é necessário especificar que a resolução básica utilizada foi de 640x480. Serão realizadas algumas comparações de desempenho com a resolução 800x600.

É importante frisar também que desempenhos relacionados com velocidade dos algoritmos também dependem diretamente da velocidade do processador do computador em que o software será utilizado. O processador do computador em que foi realizado esses testes de desempenho foi um Core 2 Duo 2.4 Ghz.

6.1.1 Quanto à capacidade de localizar regiões que contenham o objeto de interesse

- A distância mínima, entre a webcam e o marcador, que os algoritmos de localização conseguem encontra-lo é de 10 cm. Já a distância máxima é de 180 cm. Esses valores dependem diretamente tanto da resolução de captura da imagem quanto do tamanho do marcador. A resolução utilizada (como já foi dito acima) foi 640x480 e o tamanho do marcador foi de 76 mm de lado.

- Toda rotação realizada no marcador em relação ao eixo z (eixo perpendicular ao plano do objeto) é totalmente permitida, ou seja, não impossibilitam o método de localização de encontrar a posição em que o marcador se encontra. Este ângulo é determinado com uma precisão satisfatória.

Porém rotações em torno do eixo x ou y (eixos coplanares ao monitor) são limitadas e dependem da distância entre webcam e o marcador e também do tamanho do mesmo.

- O tempo médio gasto pelo o método de localização é de 3,5 ms para a resolução 640x480 e 5,5 ms para a resolução 800x600.

6.1.2 Quanto aos resultados obtidos nos outros algoritmos

A tabela 2 apresenta o tempo médio de cada algoritmo aplicado nas imagens.

Tabela 2 Tempo médio dos processos realizados nas imagens

	Tempo médio nas resoluções:	
Processos realizados nas imagens:	640x480	800x600
Obtenção da imagem via webcam	2 ms	3ms
Ler e copiar em um array	4 ms	7 ms
Transformar em escala de cinza e copiá-la em um array	5,5 ms	9,5 ms
Detecção de borda (Sobel)	61 ms	95 ms
Remoção de bordas fracas com hiterese	4,5 ms	6,5 ms
Transformar em preto e branco	2 ms	3,5 ms
Transformar o array em imagem	4 ms	6,5 ms
Desenhar a imagem	3 ms	5 ms

- O tempo total de todos os métodos de tratamento de imagens incluindo o método de localização das regiões de interesse é de 24 ms para a resolução 640x480 e 40 ms para a resolução 800x600 (Não considerando os métodos Sobel e HystThresh que removidos da aplicação final por motivos de optimização).

- O tempo médio para realizar todos os cálculos relativos ao posicionamento do robô, do resíduo e para desenhá-los é de 6,5 ms.

- O número de quadros (imagens) por segundo (frames per seconds) antes do tratamento de imagens é superior a 60 quadros por segundo. Depois da aplicação dos métodos de tratamento de imagens e localização das regiões de interesse esse valor caiu para 24 por segundo na resolução 640x480 e 15 quadros por segundo na resolução 800x600.

6.2 Resultados Qualitativos

Com este trabalho de conclusão de curso obtivemos como resultado o domínio sobre a manipulação de imagens digitais na linguagem C# e Java e o êxito na aplicação de diversos métodos de visão computacional que já foram citados em itens anteriores.

O método de localização do objeto de interesse (marcador, padrões no interior de um quadrado preto), apresentou desempenho satisfatório em imagens estáticas e médio desempenho em imagens dinâmicas (gravação de vídeos).

Quanto à captura de imagens, os resultados obtidos após a mudança para a linguagem Java foram excelentes. Após a realização do tratamento das imagens o desempenho se mostrou satisfatório para o uso em realidade aumentada.

A aplicação de realidade aumentada desenvolvida ficou interessante e também apresentou resultados satisfatórios.

7 Conclusão

Os resultados obtidos dos algoritmos desenvolvidos para a localização do objeto de interesse (marcador) foram satisfatórios para imagens estáticas e para imagens dinâmicas (gravação e reprodução de vídeo), se considerarmos o uso em realidade aumentada.

Em imagens dinâmicas a localização do marcador é prejudicada quando movimentamos o mesmo rapidamente. Isso porque em imagens estáticas temos uma imagem clara do marcador; mas quando movimentamos o marcador muito rapidamente a imagem dele obtida é borrada. E a partir desta os algoritmos de localização desenvolvidos não são capazes de localizar o marcador.

Porém essa característica é comum na maior parte dos algoritmos e aplicações desenvolvidas para realidade aumentada, que se limitam na maioria das vezes em deixar os marcadores parados na frente da webcam ou realizam movimentos suaves com os mesmos.

Podemos perceber uma grande diferença de desempenho em relação a biblioteca de captura de imagem utilizada nos testes dos algoritmos (em C#) e a utilizada na aplicação de realidade aumentada desenvolvida (Java com bibliotecas do Processing). Enquanto que os métodos de aquisição, processamento de imagem e localização do marcador demoravam em média 290 ms no software antigo, no novo esse tempo passou a ser de 24 ms.

Falando mais especificamente no desempenho de quadros por segundo, passamos de 2 quadros por segundo para 24 na nova aplicação. Isso viabiliza o uso dos algoritmos aqui desenvolvidos em uma aplicação em tempo real como a de realidade aumentada.

A aplicação de realidade aumentada desenvolvida ficou muito interessante e interativa. Seria muito gratificante ter a oportunidade de poder utilizá-la em eventos ou feiras de forma a reforçar a educação de crianças e adultos quanto à reciclagem.

8 Referências

[1] - http://www.se.rit.edu/~jrv/research/ar/introduction.html#Section1.2 - Acesso em 17/06/09.

[2] - AZUMA, R. et al. **Recent Advances in Augmented Reality**. Malibu, 2001.

[3] - AZUMA, R. et al. **A Survey of Augmented Reality**. Malibu, 1997.

[4] - SILVA, R. et al. **Introduction to Augmented Reality**. Petropolis, [entre 1995 e 2009].

[5] - REVISTA INFO EXAME. São Paulo: Editora Abril, mai. 2009.

[6] - http://www.pages.drexel.edu/~weg22/edge.html - Acesso em 19/06/09.

[7] - http://pt.wikipedia.org/wiki/Imagem_digital - Acesso em 19/06/09.

[8] - http://www.planet-source-code.com/vb/scripts/ShowCode.asp?txtCodeId=1339&lngWId=10 - Acesso em 20/06/09.

[9] - http://www.codersource.net/csharp_measure_execution_time.aspx - Acesso em 20/06/09.

[10] - TSUZUKI, M. Apostila de PMR2520 – Introdução ao CAD/CAM, Escola Politécnica da Universidade de São Paulo, São Paulo.

[11] - Santos, F. M. et al. **Estudo da Cinemática Inversa Aplicada Num Braço Robótico.** São Paulo, 2006

[12] - http://www.hitl.washington.edu/artoolkit/ - Acesso em 15/08/2010