Destroy The Earth

Centro Universitário SENAC
Guilherme Henrique, Guilherme Sousa e Ricardo Suman

Resumo

No projeto aqui descrito, conceitos de Visão Computacional foram utilizados para o desenvolvimento de um jogo com interface não baseada em mouse e teclado. O jogo criado tem como base os jogos Shoot 'Em Up nos quais um joystick controla a nave que tem por objetivo eliminar o máximo de inimigos possíveis que vinham de cima. Por meio de um objeto de determinada cor, o jogador pode interagir de modo que seus gestos guiam a nave com o objetivo de atingir os inimigos. A biblioteca de visão computacional OpenCV permitiu a manipulação das imagens obtidas pela câmera, que após técnicas para obter valores como o brilho nas imagens, possibilitou o mapeamento de coordenadas para a definição da posição da nave na tela.

I. Introdução

Este artigo tem o objetivo de descrever as técnicas e algoritmos utilizados para a realização do Projeto Integrador III. Onde foi desenvolvido um jogo utilizando visão computacional em linguagem de programação C juntamente com a biblioteca Allegro 5 e OpenCV. A junção das duas bibliotecas nos possibilita criar o jogo com interação feita através da câmera. A movimentação do jogo é totalmente baseada na obtenção de uma certa cor, que pela sua posição determina a localização da nave na tela.

II. Descrição

Com uma nave o jogador deve destruir o máximo de naves inimigas, atirando contra as mesmas. O movimento da nave é livre pela tela, pois os inimigos surgem de todos os cantos, a nave é controlada por uma câmera que captura a cor laranja, com o movimento da cor a nave se desloca pela tela. A movimentação das naves inimigas é devagar no início. Com o aumento da pontuação elas ficam mais rápidas e numerosas.

III. Visão Computacional

A proposta do Projeto Integrador III envolve usar comandos lógicos em linguagem C, versão 99, com a biblioteca de interface gráfica Allegro 5 associados à comandos da biblioteca de Visão Computacional OpenCV. Para tal, usamos a webcam como controle das direções de nossa nave. Usando conceitos básicos sobre imagens digitais pudemos ter controle sobre nosso jogo sem o uso do teclado.

A análise e manipulação das imagens obtidas pela câmera são fundamentais para a interação com o usuário. As três dimensões da matriz que compõe a imagem são usadas para a verificação de cada pixel ou seja, considerando um apontador "camera *matriz"temos:

```
matriz[y][x][0]
matriz[y][x][1]
matriz[y][x][2]
```

A primeira dimensão da matriz são as linhas (y) da imagem, a segunda dimensão são as colunas da imagem (x), e por fim a terceira dimensão representa os canais de cores vermelho (R), verde (G) e azul (B). Assim, podemos obter os valores de R, G e B na linha "y"e coluna "x"da imagem.

Após o uso da função de atualização da câmera, a seguinte função:

```
converte_HSV(cam);
```

Recebe a estrutura de "camera"apontada por "cam"para conversão de valores.

A partir de uma matriz com todos os valores de R, G e B de um frame, percorremos essa matriz e aplicamos a fórmula para converter os números dos pixels para HSV. Assim, a

cada conversão de pixel podemos verificar se ele está no intervalo que queremos.

Optamos por usar o espaço de cor HSV, em vez de o espaço mais comum de cor RGB. Isso nos dá a vantagem de ter um pequeno número de tonalidades (Matiz) para a bola laranja, apesar de possuir vários tons de laranja (de laranja escuro a um laranja brilhante). Assim, podemos escolher uma cor específica tornando possível identificá-la mesmo em lugares com pouca luz seguindo os seguintes parâmetros:

- Matiz (Hue)

  Indica a cor. Por exemplo, é possível identificar um céu azul (e todos os outros objetos em azul) do ciano ao roxo.

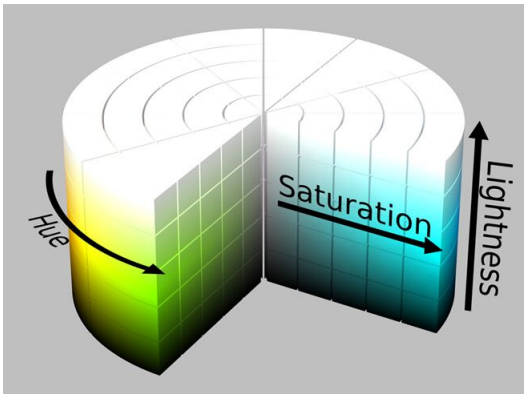
- Saturação (Saturation)

  Indica a quantidade de cor. Quanto menos cor, mais cinza.

- Luminância (Lightness)

  Indica o brilho da cor.

A escolha de cores no HSL não é baseada na mistura, como em RGB, mas sim em um esquema baseado em um cilindro. O primeiro é onde escolhemos a cor. Começamos no topo com vermelho, onde o valor é 0, e damos uma volta de 360 graus, retornando novamente no topo, na cor vermelha. Conforme aumentamos o valor vamos selecionando as cores.



Embora possamos escolher qualquer cor misturando as cores com o RGB, é mais preciso escolher uma cor específica e modificar sua luminosidade.

Para o cálculo do HSL é usado a seguinte fórmula:

$$H = \begin{cases} 60 * \frac{G-B}{MAX-MIN} + 0 \rightarrow if(MAX == R) \\ 60 * \frac{G-B}{MAX-MIN} + 360 \rightarrow if(MAX == R) \\ 60 * \frac{B-R}{MAX-MIN} + 120 \rightarrow if(MAX == G) \\ 60 * \frac{R-G}{MAX-MIN} + 240 \rightarrow if(MAX == B) \end{cases} \quad (1)$$

$$S = \frac{MAX - MIN}{MAX} \quad (2)$$

$$V = MAX \quad (3)$$

Para o cálculo, o valor de R, G e B devem ser normalizados entre 0.0 e 1.0. Assim calculamos o limite dos valores da seguinte forma:

```
float vermelho = r * 1.0;
float verde = g * 1.0;
float azul = b * 1.0;

float f_r = vermelho / 255;
float f_g = verde / 255;
float f_b = azul / 255;
```

Temos de multiplicar a quantidade de cor do pixel por 1.0 para que as variáveis f_r, f_g e f_b do programa em C possa realizar a divisão e obter a normalização com uma melhor precisão. Após a normalização, calculamos o maior valor e o menor valor entre os números normalizados e obtemos nossas variáveis controles "MAX"e "MIN". Dessa forma podemos calcular o valor HSV do pixel aplicando a fórmula mencionada e obtendo os valores para H, S e V. Ao final temos de transformar os valores "float"do HSV em inteiros para que fique fácil a manipulação de qual cor queremos para deteção. Para a cor laranja usada para a detecção, usamos:

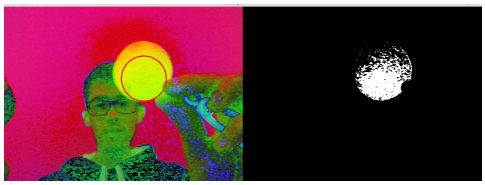
```
if (H <= 30 && S >= 50 && V >= 70)
```

Caso os valores convertidos dos pixels obedeçam a essa condição, sabemos que a cor do objeto foi detectada pela câmera e podemos aplicar o threshold.

A operação de threshold é uma forma simples e eficaz de selecionar um objeto. Nele, todos os pixels de uma imagem são comparados á um valor limite e são alterados conforme a necessidade. Um exemplo é comparar todos os pixels a um valor limite e caso sejam maiores não alterar seu valor e, caso sejam menores, igualar a zero. Desta forma destaca-se uma região da imagem.

Assim, quando encontramos um pixel no intervalo que definimos, podemos destacar o pixel e incrementar em 1 uma variável que usamos para saber quantos pixels temos destacados em um frame. Ao mesmo tempo outras duas variáveis servem de guia para a obtenção das coordenadas que guiarão a nave do jogador. Sendo que uma coordenada x é incrementada á cada coluna e outra coordenada y incrementada á cada linha. Quando fazemos a divisão das coordenas x e das coordenadas y percorridas pelo número de pixels destacados, temos as coordenadas da posição da cor reconhecida através da visão computacional.

A imagem abaixo mostra a imagem em HSL e a detecção feita pelo método de thresholding:



IV. Movimento das naves

A base da trajetória das naves encontra-se na distância entre dois pontos, as naves utilizam o cálculo desse triângulo retângulo criado entre estes pontos, em um plano cartesiano, primeiramente se verifica quem tem a maior distância, se no eixo X ou no eixo Y. Por exemplo, se no eixo Y a distância for 20 e a distância no eixo X for 10, o X tem distância menor, ou seja, vamos movimentar a nave no eixo X em uma constante 2, e o Y usa-se a fórmula:

$$Y = \frac{distanciaY}{distanciaX/2} \tag{4}$$

ou seja,

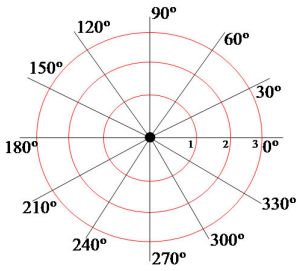
$$Y = \frac{20}{10/2} = 4 \tag{5}$$

X andará em 2 e o Y andará em 4, assim se manterá a proporção no movimento da nave até o objetivo. O valor padrão usado para movimentar o X (nesse caso 2) é o valor usado para dividir na fórmula do Y.

Se a distância de Y fosse menor que a distância de X a conta seria a mesma, porém com os eixos invertidos.

O movimento dos inimigos seguem os mesmos parâmetros do movimento da nave, pois o inimigo está se dirigindo para a nave do jogador, as naves inimigas surgem de qualquer lado, e tem como objetivo ir até a nave do jogador, surgem de início 5 naves inimigas, com o decorrer vão aumento o número,a velocidade também é alterada para dificultar o jogo.

V. Rotação da nave



A imagem das naves tem que estar de acordo com o movimento da mesma, para isso, calculamos o arco tangente para descobrir o angulo entre a nave e o objetivo, para utilizar a fórmula descrita abaixo, precisaremos da distância entre o eixo X da nave em relação ao eixo X de seu objetivo e da distância entre os eixos Y entre a nave e seu objetivo, com o valor

do ângulo acrescentamos 90, pois no Allegro o ângulo 0 está na parte superior. Assim a imagem vai se atualizando e acompanhando o movimento da nave objetivo.

$$distanciaX = ObjetivoX - NaveX \quad (6)$$

$$distanciaY = ObjetivoY - NaveY \quad (7)$$

$$angulo = atan\frac{distanciaY}{distanciaX} + 90 \quad (8)$$

VI. Colisão

A ideia da colisão é marcar quatro pontos nos cantos da nave jogador e das naves inimigas pegando a posição x e y da nave e subtraindo pela distâcia das bordas. Assim é possível criar uma espécie de "caixa imaginária"em volta das naves e saber se houve a colisão não importando onde estejam. No caso dos tiros é preciso verificar a posição x e y já que os tiros são pequenos e a diferença entre a borda dele (posição x e y) é muito pequena e quase imperceptível para quem está jogando.

VII. Considerações finais

Com isso concluímos este Projeto Integrador III cujo objetivo era o desenvolvimento de um jogo utilizando a Visão Computacional. Em relação a parte do jogo em si, houve apenas algumas dificuldades na adaptação para diversos ângulos (tiros, colisões, movimentos, nave seguir o mouse, etc), mas isso devido as próprias características do Allegro. Já com o OpenCV foi preciso um estudo acerca do assunto e dos algoritmos necessários para que sua implementação funcionasse em diversos ambientes.

Referências

[1] Augusto, M. F., Aplicação De Visão Computacional Para Extração De Características Em Imagens Do Olho Humano, 2007.

[2] RGB e HSL. Tableless. Disponível em: <http://tableless.com.br/rgb-e-hsl/>. Acesso em: 05 mai. 2015.

[3] Ajustar cores de imagem com controles deslizantes HSL. Adobe.com. Disponível em: <http://help.adobe.com/pt_BR/lightroom/using/WS46B0BFFC-868B-4f96-A182-418D53FD83FF.html>. Acesso em: 05 mai. 2015.

[4] Página Dinâmica para Aprendizado do Sensoriamento Remoto. UFRGS.br. Disponível em: <http://www.ufrgs.br/engcart/PDASR/formulario1.html>. Acesso em: 02 jun. 2015.

[5] OpenCV Rastreando Objetos. Academia.edu. Disponível em: <http://www.academia.edu/4654828/OpenCV_Rastreando_Objetos>. Acesso em: 02 jun. 2015.

[6] Object Detection with OpenCV. progCode.in. Disponível em: <http://www.progcode.in/object_detection_with_opencv.php>. Acesso em: 02 jun. 2015.

[7] HSV. Wikipedia.org. Disponível em: <http://pt.wikipedia.org/wiki/HSV>. Acesso em: 02 jun. 2015.

[8] RGB to HSL color conversion. RapidTables. Disponível em: <http://www.rapidtables.com/convert/color/rgb-to-hsl.htm>. Acesso em: 02 jun. 2015.

[9] HSV Color Conversion. Shervinemami.info. Disponível em: <http://shervinemami.info/ colorConversion.html>. Acesso em: 02 jun. 2015.

[10] OpenCV Threshold (Python , C++). Learn OpenCV. Disponível em: <http://www.learnopencv.com/opencv-threshold-python-cpp/>. Acesso em: 02 jun. 2015.

[11] Reading and Writing Images and Video. docs.opencv.org. Disponível em: <http://docs.opencv.org/modules/highgui/doc/reading_and_writing_images_and_video.html>. Acesso em: 02 jun. 2015.