**PSI3471 - Fundamentos de Sistemas Eletrônicos Inteligentes**

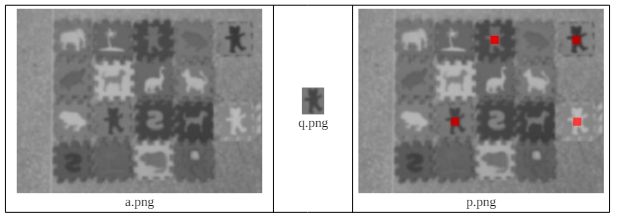
Lição de Casa 3 (Aulas 5 e 6)

| **Nome Completo** | **Número USP** |
| --- | --- |
| Gustavo Henrique da Silva Amaral | 12551686 |
| Thiago da Rocha Calomino Gonçalves | 12554647 |

**Lição de casa #3 (Parte 1)**

Escreva um programa que detecta as 4 ocorrências de urso “q.png” na imagem a analisar “a.png” gerando uma imagem processada semelhante a “p.png”, chamando uma única vez a rotina de template matching e sem detectar as bordas.

*Nota: Este exercício pode ser resolvido usando template matching aplicado diretamente em níveis de cinza. Não detecte as bordas antes de chamar template matching. Detectar bordas é uma solução muito menos robusta do que fazer template matching direto. Pense o seguinte. Digamos que você tenha detectado as bordas das imagens a e q abaixo e tenha conseguido um casamento das bordas das duas imagens. Neste caso, as bordas de a e q de deixam de “bater” se deslocar q em um pixel em qualquer direção, se mudar minimamente a escala ou fizer uma pequena rotação. Enquanto isso, o casamento levando em conta o nível de cinza das imagens continuam “batendo” mesmo com pequenas distorções.*



**Explicação completa:**

Usamos como entrada “a.png” e como imagem de busca de ocorrências, “q.png”. Criamos uma outra variável q2 que é o negativo de “q.png” para detectar todas as ocorrências possíveis.

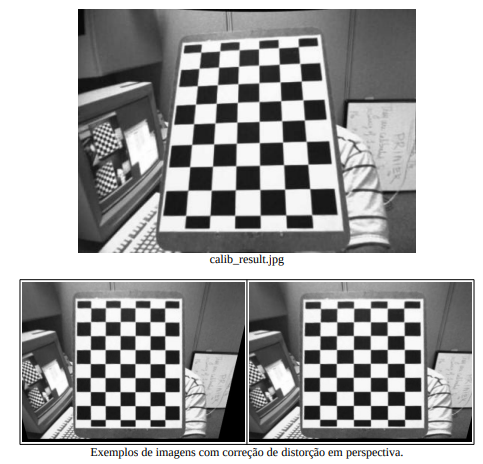
As variáveis inst e inst2 são template matching para q e q2. Após isso, fazemos um laço de repetição por toda a imagem em busca de ocorrências de inst e inst2 na imagem total e para cada ocorrência, marcamos com um pixel vermelho na imagem de saída

**Código da lição:**

| // Lição de casa 3 (Parte 1)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647  #include<cekeikon.h>  int main(){  COR vermelho = COR(0,0,255);  Mat\_<FLT> ent, q, q2, inst, inst2;  Mat\_<COR> saida;  float lim = 0.8;  le(ent,"a.png");  le(q, "q.png");  q2 = q.clone();  q2 = 1-q2;  inst = matchTemplateSame(ent,q,CV\_TM\_CCOEFF\_NORMED, 0.0);  inst2 = matchTemplateSame(ent,q2,CV\_TM\_CCOEFF\_NORMED, 0.0);  converte(ent,saida);  mostra(inst);  mostra(inst2);  for(int l = 0 ; l<inst.rows ; l++){  for(int c = 0 ; c<inst.cols ; c++){  if(inst(l,c)>lim || inst2(l,c)>lim){  saida(l,c) = vermelho;  }  }  }  imp(saida,"p.png");  } |
| --- |

**Lição de casa #3 (Parte 2)**

Corrija a deformação em perspectiva do tabuleiro de xadrez abaixo, gerando uma imagem onde cada casa do tabuleiro é um quadrado alinhado aos eixos do sistema de coordenadas. Consequentemente, o tabuleiro todo será um retângulo alinhado aos eixos do sistema de coordenadas. Nota: Você não precisa determinar automaticamente as esquinas do tabuleiro. Pode colocar no seu programa, manualmente, as suas coordenadas.

****

**Explicação:**

A imagem de entrada está em perspectiva e com as matrizes src e dst conseguimos uma transformação de perspectiva. Esta por sua vez é aplicada sobre a nossa imagem de entrada e gerando uma saída como a vista frontal do tabuleiro.

**Código da lição:**

| // Lição de casa 3 (Parte 2)  //  // Gustavo Henrique da Silva Amaral - 12551686  // Thiago da Rocha Calomino Gonçalves - 12554647  #include<cekeikon.h>  int main(){  Mat\_<GRY> ent, saida;  le(ent,"calib\_result.jpg");  mostra(ent);  Mat\_<float> src = (Mat\_<float>(4,2) <<  139,44,  108,295,  353,294,  322,34);  Mat\_<float> dst = (Mat\_<float>(4,2)<<  110,50,  110,300,  360,300,  360,50);  Mat\_<double> perspective\_correction = getPerspectiveTransform(src,dst);  warpPerspective(ent,saida,perspective\_correction,Size(ent.cols,ent.rows), 1,1);  imp(saida,"correcao.jpg");  } |
| --- |