Template Matching

Resumo: Durante duas aulas, desenvolveremos sistemas que detectam placas. Na fase 3, faremos um programa que detecta as placas nos quadros de um vídeo. Nesta fase, trabalharemos somente com computador, sem Raspberry. Usaremos processamento paralelo (OpenMP ou thread do C++) para acelerar o processamento. Na fase 4, faremos um sistema que faz o carrinho seguir uma placa.

Nota 2018: Este ano, temos uma aula a menos do que no ano passado. Fases 3 e 4 ficaram numa única aula. É muito difícil fazer fases 3 e 4 numa única aula. Façam a fase 3 em casa. A fase 3 é bem parecido com EP1 do primeiro semestre e não precisa do Raspberry. Dá para fazer em Windows ou Linux (usando Cekeikon/OpenCV).

Cuidado: Em todos os demais componentes do nosso projeto, o componente desenvolvido ou funciona ou não funciona - é fácil perceber quando tem algum problema. No template matching, existem diferentes "graus de funcionamento": pode não funcionar, pode funcionar mal, pode funcionar mais ou menos, pode funcionar bem a maior parte do tempo mas não sempre, até funcionar perfeitamente. Se template matching não estiver funcionando perfeitamente, o carrinho não conseguirá seguir a placa dependendo da iluminação, da "poluição visual" do ambiente, etc.

1 Introdução

A parte teórica do template matching já foi visto na disciplina teórica. Veja as apostilas no site www.lps.usp.br/hae/apostila.

- Explicação simplificada de template matching (tmatch-simp)
- Template matching (tmatch).
- Template matching invariante a RST (rstmatch).

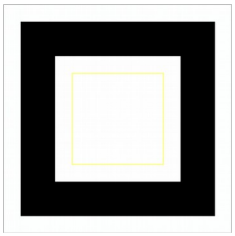
Em especial, preste atenção em:

1) O que se deve fazer para que alguns pixels sejam considerados "don't care".
2) Como obter template matching invariante somente ao brilho (detecta objetos de alto contraste, mesmo que o formato difira um pouco ao do modelo) e invariante invariante por brilho e contraste (detecta objetos com o formato parecido ao modelo buscado, mesmo que seja de baixo contraste).

2 Terceira fase do projeto

Faça um programa fase3.cpp que lê um vídeo 240x320 pixels (capturado*.avi), um modelo de placa quadrado.png e localiza a placa-modelo nos seus quadros, toda vez que a placa estiver entre aproximadamente 30 e 150 cm da câmera, gerando o vídeo de saída (figura 1).

Quadro 92 de capturado2.avi.



Placa quadrado.png.

Quadro 92 de localiza2.avi.

Figura 1: Localização da placa.

A quantidade de falsos positivos (dizer que há placa quando não há) do seu programa deve ter muito pequeno, caso contrário o carrinho pode começar a andar mesmo na ausência de uma placa. Da mesma forma, a quantidade de erros de localização (localizar placa numa posição incorreta) do seu programa deve ser muito pequeno para que o carrinho não ande numa direção errada (atrás de algum objeto que se parece com a placa). Por outro lado, o seu programa pode cometer alguns falsos negativos (dizer que não há placa quando há), pois neste caso o carrinho andará dando "solavancos", o que é menos grave do que os dois erros anteriores. A figura 2 mostra um caso de falso negativo.

Quadro 82 do capturado2.avi

Quadro 82 do localiza2.avi

Figura 2: Falso negativo (não conseguiu localizar placa borrada).

Os vídeos-testes (capturado?.avi) e a placa (quadrado.png) estão em:

```
http://www.lps.usp.br/hae/apostilaraspi/capturado.avi
http://www.lps.usp.br/hae/apostilaraspi/capturado2.avi
http://www.lps.usp.br/hae/apostilaraspi/capturado3.avi
http://www.lps.usp.br/hae/apostilaraspi/quadrado.png
```

Executando o seu programa compilado:

```
$ fase3 capturado.avi quadrado.png seu_localiza.avi
```

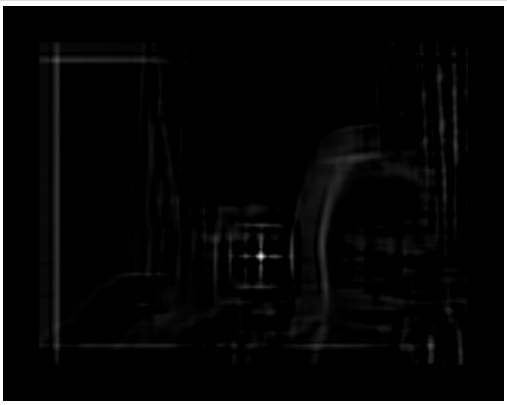
deverá ler o vídeo capturado.avi, a imagem quadrado.png e gerar o vídeo seu_localiza.avi. Exemplos de saída estão em:

```
http://www.lps.usp.br/hae/apostilaraspi/localiza.avi
http://www.lps.usp.br/hae/apostilaraspi/localiza2.avi
http://www.lps.usp.br/hae/apostilaraspi/localiza3.avi
```

No interior da placa quadrado.png, há um quadrado menor amarelo pouco visível. Dentro do quadrado amarelo, haverá dígitos manuscritos. Durante a localização da placa, você deve considerar o conteúdo no interior do quadrado amarelo como "don't care", para que o manuscrito não atrapalhe a busca do modelo. Todos os pixels brancos dentro do quadrado amarelo possuem valores (255, 255, 255), enquanto que os pixels brancos fora do quadrado amarelo possuem valores (255, 255, 254) ou (255, 254, 255) - em ordem (b, g, r). Veja na figura 3 a diferença entre usar ou não "don't care". Nas duas imagens, o pixel com o maior valor foi mapeado no branco e o pixel com o menor valor foi mapeado no preto. Repare que, sem usar "don't care", há vários pixels com alto brilho, enquanto que usando "don't care", praticamente somente o pixel do casamento verdadeiro fica altamente brilhante.

Repare que muitos quadros dos vídeos capturado2 e capturado3 estão bastante borrados, pois foram capturados pela câmera montado no carrinho em movimento..

(a) Saída sem usar "don't care".



(b) Saída usando "don't care".

Figura 3: Diferença entre usar ou não pixels "don't care". Nas duas imagens, o pixel com o maior valor foi mapeado no branco e o pixel com o menor valor foi mapeado no preto.

Experimente rodar o programa sem/com processamento paralelo usando OpenMP. Verifique a diferença na taxa de quadros por segundo.

Template matching com "don't care": O modelo tratado para fazer template matching NCC deve ter:

1) Pixels don't care mapeados em zero.
2) Soma de todos os pixels devem ser zero.

O modelo tratado para fazer template matching CC deve ter, além disso:

3) Soma absoluta de todos os pixels deve dar 2.

Quando lerem quadrado.png numa imagem FLT, preto será mapeado em 0 e branco (255,255,255) será mapeado em 1. Os pixels 1 devem ser considerados don't care.

Como eu fiz (2017): Primeiro, redimensiono a placa para 55x55 pixels. Depois, construo 8 modelos em progressão geométrica entre escalas 1 (55x55 pixels) e 0.273 (15x15 pixels). Faço o "tratamento dos modelos". Procuro todos esses modelos no quadro usando correlação não-normalizada. Pego a escala que deu a maior correlação não-normalizada (invariante a brilho) e faço a correlação normalizada (invariante a brilho e contraste) somente nessa escala. Se as duas correlações indicarem posições muito próximas e os valores das duas correlações forem acima de 0,18, considero que há uma verdadeira detecção de placa. Caso contrário, considero que não há placa.

Como eu fiz (2018): Este ano, fiz uma espécie de "tracking" simples para melhorar a detecção das placas. Quando tem uma placa estática na frente do carrinho, essa placa é detectada e o carrinho começa a se mover. Só que, com o movimento do carrinho, a imagem da placa capturada fica borrada pelo movimento, impedindo a nova detecção, e o carrinho para. Com o carrinho parado, a imagem fica nítida novamente e a placa é novamente detectada. Isso faz com que o carrinho ande dando "solavancos". A ideia é, se no quadro anterior havia com certeza uma placa, no quadro atual deve ter placa numa posição e escala próximas. Primeiro, redimensiono a placa para 55x55 pixels. Depois, construo 8 modelos em progressão geométrica entre escalas 1 (55x55 pixels) e 0.273 (15x15 pixels). Faço o "tratamento dos modelos". (a) Se não havia localizado placa no quadro anterior, procuro todos esses modelos no quadro atual usando correlação não-normalizada. Pego a escala que deu a maior correlação não-normalizada e faço a correlação normalizada somente nessa escala. Se as duas correlações indicarem posições muito próximas e os valores das duas correlações forem acima de um certo limiar (0,18), considero que há uma verdadeira detecção de placa. Caso

contrário, considero que não há placa. (b) Se havia localizado placa no quadro anterior, procuro pela placa somente nos pixels próximos usando um limiar mais baixo (0,10) e somente nas escalas próximas. Se não consegui localizar placa desta forma, tento detectar placa da forma (a).

Item obrigatório valendo 5 pontos) (a) Faça o programa fase3 e teste o programa nos vídeos capturado?.avi. Mostre ao professor. (b) Acelere o programa usando o processamento paralelo OpenMP ou thread do C++ (fase3paralela). Mostre ao professor o programa acelerado.

Nota: Veja a apostila "paralelismo" para exemplos de processamento paralelo.
Sugestão: Pode testar primeiro o seu programa nas imagens capturado2-???.jpg.

3 Quarta fase do projeto

Modifique o programa da fase 3 (fase3 ou fase3paralela) para controlar o carrinho de forma que ele siga continuamente a placa quadrado.png. Para isso, você irá usar somente a posição *x* da localização da placa, desprezando a posição *y*.

```
raspberrypi$ servidor4
computador$ cliente4 192.168.0.110 [videosaida.avi]
```

Como eu fiz: Fiz com que a diferença das velocidades das rodas esquerda e direita dependa do quão distante a posição *x* da placa está do centro da imagem. Isto é, se a placa estiver muito à esquerda, o carrinho irá girar rapidamente para esquerda. Se a placa estiver ligeiramente fora do centro, o carrinho irá corrigir a rota suavemente. Isto fez com que o movimento do carrinho ficasse "suave".

O carrinho deve parar nas duas situações:

a) Quando não conseguir localizar a placa no quadro do vídeo capturado.
b) Quando a câmera estiver muito próximo da placa, para evitar uma colisão. Note que o carrinho não consegue parar instantaneamente.

Este programa deve rodar no modo servidor-cliente (servidor4.cpp e cliente4.cpp), onde o processamento mais pesado (a localização da placa) deve ser realizado no computador.

O vídeo abaixo mostra o carrinho seguindo a placa:

```
http://www.lps.usp.br/hae/apostilaraspi/segueplaca.mp4
http://www.lps.usp.br/hae/apostilaraspi/segueplaca2.mp4
```

Nota: Pode utilizar o processamento paralelo (OpenMP ou thread do C++).

Item obrigatório valendo 5 pontos) Mostre ao professor servidor4/cliente4 funcionando.