Digital Image Processing

Berlin University of Technology (TUB),
Computer Vision and Remote Sensing Group
Berlin, Germany

6. Exercise – Given

```
int main(int argc, char** argv)
```

- Loads image, extracts and shows keypoints
- argv[1] == path to image
- argv[2] == scale of kernel (std-dev)

```
void showImage(Mat& img, const char* win, int wait, bool show,bool save)
```

`img`	image
`win`	window/file name
`wait`	time to wait (0 == wait for key pressed)
`show`	shall the image be shown...
`save`	and/or saved? image will be saved to ./img/ (has to be created before)

- shows/saves image (normalized! [min, max] → [0,255])

```
Mat nonMaxSuppression(Mat& img)
```

`img`	input image
`return`	output image

- deletes all non-maxima (maxima = largest value in 4-neighbourhood)

6. Exercise – ToDo

```
void getInterestPoints(Mat& img, double sigma, vector<KeyPoint>& points)
```

`img`	input image
`sigma`	std-dev of filter kernel (first derivative of Gaussian)
`points`	found keypoints

- Computes keypoints using structure tensor
- Needs image gradients: → Calculate directional gradients
  → Convolution with first derivative of Gaussian

```
Mat createFstDevKernel(double sigma)
```

`sigma`	std-dev of filter kernel (first derivative of Gaussian)
`return`	the created kernel

- Generates kernel that corresponds to the first derivative of a Gaussian

Useful cv functions: filter2D(..), multiply(..), GaussianBlur(..), divide(..), threshold(..)

Deadline: 26.01.2018

```
// uses structure tensor to define interest points (foerstner)
void Dip5::getInterestPoints(Mat& img, double sigma, vector<KeyPoint>& points){

    // generate filter kernel
    // 1st derivative of gaussian
    Mat devGauss_x, devGauss_y;
    devGauss_x = createFstDevKernel(sigma);
    devGauss_y = devGauss_x.clone().t();

    // calculate directional gradients
    Mat grad_x, grad_y;
...

    // calculate structure tensor
    // squared gradient
    Mat grad_xx, grad_yy, grad_xy;
...
    // averaging with gaussian window of size 7
    int n = 7;
    GaussianBlur(...
...

    // calculate trace ...
...
    // ... and determinant of structure tensor
...
```

```
    // get weight of interest points
...
    // non-maxima suppression
...
    // global thresholding
...

    // get isotropy of interest points
...
    // non-maxima suppression
...
    // global thresholding
...

    // get interest points
    Mat interest;
    multiply(weight, isotropy, interest);
    for(int x=0; x<img.cols; x++){
                for(int y=0; y<img.rows; y++){
                        if (interest.at<float>(y, x) > 0){
                                points.push_back(KeyPoint(Point2f(x,y),sigma*10));
                        }
                }
    }
}
```

```
// creates a specific kernel
/*
kernel  the calculated kernel
kSize   size of the kernel
name    specifies which kernel shall be computed
*/
Mat createKernel(double sigma, string name){

    int kSize = 5*sigma;
    if (kSize<3) kSize = 3;

    // alloc memory for 1D-kernel (horizontal)
    Mat kernel(kSize, kSize, CV_32FC1);

    // some variables
    double mu_x = kernel.cols/2;        // x-coordinate of center
    double mu_y = kernel.cols/2;        // x-coordinate of center
    double sigma_x = sigma;                     // standard deviation in x direction
    double sigma_y = sigma;                     // standard deviation in y direction
...
}


// creates kernel representing fst derivative of a Gaussian kernel in x-direction
/*
sigma   standard deviation of the Gaussian kernel
return  the calculated kernel
*/
Mat Dip5::createFstDevKernel(double sigma){

        return createKernel(sigma, "gaussianDevX");

}
```