VISION WITH OF

Add Vision to your Apps

Hasan Ijaz

November 6, 2015

EmbeddedLance

MOTION ANALYSIS

Optical Flow

Optical Flow

- motion between two consecutive frames from a camera feed at every voxel position
- calculated with differential methods

Implementation

- Block-matching
- Lucas Kanade

Code

cvOpticalFlowLK

```
void cvCalcOpticalFlowLK(const CvArr* prev, const CvArr* curr, CvSize win_size,
    CvArr* velx, CvArr* vely)
```

- prev – input frame 1
- curr – input frame 2
- win_size – Size of the averaging window used for grouping pixels
- velx – output velocity in x direction
- vely – output velocity in y direction

Code

calcOpticalFlowPyrLK

```
void calcOpticalFlowPyrLK(InputArray prevImg, InputArray nextImg, InputArray
    prevPts, InputOutputArray nextPts, OutputArray status, OutputArray err, Size
    winSize=Size(21,21), int maxLevel=3, TermCriteria criteria=TermCriteria(
    TermCriteria::COUNT+TermCriteria::EPS, 30, 0.01), int flags=0, double
    minEigThreshold=1e-4 )
```

- prevImg – input frame/image 1
- nextImg – input frame/image 2
- prevPts – vector of 2D points for which the flow needs to be found;
- nextPts – output vector of 2D points containing the calculated new positionsof input features in the second image;
- status – output status vector

calcOpticalFlowPyrLK

```
void calcOpticalFlowPyrLK(InputArray prevImg, InputArray nextImg, InputArray
    prevPts, InputOutputArray nextPts, OutputArray status, OutputArray err, Size
    winSize=Size(21,21), int maxLevel=3, TermCriteria criteria=TermCriteria(
    TermCriteria::COUNT+TermCriteria::EPS, 30, 0.01), int flags=0, double
    minEigThreshold=1e-4 )
```

- err – output vector of errors;
- winSize – size of the search window at each pyramid level.
- maxLevel – 0-based maximal pyramid level number;
- criteria – parameter, specifying the termination criteria of the iterative search algorithm
- flags.
- minEigThreshold – allows to remove bad points and get a performance boost.

GoodFeaturesToTrack

Shi Tomashi

```
void cvGoodFeaturesToTrack(const CvArr* image, CvArr* eig_image, CvArr* temp_image
    , CvPoint2D32f* corners, int* corner_count, double quality_level, double
    min_distance, const CvArr* mask=NULL, int block_size=3, int use_harris=0,
    double k=0.04 )
```

- image – Input 8-bit or floating-point 32-bit, single-channel image.
- corners – Output vector of detected corners
- maxCorners – Maximum number of corners to return. If there are more corners than are found, the strongest of them is returned
- qualityLevel – Parameter characterizing the minimal accepted quality of image corners. the lower the more points we will get

GoodFeaturesToTrack

```
void cvGoodFeaturesToTrack(const CvArr* image, CvArr* eig_image, CvArr* temp_image
    , CvPoint2D32f* corners, int* corner_count, double quality_level, double
    min_distance, const CvArr* mask=NULL, int block_size=3, int use_harris=0,
    double k=0.04 )
```

- minDistance – Minimum possible Euclidean distance between the returned corners.
- mask – Optional region of interest
- blockSize – average block size for computing a derivative covariation matrix
- useHarrisDetector – Parameter indicating whether to use a Harris detector, makes tracking a bit better when set to true
- k – Free parameter of the Harris detector

Code

FAST

```
void FAST(InputArray image, vector<KeyPoint>& keypoints, int threshold, bool
    nonmaxSuppression)
```

- image – grayscale input image
- keypoints – output keypoints
- threshold – threshold
- nonmaxSuppression – if true, non-maximum suppression is applied to detected corners

You can perform motion analysis using openFrameworks :)