Chapter 10 : Tracking and Motion

Md.Al-Helal, 2018-19

Computer Science & Engineering
University of Dhaka

May 16, 2018

The Basics of Tracking

Understanding the motion of a object has two main components:

- **Identification**
  Finding the object of interest from one frame in a subsequent frame of the video stream.
  - Momemnts
  - Coloer Histogram

  Tracking unidentified objects is important when we wish to determine what is interesting based on its motion.
  - Lucas-Kanade
  - Horn-Schunck
- **Modeling**
  Provides a noisy measurement of the object's actual position.Many powerful mathematical techniques have been developed for this.
  These methods are applicable to 2D or 3D objects and their locations.

Corner Finding I

- If we pick a **point on a large blank wall** then it won't be easy to find that same point in the next frame of a video.
- If all points on the wall are **identical or even very similar**, then we won't have much luck tracking that point in subsequent frames.
- On the other hand, if we choose a point that is **unique** then we have a pretty **good chance** of finding that point again.
  In practice, the point or feature we select should be **unique, or nearly unique**, and should be **parameterizable** in such a way that it can be **compared** to other points in another image.

Corner Finding II

Corner Finding III

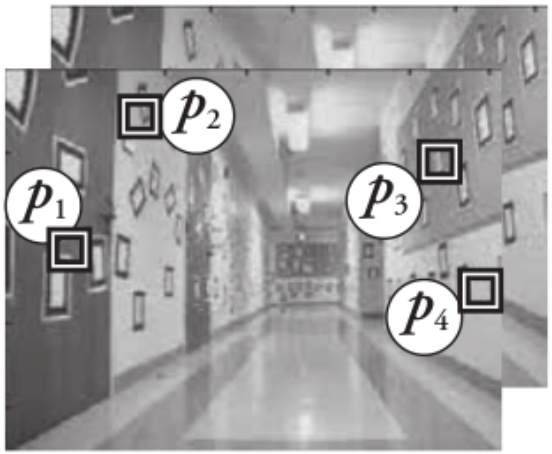
How we can find the corners?
If strong derivatives are observed in two orthogonal directions then we can hope that this point is more likely to be unique. For this reason, many trackable features are called **corners** (corners are not edges).
The cvGoodFeaturesToTrack() function computes the second derivatives (using the Sobel operators) and returns a list of the points that meet our definition of being good for tracking.

Code

```
void cvGoodFeaturesToTrack(
        const CvArr*     image,
        CvArr*           eigImage,
        CvArr*           tempImage,
        CvPoint2D32f*    corners,
        innt*            corner_count,
        double           quality_level,
        double           min_distance,
        const CvArr*     mask = NULL,
        int              block_size   = 3,
        int              use_harris   = 0,
        double           k            = 0.4
);
```

tempImage and eigImage	Used as scratch by the algorithm
corners	contain the result points after the algorithm has run
corner_count	indicates the maximum number of points
quality_level	indicates the minimal acceptable lower eigenvalue for a point
min_distance	guarantees that no two returned points are within the indicate

Optical Flow I



$I(t)$, Points : $\{p_i\}$

$I\,(t+1)$

Velocity vectors $\{\vec{v}_3\}$

Image sequence
(single camera)

Tracked sequence

Optical Flow II

- **Dense optical flow**
  Find velocity or displacement for all pixel between previous and current frame. Horn-Schunck method and the block matching method.
- **Sparse optical flow**
  Find velocity or displacement for subset of pixel those have certain desirable properties such as 'corners'. Most popular sparse tracking technique, Lucas-Kanade (LK) optical flow.

Optical Flow (Dense Optical Flow)

Horn-Schunck technique code

```
void cvCalcOpticalFlowHS(
        const CvArr*        imgA,
        const CvArr*        imgB,
        int                 usePrevious,
        CvArr*              velx,
        CvArr*              vely,
        double              lambda,
        CvTermCriteria      criteria
);
```

lambda	Lagrange multiplier
usePrevious	tells the algorithm to use the velx and vely velocities compute
criteria	termination criteria

Optical Flow (Dense Optical Flow)

Block matching code

```
void cvCalcOpticalFlowBM(
        const CvArr*        prev,
        const CvArr*        curr,
        CvSize              block_size,
        CvSize              shift_size,
        CvSize              max_range,
        int                 use_previous,
        CvArr*              velx,
        CvArr*              vely
);
```

block_size	size of the block to be used
shift_size	step size between blocks
max_range	size of the region around a given block that will be searched fo

Optical Flow (Sparse Optical flow)

Lucas-Kanade code

```
void cvCalcOpticalFlowLK(
        const CvArr*    imgA,
        const CvArr*    imgB,
        CvSize          winSize,
        CvArr*          velx,
        CvArr*          vely
);
```

winSize window used for computing the local coherent motion

Thank You