Synthetic Sensing: Machine Vision: Tracking II

MediaRobotics Lab, March 2010

References:

Forsyth / Ponce: Computer Vision

Horn: Robot Vision

Schunk: Machine Vision

University of Edingburgh online image processing reference
http://www.cee.hw.ac.uk/hipr/html/hipr_top.html

The Computer Vision Homepage
http://www.cs.cmu.edu/~cil/vision.html

Rice University Eigenface Group
http://www.owlnet.rice.edu/~elec301/Projects99/faces/code.html

OpenCV
http://opencv.willowgarage.com/wiki/
http://opencv.willowgarage.com/wiki/CvReference

Also
https://webeng.cs.ait.ac.th/cvwiki/opencv:tutorial:optical_flow
http://homepages.inf.ed.ac.uk/rbf/CVonline/LOCAL_COPIES/OWENS/LECT12/node4.html
http://ai.stanford.edu/~dstavens/cs223b/

Tracking

The idea is old. Tracking is just keep note of things ...

General requirements:

- a something to detect
- a way of representing that object to your system
- a way to tally the results
- a way to find previous results
- a way to recover from mistakes

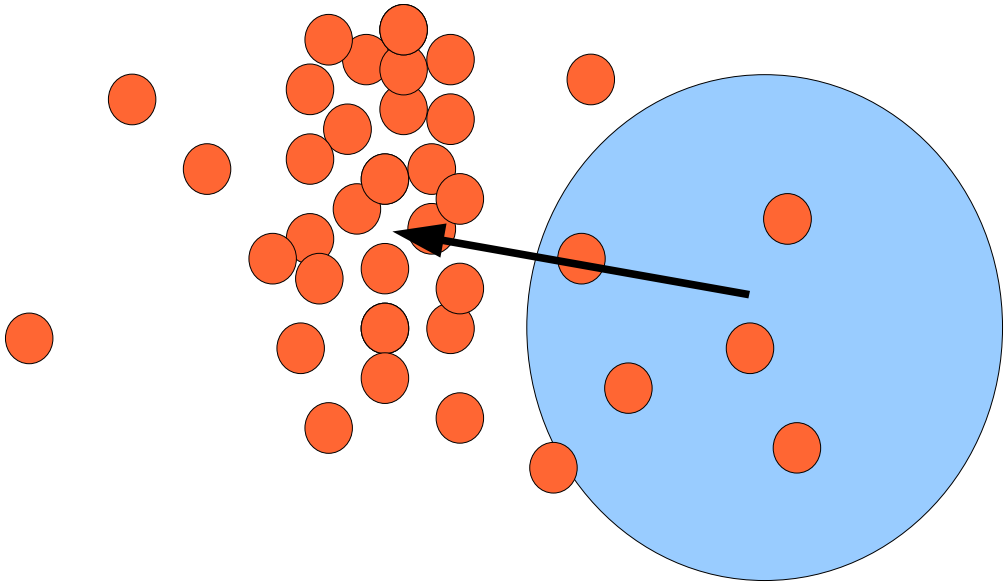
Available through opencv:

Color object tracking:	cvCamShift
Motion templates:	cvSegmentMotion, cvCalcMotionGradient
Feature tracking:	cvCalcOpticalFlow PyrLK

Color object tracking: cvCamShift (based on mean shift)

Mean-shift algorithm:

Robust method of finding local extrema in the density distribution of a data set.
This is easy for continuously distributed data, but difficult for discrete data sets.



Mean-shift algorithm:

Approach:

1 Choose a search window with
- -a starting location
- -a type (uniform, exponential, Gaussian)
- -a shape (symmetric, skewed)
- -a size

2 Compute the center of Mass (CoM)

3 Center the window on the CoM

4 Return to 2 until the window stops moving.

This terminates when a local maximum is found. The size of the window matters (it will change the local maxima).

Applied to successive images (showing the same feature), the mean-shift algorithm will find the new peak/mode of the feature in motion across the screen.

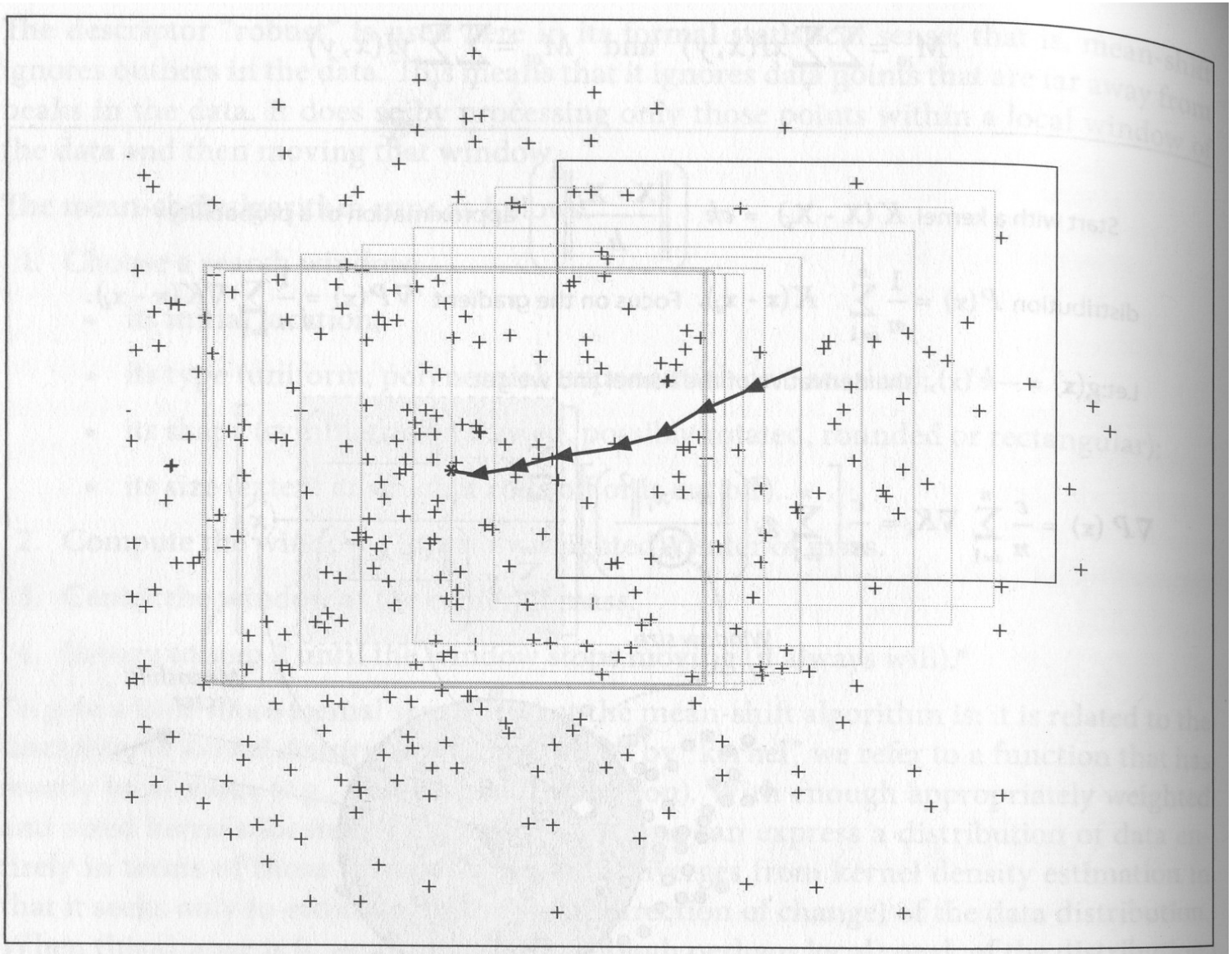
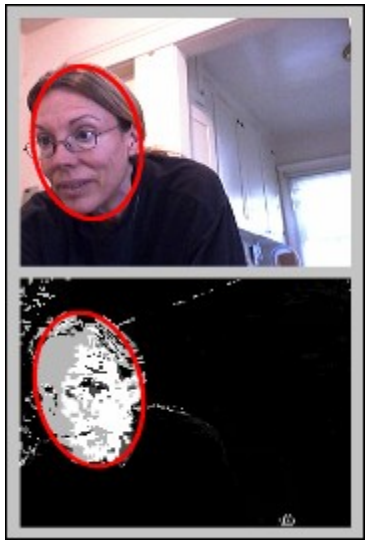


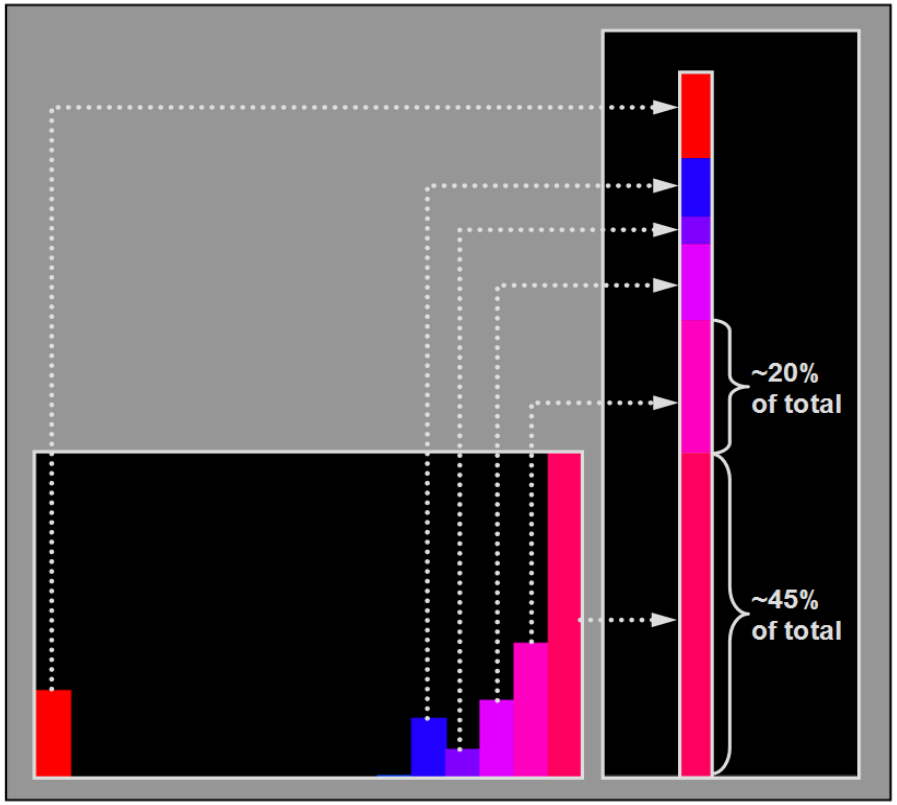
Figure 10-12. Mean-shift algorithm in action: an initial window is placed over a two-dimensional array of data points and is successively recentered over the mode (or local peak) of its data distribution until convergence

Camshift algorithm

(Continuously Adaptive Mean Shift Algorithm)

1. Create a color histogram to represent the face
2. Calculate a "face probability" for each pixel in the incoming video frames
3. Shift the location of the face rectangle in each video frame
4. Calculate the size and angle





http://www.cognotics.com/opencv/servo_2007_series/part_3/sidebar.html
http://crpit.com/confpapers/CRPITV36Allen.pdf

Color object tracking: cvCamShift

int cvCamShift(const CvArr* prob_image, CvRect window, CvTermCriteria criteria, CvConnectedComp* comp, CvBox2D* box=NULL);
Finds object center, size, and orientation

int cvMeanShift(const CvArr* prob_image, CvRect window, CvTermCriteria criteria, CvConnectedComp* comp);
Finds object center on back projection through a search window that adjusts in size

The function cvCamShift implements CAMSHIFT object tracking algrorithm ([Bradski98]). First, it finds an object center using cvMeanShift and, after that, calculates the object size and orientation. The function returns number of iterations made within cvMeanShift.

CvCamShift – core elements

```
char* find_colored_agent (IplImage* image, char* awindow,
bool firsttime, float min_area, float max_area)

{

        cvSplit( hsv, hue, 0, 0, 0 );                          //split into hue and hsv

        cvCalcHist( &hue, hist, 0, mask );                     //create histogram
        cvGetMinMaxHistValue( hist, 0, &max_val, 0, 0 );       //get min and max values


        cvCalcBackProject( &hue, backproject, hist );          //get the histogram data

        cvAnd( backproject, mask, backproject, 0 );            //mask

        cvCamShift( backproject, track_window, TermCriteria( ... ,&track_comp, &track_box );
        cvEllipseBox( image, track_box...);

}
```

CvCamShift

```
char* find_colored_agent (IplImage* image, char* awindow, bool firsttime, float min_area, float max_area)
{
		//adapted from the continuously adapted mean shift algorithm (camshaft)
		//as delineated in Computer Vision Face Tracking For Use in a Perceptual User Interface.

		short backproject_mode = 0;
		short show_hist = 1;
		int hdims = 16;
		short vmin = 10, vmax = 256, smin = 30;
		short i, bin_w;
		char* result = "0";

		//create the helper images on the first pass only
		if (firsttime)
		{

				cvNamedWindow( "Histogram", 1 );
				cvSetMouseCallback( awindow, on_mouse, 0 );

				hsv = cvCreateImage( cvGetSize(image), 8, 3 );
				hue = cvCreateImage( cvGetSize(image), 8, 1 );
				mask = cvCreateImage( cvGetSize(image), 8, 1 );
				backproject = cvCreateImage( cvGetSize(image), 8, 1 );
				hist = cvCreateHist( 1, &hdims, CV_HIST_ARRAY, &hranges, 1 );
				histimg = cvCreateImage( cvSize(320,200), 8, 3 );
				cvZero( histimg );
		}

		cvCvtColor( image, hsv, CV_RGB2HSV );
```

CvCamShift

```
//check the range and split hsv into the hue
if( track_object )
{
        int _vmin = vmin, _vmax = vmax;
        cvInRangeS( hsv, cvScalar(0,smin,MIN(_vmin,_vmax),0),cvScalar(180,256,MAX(_vmin,_vmax),0), mask );
        cvSplit( hsv, hue, 0, 0, 0 );

        //set region of interest, calculate the histogram
        if( track_object < 0 )
        {
                float max_val = 0.f;
                cvSetImageROI( hue, selection );
                cvSetImageROI( mask, selection );
                cvCalcHist( &hue, hist, 0, mask );
                cvGetMinMaxHistValue( hist, 0, &max_val, 0, 0 );
                cvConvertScale( hist->bins, hist->bins, max_val ? 255. / max_val : 0., 0 );
                cvResetImageROI( hue );
                cvResetImageROI( mask );
                track_window = selection;
                track_object = 1;
                cvZero( histimg );
                bin_w = histimg->width / hdims;
                for( i = 0; i < hdims; i++ )
                {
                        int val = cvRound( cvGetReal1D(hist->bins,i)*histimg->height/255 );
                        CvScalar color = hsv2rgb(i*180.f/hdims);
                        cvRectangle( histimg, cvPoint(i*bin_w,histimg->height),
                        cvPoint((i+1)*bin_w,histimg->height - val),color, -1, 8, 0 );
                }

        }//end if object <0
        //get the histogram data
        cvCalcBackProject( &hue, backproject, hist );
        //mask
        cvAnd( backproject, mask, backproject, 0 );
```

CvCamShift

```
                        //find object center, size, and orientation (CORE search component of this app)
                        cvCamShift( backproject, track_window,
                                        CvTermCriteria( CV_TERMCRIT_EPS | CV_TERMCRIT_ITER, 10, 1 ),&track_comp,
                                        &track_box );
                        track_window = track_comp.rect;
                        if( backproject_mode )
                                        cvCvtColor( backproject, image, CV_GRAY2RGB );

                        if( image->origin )
                                        track_box.angle = -track_box.angle;

                        cvEllipseBox( image, track_box, CV_RGB(255,0,255), 2, CV_AA, 0 );
                        (CV_AA: draw the lines antialiased)

                        //find out how many pixels of the set distribution there are..
                        float area = track_box.size.width * track_box.size.height;
                        if((area > min_area) && (area < max_area))
                        {
                                        //printf("area is: %d\n", int(area));
                                        result = "EP";
                        }

        }//end if trackobject

        if( select_object && selection.width > 0 && selection.height > 0 )
        {
                        cvSetImageROI( image, selection );
                        cvXorS( image, cvScalarAll(255), image, 0 );
                        cvResetImageROI( image );
        }

        cvShowImage(awindow, image );
        cvShowImage("Histogram", histimg );

        return(result);
}//end function
```

Motion templates: cvSegmentMotion, cvCalcMotionGradient

> motempl.py

Motion templates are an effective method of tracking general motion and often used in gesture recognition. Using motion templates requires a 'silhouette' of an object (a clearly defined object in front of a uniform background).
With this you can also record the motion history of an object. The changing positions of the object over time create a trajectory. You can find this trajectory , the overall motion, by taking the gradient of the image objects dispersed on the image.
CvCalcmotionGradient outputs a mask, a single channel 8-bit image in which nonzero entries indicate where valid gradients are located and orientation, a floating point image that gives the gradient direction's angle at each point.

Motion templates: cvSegmentMotion, cvCalcMotionGradient

void cvCalcMotionGradient (const CvArr* mhi, CvArr* mask, CvArr* orientation,double delta1, double delta2, int aperture_size=3);
Calculates gradient orientation of motion history image

void cvUpdateMotionHistory(const CvArr* silhouette, CvArr* mhi, double timestamp, double duration);
Updates motion history image by moving silhouette

Motion templates: cvSegmentMotion, cvCalcMotionGradient

```
void cvCalcMotionGradient(
    const CvArr* mhi,
    CvArr* mask,
    CvArr* orientation,
    double delta1,
    double delta2,
    int aperture_size=3
);
```

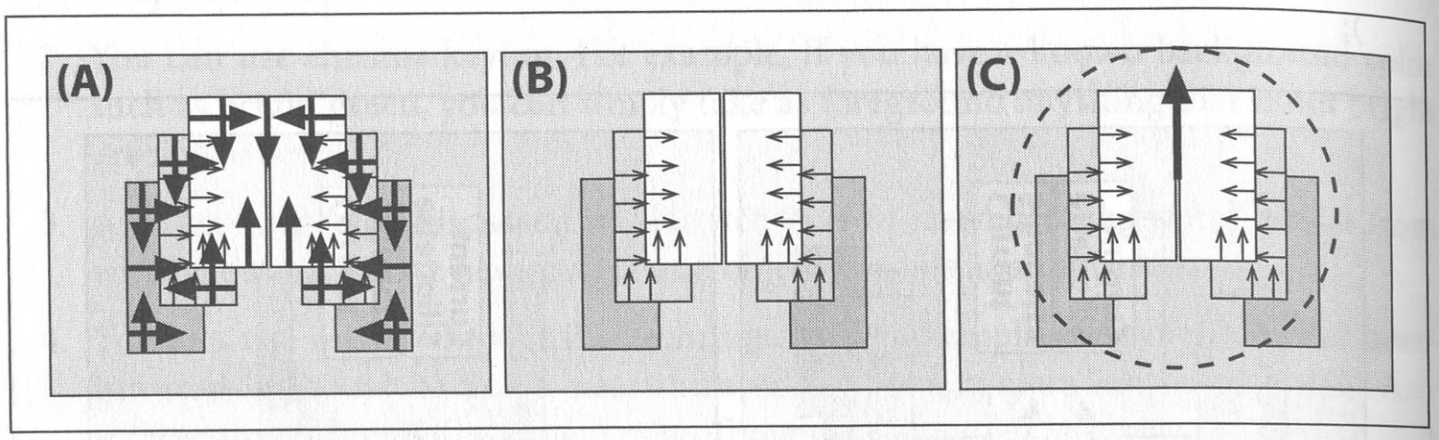


Figure 10-15. Motion gradients of the mhi image: (A) gradient magnitudes and directions; (B) large gradients are eliminated; (C) overall direction of motion is found

CvCalcMotionGradient – core elements

```
cvCvtColor( img, buf[last], CV_BGR2GRAY );                          # convert frame to grayscale

cvAbsDiff( buf[idx1], buf[idx2], silh );                            # get difference between frames
cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY );     # and threshold it
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION );        # update MHI

cvMerge( mask, None, None, None, dst );
cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3 );

seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );
for i in range(-1, seq.total):
        angle = cvCalcGlobalOrientation( orient_roi, mask_roi, mhi_roi, timestamp, MHI_DURATION);
        count = cvNorm( silh_roi, None, CV_L1, None ); # calculate number of points within silhouette ROI
        cvCircle...
        cvLine...
```

cvCalcMotionGradient

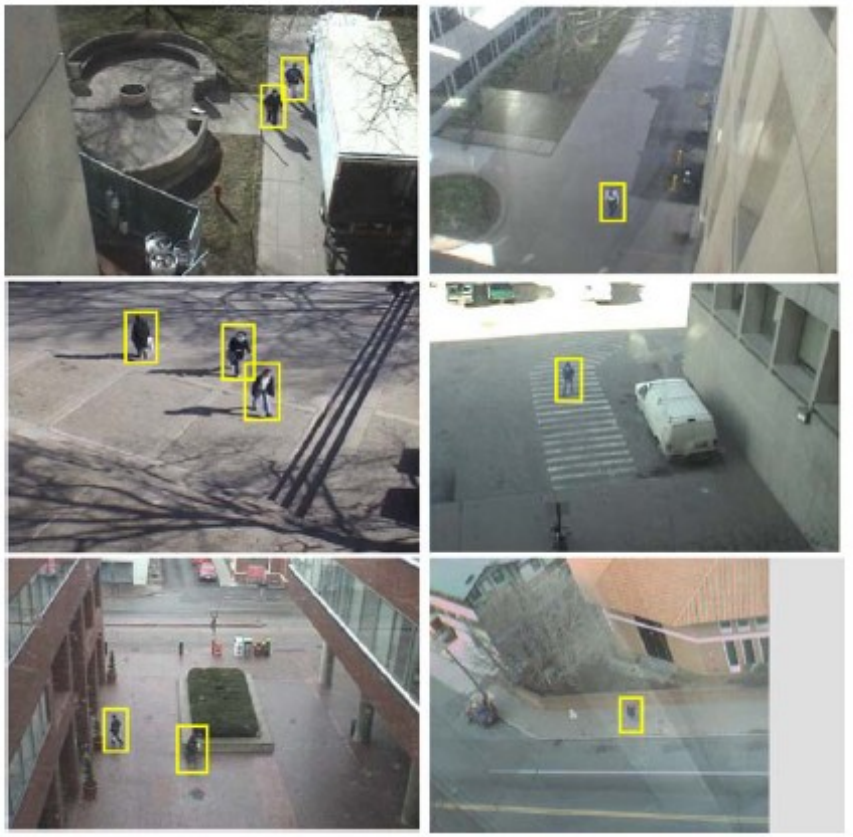
```
#!/usr/bin/python
from opencv.cv import *
from opencv.highgui import *
import sys
import time
from math import cos,sin

CLOCKS_PER_SEC = 1.0    MHI_DURATION = 1;       MAX_TIME_DELTA = 0.5;  MIN_TIME_DELTA = 0.05;
N = 4;                  buf = range(10)         last = 0;              mhi = None; # MHI
orient = None; # orientation
mask = None; # valid orientation mask
segmask = None; # motion segmentation map
storage = None; # temporary storage


def update_mhi( img, dst, diff_threshold ):
    global last
    global mhi
    global storage
    global mask
    global orient
    global segmask
    timestamp = time.clock()/CLOCKS_PER_SEC; # get current time in seconds
    size = cvSize(img.width,img.height); # get current frame size
    idx1 = last;
    if not mhi or mhi.width != size.width or mhi.height != size.height:
        for i in range( N ):
            buf[i] = cvCreateImage( size, IPL_DEPTH_8U, 1 );
            cvZero( buf[i] );
        mhi = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        cvZero( mhi ); # clear MHI at the beginning
        orient = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        segmask = cvCreateImage( size, IPL_DEPTH_32F, 1 );
        mask = cvCreateImage( size, IPL_DEPTH_8U, 1 );
```

CvCalcMotionGradient

```
cvCvtColor( img, buf[last], CV_BGR2GRAY ); # convert frame to grayscale
idx2 = (last + 1) % N; # index of (last - (N-1))th frame
last = idx2;
silh = buf[idx2];
cvAbsDiff( buf[idx1], buf[idx2], silh ); # get difference between frames
cvThreshold( silh, silh, diff_threshold, 1, CV_THRESH_BINARY ); # and threshold it
cvUpdateMotionHistory( silh, mhi, timestamp, MHI_DURATION ); # update MHI
cvCvtScale( mhi, mask, 255./MHI_DURATION,(MHI_DURATION - timestamp)*255./MHI_DURATION );
cvZero( dst );
cvMerge( mask, None, None, None, dst );
cvCalcMotionGradient( mhi, mask, orient, MAX_TIME_DELTA, MIN_TIME_DELTA, 3 );
if( not storage ):
        storage = cvCreateMemStorage(0);
else:
        cvClearMemStorage(storage);
seq = cvSegmentMotion( mhi, segmask, storage, timestamp, MAX_TIME_DELTA );
for i in range(-1, seq.total):
    if( i < 0 ):  # case of the whole image
            comp_rect = cvRect( 0, 0, size.width, size.height );
            color = CV_RGB(255,255,255);
            magnitude = 100.;
      else:  # i-th motion component
            comp_rect = seq[i].rect
            if( comp_rect.width + comp_rect.height < 100 ): # reject very small components
                  continue;
             color = CV_RGB(255,0,0);
             magnitude = 30.;
       silh_roi = cvGetSubRect(silh, comp_rect);
       mhi_roi = cvGetSubRect( mhi, comp_rect );
       orient_roi = cvGetSubRect( orient, comp_rect );
       mask_roi = cvGetSubRect( mask, comp_rect );
       angle = cvCalcGlobalOrientation( orient_roi, mask_roi, mhi_roi, timestamp, MHI_DURATION);
       angle = 360.0 - angle;  # adjust for images with top-left origin
       count = cvNorm( silh_roi, None, CV_L1, None ); # calculate number of points within silhouette ROI
```

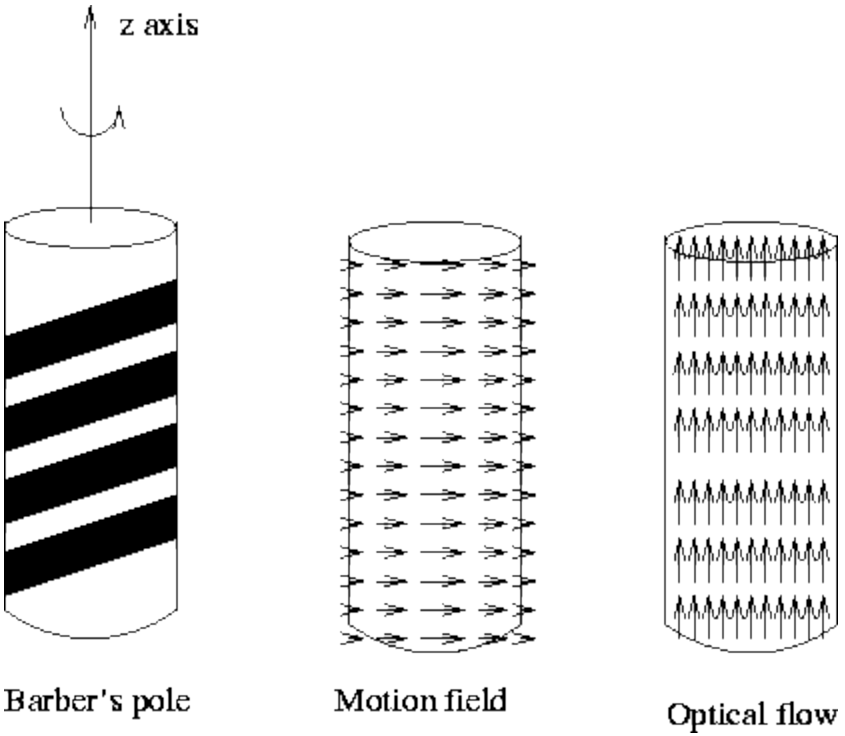
cvCalcMotionGradient

```
if( count < comp_rect.width * comp_rect.height * 0.05 ):
        continue;
center = cvPoint( (comp_rect.x + comp_rect.width/2),(comp_rect.y + comp_rect.height/2) );
cvCircle( dst, center, cvRound(magnitude*1.2), color, 3, CV_AA, 0 );
cvLine( dst, center, cvPoint( cvRound( center.x + magnitude*cos(angle*CV_PI/180)),
                cvRound( center.y - magnitude*sin(angle*CV_PI/180))), color, 3, CV_AA, 0 );


if __name__ == "__main__":
    motion = 0;
    capture = 0;

    if len(sys.argv)==1:
        capture = cvCreateCameraCapture( 0 )
    elif len(sys.argv)==2 and sys.argv[1].isdigit():
        capture = cvCreateCameraCapture( int(sys.argv[1]) )
    elif len(sys.argv)==2:
        capture = cvCreateFileCapture( sys.argv[1] );

    if not capture:
        print "Could not initialize capturing..."
        sys.exit(-1)

    cvNamedWindow( "Motion", 1 );
    while True:
        image = cvQueryFrame( capture );
        if( image ):
            if( not motion ):
                motion = cvCreateImage( cvSize(image.width,image.height), 8, 3 );
                cvZero( motion );
                motion.origin = image.origin;
            update_mhi( image, motion, 30 );
            cvShowImage( "Motion", motion );
```



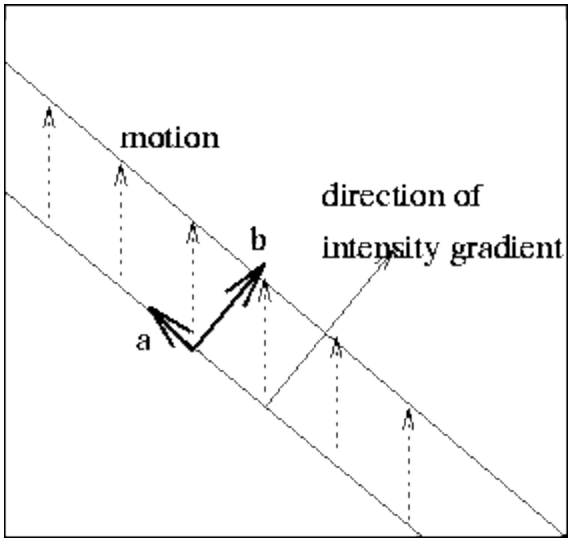
Viola, Jones, Snow: Detecting Pedestrians Using Patterns of Motion and Appearance, Springer 2005
> integrates image intensity information with motion information
> works even in bad weather (rain, snow)

Feature tracking:



Object motion, motion field and optical flow

Optical Flow



One can not measure the component of optical flow that is in the direction of the intensity gradient; the component (a) tangential to the intensity gradient is not available.

Feature tracking: Optical Flow

Optical flow is the apparent visual motion (motion of brightness patterns) perceivable as you (or a camera) moves through the world. Objects close by seem to move (backwards) quickly while distant objects move so slowly they appear still. (If you double the speed of travel, the perceived optic flow rate will double). Optical flow is also angle dependent; it is highest for objects at your side (90 degrees from the vector of motion). Objects directly in front of an observer appear flow-less. This flow-less center point is called the focus of expansion (FOE) and indicates the current direction of motion.

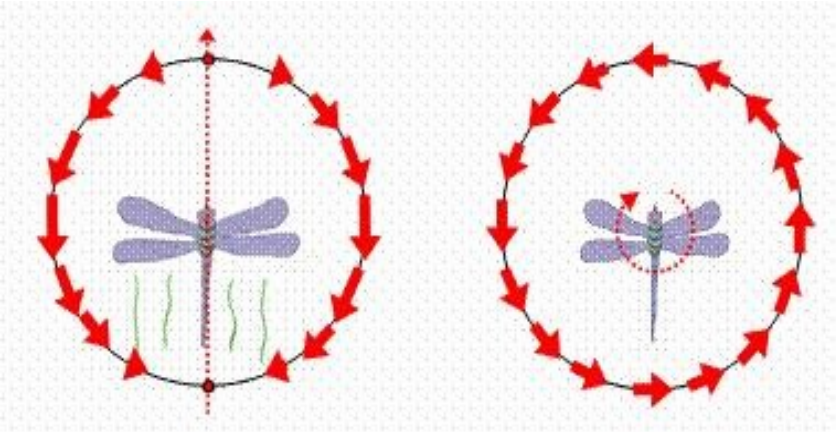
Feature tracking: Optical Flow

Optical flow in computer vision allows the assessment of motion between two frames without prior knowledge about the content of the frames. As opposed to dense optical flow, sparse optical flow specifies a subset of points to track (such as corners). Additionally, some approaches include tracking across image pyramids, starting with the highest level with lowest detail and working down the pyramid to finer details.

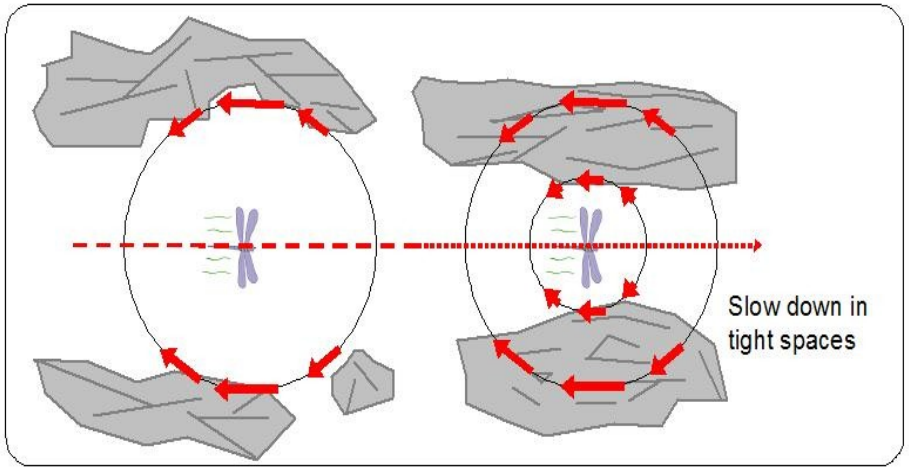
Optical flow has been used in many other contexts. For example, Hecht and Bauer of Gutenberg Universität Mainz refer to experiments in optical flow to describe the way convex rear view mirrors compromise distance and time to contact judgments (Ergonomics Vol. 50, No. 4, April 2007, 601–614).

Feature tracking:

Optical Flow



How optic flow relates to relative motion



How optic flow assists in navigating tight spaces

http://www.centeye.com/pages/techres/opticflow.html

Feature tracking: cvCalcOpticalFlowPyrLK

CvOpticalFlow

void cvCalcOpticalFlowLK(const CvArr* prev, const CvArr* curr, CvSize win_size, CvArr* velx, CvArr* vely);
Calculates optical flow for two images

The function cvCalcOpticalFlowLK computes flow for every pixel of the first input image using Lucas & Kanade algorithm [Lucas81].

Feature tracking: cvCalcOpticalFlow PyrLK

CvOpticalFlow

Basic assumptions of the LK algorithm:

Brightness constancy – the brightness of a pixel does not change from frame to frame -> dI/dt = 0

Temporal persistence – the image motion of a surface patch changes slowly in time

Spatial coherence – neighboring points in a scene belong to the same surface, have similar motion

CvOpticalFlow – core elements

```
While (true)
{
frame = cvQueryFrame( input_video );

optical_flow_termination_criteria = cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 );

cvGoodFeaturesToTrack(frame1_1C, eig_image, temp_image, frame1_features, &number_of_features, .01, .01, NULL);

cvCalcOpticalFlowPyrLK(frame1_1C, frame2_1C, pyramid1, pyramid2, frame1_features, frame2_features,
number_of_features, optical_flow_window, 5, optical_flow_found_feature, optical_flow_feature_error,
optical_flow_termination_criteria, 0 );
}
```

CvOpticalFlow

```
#include <stdio.h>
#include <cv.h>
#include <highgui.h>
#include <math.h>
static const double pi = 3.14159265358979323846;
#define NUMBEROFSHITOMASIFEATURES 400

inline static double square(int a)
{
   return a * a;
}

inline static void allocateOnDemand( IplImage **img, CvSize size, int depth, int channels )
{
   if ( *img != NULL ) return;
   *img = cvCreateImage( size, depth, channels );
   if ( *img == NULL )
   {
      fprintf(stderr, "Error: Couldn't allocate image.  Out of memory?\n");
      exit(-1);
   }
}

#define NUMSHOTCUTBINS 100
#define SHOTRATIO 0.7
#define MINMOVEMENT 0.1

bool global_toggle_shot_detect = false;
double global_shot_cut_detect[NUMSHOTCUTBINS];
double global_previous_shot_cut_total = 0;
double global_pixel_count = 0;
double global_pixel_zero_count = 0;
```

CvOpticalFlow

```
void ShotDetectPerPixel(CvPoint a, CvPoint b)
{
   double angle; angle = atan2( (double) a.y - b.y, (double) a.x - b.x );
   double hypotenuse;  hypotenuse = sqrt( square(a.y - b.y) + square(a.x - b.x) );
   if (hypotenuse == 0)
                  global_pixel_zero_count++;
   global_pixel_count++;
   int index = (int)( (pi+angle) / (2*pi/NUMSHOTCUTBINS) );
   if (global_shot_cut_detect[index]<hypotenuse)
      global_shot_cut_detect[index]=hypotenuse;
}

bool ShotCutDetect()
{
   if (global_toggle_shot_detect == false)
      return false;

   double accumilator=0;
   for (int i = 0 ; i < NUMSHOTCUTBINS ; i++)
      accumilator += ( global_shot_cut_detect[i]*global_shot_cut_detect[i] );

   double metric = abs(accumilator - global_previous_shot_cut_total)/global_previous_shot_cut_total;
   if (global_pixel_zero_count/global_pixel_count>(1-MINMOVEMENT))
   {
      global_previous_shot_cut_total = accumilator;
      return false;
   }

   if (metric > SHOTRATIO)
   {
      global_previous_shot_cut_total = accumilator;
      return true;
   }
   global_previous_shot_cut_total = accumilator;
   return false;
}
```

CvOpticalFlow

```
int main(int argc , char *argv[])
{
   CvCapture *input_video;
   if(argv[1] == NULL)
      input_video = cvCaptureFromFile("optical_flow_input.avi");

   IplImage *velx, *vely, *velX, *velY;
   CvSize blockSize = cvSize(4,4), shiftSize = cvSize(1,1), maxRange = cvSize(3,3); int type_of_of = 0;

   /* Read the video's frame size out of the AVI. */
   CvSize frame_size;
   frame_size.height =(int) cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_HEIGHT );
   frame_size.width = (int) cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_WIDTH );

   velY = cvCreateImage(cvSize(frame_size.width,frame_size.height),IPL_DEPTH_32F,1);
   velX = cvCreateImage(cvSize(frame_size.width,frame_size.height),IPL_DEPTH_32F,1);

   velx = cvCreateImage(cvSize(frame_size.width/blockSize.width,frame_size.height/blockSize.height),IPL_DEPTH_32F,1);
   vely = cvCreateImage(cvSize(frame_size.width/blockSize.width,frame_size.height/blockSize.height),IPL_DEPTH_32F,1);

   /* Determine the number of frames in the AVI. */
   long number_of_frames;
   /* Go to the end of the AVI (ie: the fraction is "1") */
   cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_AVI_RATIO, 1. );

   /* Now that we're at the end, read the AVI position in frames */
   number_of_frames = (int) cvGetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES );

   /* Return to the beginning */
   cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, 0. );

   /* Create a windows called "Optical Flow" for visualizing the output,change its size to match the output.*/
   cvNamedWindow("Optical Flow", CV_WINDOW_AUTOSIZE);
```

CvOpticalFlow

```
long current_frame = 0;
 while(true)
    {
       static IplImage *frame = NULL, *frame1 = NULL, *frame1_1C = NULL, *frame2_1C = NULL, *eig_image = NULL,
      *temp_image = NULL, *pyramid1 = NULL, *pyramid2 = NULL;

       cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, current_frame );

       /* Get the next frame of the video.
       * IMPORTANT!  cvQueryFrame() always returns a pointer to the _same_memory location.  So successive calls:
       * frame1 = cvQueryFrame(); frame2 = cvQueryFrame(); frame3 = cvQueryFrame();
       * will result in (frame1 == frame2 && frame2 == frame3) being true. The solution is to make a copy of the cvQueryFrame() output  */
       frame = cvQueryFrame( input_video );
       if (frame == NULL)
       {
           fprintf(stderr, "Error: Hmm. The end came sooner than we thought.\n");
           return -1;
       }

      /* Allocate another image if not already allocated. Image has ONE channel of color (ie: monochrome) with 8-bit "color" depth.
       * This is the image format OpenCV algorithms actually operate on (mostly).*/

       allocateOnDemand( &frame1_1C, frame_size, IPL_DEPTH_8U, 1 );

       /* Convert whatever the AVI image format is into OpenCV's preferred format. AND flip the image vertically. OpenCV reads in AVIs
       upside-down by default. */

       cvConvertImage(frame, frame1_1C, CV_CVTIMG_FLIP);

       /* We'll make a full color backup of this frame so that we can draw on it.
       * (It's not the best idea to draw on the static memory space of cvQueryFrame().) */

       allocateOnDemand( &frame1, frame_size, IPL_DEPTH_8U, 3 );
       cvConvertImage(frame, frame1, CV_CVTIMG_FLIP);
```

CvOpticalFlow

```
allocateOnDemand( &frame2_1C, frame_size, IPL_DEPTH_8U, 1 );
cvConvertImage(frame, frame2_1C, CV_CVTIMG_FLIP);

/* Shi and Tomasi Feature Tracking! */
/* Preparation: Allocate the necessary storage. */
allocateOnDemand( &eig_image, frame_size, IPL_DEPTH_32F, 1 );
allocateOnDemand( &temp_image, frame_size, IPL_DEPTH_32F, 1 );

/* Preparation: This array will contain the features found in frame 1. */
CvPoint2D32f frame1_features[NUMBEROFSHITOMASIFEATURES];

/* Preparation: BEFORE the function call this variable is the array size (or the maximum number of features to find).
AFTER the function call
* this variable is the number of features actually found.*/

int number_of_features = NUMBEROFSHITOMASIFEATURES;
* "frame1_1C" is the input image.
* "eig_image" and "temp_image" are just workspace for the algorithm.
* The first ".01" specifies the minimum quality of the features (based on the eigenvalues).  The second ".01" specifies the
 minimum Euclidean distance between features.  "NULL" means use the entire input image.  You could point to a part of
the image. WHEN THE ALGORITHM RETURNS:
* "frame1_features" will contain the feature points.
* "number_of_features" will be set to a value <= 400 indicating the number of feature points found. */
cvGoodFeaturesToTrack(frame1_1C, eig_image, temp_image, frame1_features, &number_of_features, .01, .01, NULL);
/* Pyramidal Lucas Kanade Optical Flow! */
/* This array will contain the locations of the points from frame 1 in frame 2. */
CvPoint2D32f frame2_features[NUMBEROFSHITOMASIFEATURES];

/* The i-th element of this array will be non-zero if and only if the i-th feature of
* frame 1 was found in frame 2. */
char optical_flow_found_feature[NUMBEROFSHITOMASIFEATURES];
/* The i-th element of this array is the error in the optical flow for the i-th feature  of frame1 as found in frame 2.  If the i-th
feature was not found (see the array above) */
float optical_flow_feature_error[NUMBEROFSHITOMASIFEATURES];
```

CvOpticalFlow

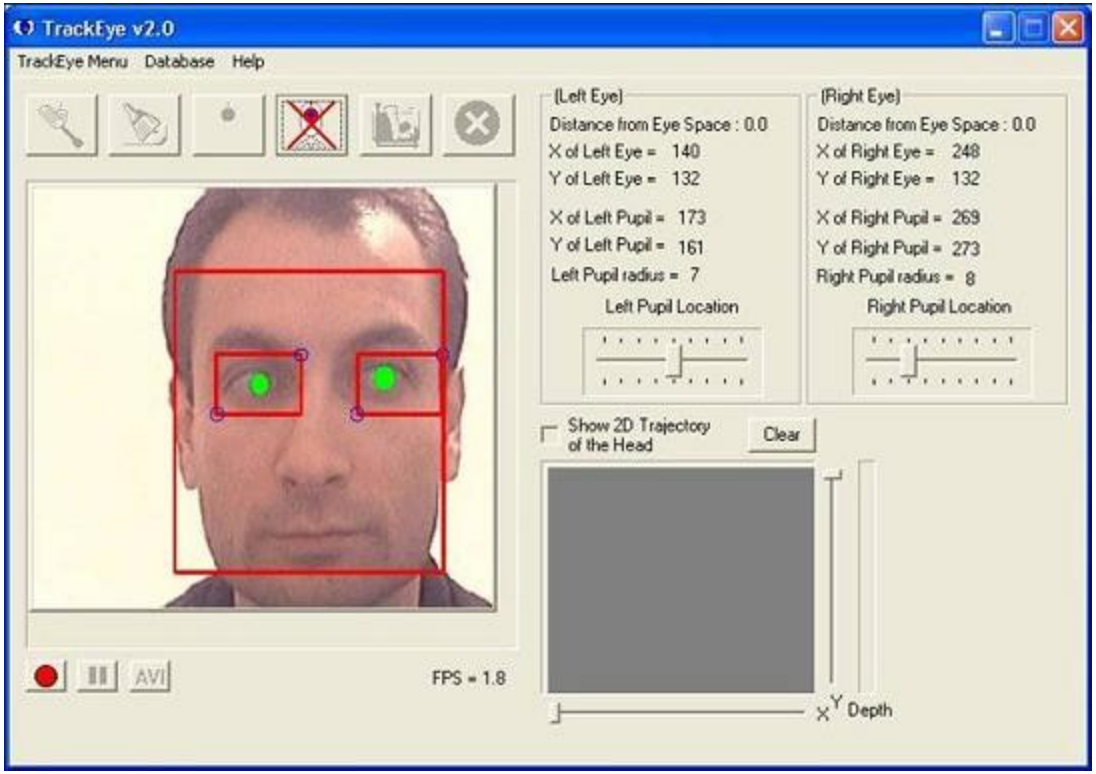
```
    /* This is the window size to use to avoid the aperture problem (see slide "Optical Flow: Overview"). */
     CvSize optical_flow_window = cvSize(5,5);

     /* This termination criteria tells the algorithm to stop when it has either done 20 iterations or when epsilon is better than .3.  You can
      play with these parameters for speed vs. accuracy but these values work pretty well in many situations.*/
     CvTermCriteria optical_flow_termination_criteria = cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 );

    //This is some workspace for the algorithm. (The algorithm actually carves the image into pyramids of different resolutions.)
     allocateOnDemand( &pyramid1, frame_size, IPL_DEPTH_8U, 1 );
     allocateOnDemand( &pyramid2, frame_size, IPL_DEPTH_8U, 1 );

     cvCalcOpticalFlowPyrLK(frame1_1C, frame2_1C, pyramid1, pyramid2, frame1_features, frame2_features, number_of_features,
                optical_flow_window, 5, optical_flow_found_feature, optical_flow_feature_error, optical_flow_termination_criteria, 0 );

     for (int i = 0 ; i < NUMSHOTCUTBINS ; i ++)
        global_shot_cut_detect[i] = 0.0;

     global_pixel_count = 0;
     global_pixel_zero_count = 0;
     cvConvertImage(frame1_1C,frame1);
     CvPoint p0,p1;

     //LINE DRAWING CODE OMMITTED HERE...
     bool shotDetect = ShotCutDetect();
     if (shotDetect)
                cvWaitKey(-1);
     cvShowImage("Optical Flow", frame1);
     current_frame++;
     if (current_frame < 0)
            current_frame = 0;
     if (current_frame >= number_of_frames - 1)
            break;
}
cvReleaseCapture(&input_video);
```

CvOpticalFlow

```
    bool shotDetect = ShotCutDetect();

    if (shotDetect)
        cvWaitKey(-1);

    cvShowImage("Optical Flow", frame1);
    /* And wait for the user to press a key (so the user has time to look at the image).
    * If the argument is 0 then it waits forever otherwise it waits that number of milliseconds.
    * The return value is the key the user pressed.
    */
    int key_pressed;
    key_pressed = cvWaitKey(40);

    /* If the users pushes "b" or "B" go back one frame.
    * Otherwise go forward one frame.
    */

    if (key_pressed == 'q' || key_pressed == 'Q')
        break;//    exit(0);
    else if (key_pressed == 'p' || key_pressed == 'P')
        cvWaitKey(0);

    current_frame++;

    if (current_frame < 0)
            current_frame = 0;
    if (current_frame >= number_of_frames - 1)
            break;

}
cvReleaseCapture(&input_video);
```

A game:
http://www.youtube.com/watch?v=ddNvNJXwYxU

failure:
http://www.youtube.com/watch?v=oCsdU7xGCpI



Dork Attack !

Camshift version + template matching version :

http://www.codeproject.com/KB/cpp/TrackEye.aspx?
display=PrintAll&fid=1403145&df=90&mpp=25&noise=3&sort=Position&view=Quick&select=2823443