The OpenCV Library: Computing Optical Flow

David Stavens
Stanford Artificial Intelligence Lab

I received some requests...

- ...on what to cover tonight:

  - “Perhaps you could do one of the 14 projects in the course? In front of us. In one hour.”

    -- Anonymous
Tonight we’ll code:

A fully functional sparse optical flow algorithm!

Plan

- OpenCV Basics
  - What is it?
  - How do we get started?

- Feature Finding and Optical Flow
  - A brief mathematical discussion.

- OpenCV Implementation of Optical Flow
  - Step by step.
What is OpenCV?

- Really four libraries in one:
  - “CV” – Computer Vision Algorithms
    - All the vision algorithms.
  - “CVAUX” – Experimental/Beta
    - Useful gems :-)
  - “CXCORE” – Linear Algebra
    - Raw matrix support, etc.
  - “HIGHGUI” – Media/Window Handling
    - Read/write AVIs, window displays, etc.
- Created/Maintained by Intel

Installing OpenCV

- Download from:
  - [http://sourceforge.net/projects/opencvlibrary/](http://sourceforge.net/projects/opencvlibrary/)

- Be sure to get the August 2004 release:
  - “Beta 4” for Windows XP/2000
  - “Beta 4” or “0.9.6” for Linux

- Windows version comes with an installer.
- Linux:
  - `gunzip opencv-0.9.6.tar.gz`;
  - `tar -xvf opencv-0.9.6.tar`
  - `cd opencv-0.9.6`;
  - `./configure --prefix=/usr; make`
  - `make install` [as root]
Tell Visual Studio where the includes are.

Tell Visual Studio to link against cxcore.lib, cv.lib, and highgui.lib.
Tell Visual Studio to disable managed extensions.

Plan

- OpenCV Basics
  - What is it?
  - How do we get started?

- Feature Finding and Optical Flow
  - A brief mathematical discussion.

- OpenCV Implementation of Optical Flow
  - Step by step.
Optical Flow: Overview

- Given a set of points in an image, find those same points in another image.
- Or, given point \([u_x, u_y]^T\) in image \(I_1\) find the point \([u_x + \delta_x, u_y + \delta_y]^T\) in image \(I_2\) that minimizes \(\varepsilon\):

\[
\varepsilon(\delta_x, \delta_y) = \sum_{x=-w_x}^{u_x+w_x} \sum_{y=-w_y}^{u_y+w_y} (I_1(x, y) - I_2(x + \delta_x, y + \delta_y))
\]

- (the \(\Sigma/w\)'s are needed due to the aperture problem)

Optical Flow: Utility

- Tracking points ("features") across multiple images is a fundamental operation in many computer vision applications:
  - To find an object from one image in another.
  - To determine how an object/camera moved.
  - To resolve depth from a single camera.
  - ...or stereo.

- ~ 75% of this year’s CS 223b projects.

- But what are good features to track?
Finding Features: Overview

- Intuitively, a good feature needs at least:
  - Texture (or ambiguity in tracking)
  - Corner (or aperture problem)
- But what does this mean formally?

\[
\begin{bmatrix}
\sum_{\text{neighborhood}} \left( \frac{\partial I}{\partial x} \right)^2 & \sum_{\text{neighborhood}} \frac{\partial^2 I}{\partial x \partial y} \\
\sum_{\text{neighborhood}} \frac{\partial^2 I}{\partial x \partial y} & \sum_{\text{neighborhood}} \left( \frac{\partial I}{\partial y} \right)^2
\end{bmatrix}
\]

- A good feature has big eigenvalues, implies:
  - Texture
  - Corner

- Shi/Tomasi. Intuitive result really part of motion equation. High eigenvalues imply reliable solvability. Nice!

Plan

- OpenCV Basics
  - What is it?
  - How do we get started?

- Feature Finding and Optical Flow
  - A brief mathematical discussion.

- OpenCV Implementation of Optical Flow
  - Step by step.
So now let’s code it!

- Beauty of OpenCV:
  - All of the Above = Two Function Calls
  - Plus some support code :-)

- Let’s step through the pieces.

- These slides provide the high-level.
  - Full implementation with extensive comments:
    - http://robotics.stanford.edu/~dstavens/cs223b

Step 1: Open Input Video

CvCapture *input_video =
    cvCaptureFromFile("filename.avi");

- Failure modes:
  - The file doesn’t exist.
  - The AVI uses a codec OpenCV can’t read.
    - Codecs like MJPEG and Cinepak are good.
    - DV, in particular, is bad.
Step 2: Get A Video Frame

cvQueryFrame( input_video );

- This is a hack so that we can look at the internals of the AVI. OpenCV doesn’t allow us to do that correctly unless we get a video frame first.

Step 3: Read AVI Properties

CvSize frame_size;
frame_size.height = cvGetCaptureProperty( input_video, CV_CAP_PROP_FRAME_HEIGHT );

- Similar construction for getting the width and the number of frames.
  - See the handout.
Step 4: Create a Window

`cvNamedWindow(“Optical Flow”, CV_WINDOW_AUTOSIZE);`

- We will put our output here for visualization and debugging.

Step 5: Loop Through Frames

- Go to frame N:
  `cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, N );`

- Get frame N:
  `IplImage *frame = cvQueryFrame(input_video);`
  - Important: `cvQueryFrame` always returns a pointer to the same location in memory.
Step 6: Convert/Allocate

- Convert input frame to 8-bit mono:
  IplImage *frame1 =
  cvCreateImage( cvSize(width, height),
      IPL_DEPTH_8U, 1);
  cvConvertImage( frame, frame1 );

- Actually need third argument to conversion: CV_CVTIMG_FLIP.

Step 7: Run Shi and Tomasi

CvPoint2D32f frame1_features[N];
cvGoodFeaturesToTrack(
    frame1, eig_image, temp_image,
    frame1_features, &N, .01, .01, NULL);

- Allocate eig,temp as in handout.
- On return frame1_features is full and N is the number of features found.
Step 8: Run Optical Flow

```c
char optical_flow_found_feature[];
float optical_flow_feature_error[];
CvTermCriteria term =
    cvTermCriteria( CV_TERMCRIT_ITER |
                    CV_TERMCRIT_EPS, 20, .3 );

cvCalcOpticalFlowPyrLK( ... );
```

- 13 arguments total. All of the above.
- Both frames, both feature arrays, etc.
- See full implementation in handout.

Step 9: Visualize the Output

```c
CvPoint p, q;
p.x = 1; p.y = 1; q.x = 2; q.y = 2;
CvScalar line_color;
line_color = CV_RGB(255, 0, 0);
int line_thickness = 1;

cvLine(frame1, p,q, line_color, line_thickness, CV_AA, 0);
cvShowImage("Optical Flow", frame1);
```

- CV_AA means draw the line antialiased.
- 0 means there are no fractional bits.
Step 10: Make an AVI output

CvVideoWriter *video_writer =
    cvCreateVideoWriter( “output.avi”,
-1, frames_per_second, cvSize(w,h) );
† (“-1” pops up a nice GUI.)

    cvWriteFrame(video_writer, frame);
      Just like cvShowImage(window, frame);

    cvReleaseVideoWriter(&video_writer);

Let’s watch the result:
(The Stanford Roadrunner Robot.)
That’s the first step for...

Stavens, Lookingbill, Lieb, Thrun; CS223b 2004; ICRA 2005

A few closing thoughts...

- I’ve really described the lowest level.

- Your projects will begin with code like this and then you’ll implement something that’s not in OpenCV.

- OpenCV is good for non-vision things too.

- Feel free ask questions!
  - dstavens@robotics.stanford.edu or Gates 226

- Good luck!! 223b projects are fun :-)

---------------------------------------------------
/* --Sparse Optical Flow Demo Program--
 * Written by David Stavens (dstavens@robotics.stanford.edu)
 */
#include <stdio.h>
#include <cv.h>
#include <highgui.h>
#include <math.h>

static const double pi = 3.14159265358979323846;

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

*/

static inline 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);
    }
}

int main(void)
{
    /* Create an object that decodes the input video stream. */
    CvCapture *input_video = cvCaptureFromFile("C:\Documents and Settings\David Stavens\Desktop\223B-Demo\optical_flow_input. avi");
    if (input_video == NULL)
    {
        /* Either the video didn't exist OR it uses a codec OpenCV doesn't support. */
        fprintf(stderr, "Error: Can't open video.\n");
        return -1;
    }

    /* This is a hack. If we don't call this first then getting capture properties (below) won't work right. This is an OpenCV bug. We ignore the return value here. But it's actually a video frame. */
    cvQueryFrame(input_video);

    /* 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 );

    /* 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 three windows called "Frame N", "Frame N+1", and "Optical Flow"
* for visualizing the output. Have those windows automatically change their
* size to match the output.
*/
cvNamedWindow("Optical Flow", CV_WINDOW_AUTOSIZE);

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;

    /* Go to the frame we want. Important if multiple frames are queried in
    * the loop which they of course are for optical flow. Note that the very
    * first call to this is actually not needed. (Because the correct position
    * is set outside the for() loop.)
    */
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)
    {
        /* Why did we get a NULL frame? We shouldn't be at the end. */
        fprintf(stderr, "Error: Hmm. The end came sooner than we thought.\n");
        return -1;
    }

    /* Allocate another image if not already allocated.
    * Image has ONE challenge 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. Flip is a shameless hack. OpenCV reads
    // in AVIs upside-down by default. (No comment :-)
    /*
    */
    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( &frame2_1C, frame_size, IPL_DEPTH_8U, 1 );
    cvConvertImage(frame, frame2_1C, CV_CVTIMG_FLIP);

    /* Get the second frame of video. Sample principles as the first. */
    frame = cvQueryFrame( input_video );
    if (frame == NULL)
    {
        /* Error: Hmm. The end came sooner than we thought. */
        return -1;
    }

    allocateOnDemand( &temp_image, frame_size, IPL_DEPTH_8U, 1 );
    cvConvertImage(frame1, temp_image, CV_CVTIMG_FLIP);
    cvConvertImage(frame2, temp_image, CV_CVTIMG_FLIP);
    cvConvertImage(temp_image, frame1, CV_CVTIMG_FLIP);
    cvSetCaptureProperty( input_video, CV_CAP_PROP_POS_FRAMES, current_frame );
    /* Shi and Tomasi Feature Tracking! */
allocateOnDemand( &eig_image, frame_size, IPL_DEPTH_32F, 1 );
allocateOnDemand( &temp_image, frame_size, IPL_DEPTH_32F, 1 );

int number_of_features;

number_of_features = 400;

cvGoodFeaturesToTrack(frame1, eig_image, temp_image, frame1_features, & number_of_features, .01, .01, NULL);

cvPoint2D32f frame2_features[400];

char optical_flow_found_feature[400];

float optical_flow_feature_error[400];

CvSize optical_flow_window = cvSize(3,3);

CvTermCriteria optical_flow_termination_criteria
    = cvTermCriteria( CV_TERMCRIT_ITER | CV_TERMCRIT_EPS, 20, .3 );
/* This is some workspace for the algorithm. */
allocateOnDemand( &pyramid1, frame_size, IPL_DEPTH_8U, 1 );
allocateOnDemand( &pyramid2, frame_size, IPL_DEPTH_8U, 1 );

/* Actually run Pyramidal Lucas Kanade Optical Flow!! */
* "frame1_IC" is the first frame with the known features.
* "frame2_IC" is the second frame where we want to find the first frame's features.
* "pyramid1" and "pyramid2" are workspace for the algorithm.
* "frame1_features" are the features from the first frame.
* "frame2_features" is the (outputted) locations of those features in the second frame.
* "number_of_features" is the number of features in the frame1_features array.
* "optical_flow_window" is the size of the window to use to avoid the aperture problem.
* "5" is the maximum number of pyramids to use. 0 would be just one level.
* "optical_flow_found_feature" is as described above (non-zero iff feature found by the flow).
* "optical_flow_feature_error" is as described above (error in the flow for this feature).
* "optical_flow_termination_criteria" is as described above (how long the algorithm should look).
* "0" means disable enhancements. (For example, the second array isn't pre-initialized with guesses.) */

cvCalcOpticalFlowPyrLK(frame1_IC, frame2_IC, 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 fun (and debugging :)), let's draw the flow field. */
for(int i = 0; i < number_of_features; i++)
{
    /* If Pyramidal Lucas Kanade didn't really find the feature, skip it. */
    if ( optical_flow_found_feature[i] == 0 ) continue;

    int line_thickness;        line_thickness = 1;
    /* CV_RGB(red, green, blue) is the red, green, and blue components of the color you want, each out of 255. */
    CvScalar line_color;       line_color = CV_RGB(255,0,0);

    /* Let's make the flow field look nice with arrows. */

    /* The arrows will be a bit too short for a nice visualization because of the high framerate */
    /* (ie: there's not much motion between the frames). So let's lengthen them by a factor of 3. */
    CvPoint p,q;
    p.x = (int) frame1_features[i].x;
    p.y = (int) frame1_features[i].y;
    q.x = (int) frame2_features[i].x;
    q.y = (int) frame2_features[i].y;

    double angle;           angle = atan2( (double) p.y - q.y, (double) p.x - q.x );
    double hypotenuse;      hypotenuse = sqrt( square(p.y - q.y) + square(p.x - q.x) );

    /* Here we lengthen the arrow by a factor of three. */
    q.x = (int) (p.x - 3 * hypotenuse * cos(angle));
    q.y = (int) (p.y - 3 * hypotenuse * sin(angle));

    /* Now we draw the main line of the arrow. */
/* "frame1" is the frame to draw on.
 * "p" is the point where the line begins.
 * "q" is the point where the line stops.
 * "CV_AA" means antialiased drawing.
 * "0" means no fractional bits in the center coordinate or radius.
 */
cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
/* Now draw the tips of the arrow. I do some scaling so that the
 * tips look proportional to the main line of the arrow.
 */
p.x = (int) ( q.x + 9 * cos(angle + pi / 4) );
p.y = (int) ( q.y + 9 * sin(angle + pi / 4) );
cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
p.x = (int) ( q.x + 9 * cos(angle - pi / 4) );
p.y = (int) ( q.y + 9 * sin(angle - pi / 4) );
cvLine( frame1, p, q, line_color, line_thickness, CV_AA, 0 );
}
/* Now display the image we drew on. Recall that "Optical Flow" is the name of
 * the window we created above.
 */
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(0);
/* If the users pushes "b" or "B" go back one frame.
 * Otherwise go forward one frame.
 */
if (key_pressed == 'b' || key_pressed == 'B') current_frame--;
else current_frame++;
/* Don't run past the front/end of the AVI. */
if (current_frame < 0) current_frame = 0;
if (current_frame >= number_of_frames - 1) current_frame = number_of_frames - 2;
}