unesco
University Network
Member

Ministry of
Education

CascadeClassifier

Sung Soo Hwang

Face Detection

- Example

```
CascadeClassifier face_classifier;

face_classifier.load("haarcascade_frontalface_alt.xml");
```

  - OPENCV provides various classifiers (eye, upper body, etc.) in "…₩opencv₩sources₩data₩haarcascades"

Face Detection

- OpenCV function

```
void cv::CascadeClassifier::detectMultiScale ( InputArray          image,
                                               std::vector< Rect > & objects,
                                               std::vector< int > & numDetections,
                                               double              scaleFactor = 1.1,
                                               int                 minNeighbors = 3,
                                               int                 flags = 0,
                                               Size                minSize = Size(),
                                               Size                maxSize = Size()
```

- **Image :** Matrix of the type CV_8U containing an image where objects are detected.
- **Objects :** Vector of rectangles where each rectangle contains the detected object, the rectangles may be partially outside the original image.

Face Detection

- openCV function
  - **numDetections :** Vector of detection numbers for the corresponding objects. An object's number of detections is the number of neighboring positively classified rectangles that were joined together to form the object.
  - **scaleFactor** : Parameter specifying how much the image size is reduced at each image scale.
  - **minNeighbors** **:** Parameter specifying how many neighbors each candidate rectangle should have to retain it.
  - **flags** : Parameter with the same meaning for an old cascade as in the function cvHaarDetectObjects. It is not used for a new cascade.
  - **minSize :** Minimum possible object size. Objects smaller than that are ignored.
  - **maxSize :** Maximum possible object size. Objects larger than that are ignored. If maxSize ==minSize, model is evaluated on single scale.

Face Detection

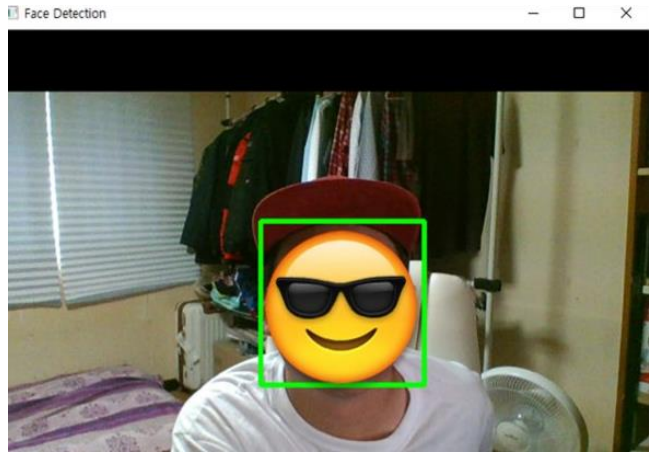
- Example code

```
CascadeClassifier face_classifier;
Mat frame, grayframe;
vector<Rect> faces;
int i;

// open the webcam
VideoCapture cap(0);

// check if we succeeded
if (!cap.isOpened()) {
		cout << "Could not open camera" << endl;
		return -1;
}

// face detection configuration
face_classifier.load("haarcascade_frontalface_alt.xml");

while (true) {
		// get a new frame from webcam
		cap >> frame;

		// convert captured frame to gray scale
		cvtColor(frame, grayframe, COLOR_BGR2GRAY);
```

Face Detection

- Example code

```
                face_classifier.detectMultiScale(
                        grayframe,
                        faces,
                        1.1, // increase search scale by 10% each pass
                        3,   // merge groups of three detections
                        0,   // not used for a new cascade
                        Size(30, 30) //minimum size for detection
                );

                // draw the results
                for (i = 0; i < faces.size(); i++) {
                        Point lb(faces[i].x + faces[i].width, faces[i].y + faces[i].height);
                        Point tr(faces[i].x, faces[i].y);
                        rectangle(frame, lb, tr, Scalar(0, 255, 0), 3, 4, 0);
                }
                // print the output
                imshow("Face Detection", frame);
                if (waitKey(33) == 27) break; // ESC
}
```



unesco
University Network
Member

Ministry of
Education

Tracking using meanShift and camShift

Sung Soo Hwang

Tracking

- Example code

```
#include <opencv2/opencv.hpp>
#include <iostream>

using namespace std;
using namespace cv;

struct CallbackParam
{
	Mat frame;
	Point pt1, pt2;
	Rect roi;
	bool drag;
	bool updated;
};

// if click and drag mouse on the window
void onMouse(int event, int x, int y, int flags, void* param)
{
	CallbackParam *p = (CallbackParam *)param;
	// clicked
	if (event == EVENT_LBUTTONDOWN){
		p->pt1.x = x;
		p->pt1.y = y;
		p->pt2 = p->pt1;
		p->drag = true;
	}
```

Tracking

- Example code

```
        // unclicked
        if (event == EVENT_LBUTTONUP){
                int w = x - p->pt1.x;
                int h = y - p->pt1.y;
                p->roi.x = p->pt1.x;
                p->roi.y = p->pt1.y;
                p->roi.width = w;
                p->roi.height = h;
                p->drag = false;
                if (w >= 10 && h >= 10){
                        p->updated = true;
                }
        }
        // clicked and moving
        if (p->drag && event == EVENT_MOUSEMOVE){
                if (p->pt2.x != x || p->pt2.y != y){
                        Mat img = p->frame.clone(); // copy paused image
                        p->pt2.x = x; // update
                        p->pt2.y = y; // dst point
                        rectangle(img, p->pt1, p->pt2, Scalar(0, 255, 0), 1);
                        imshow("Tracker", img);
                }
        }
}
```

Tracking

- Example code

```
int main(int argc, char *argv[]){
	// open the webcam
	VideoCapture cap(0);
	CallbackParam param;
	Mat frame, m_backproj, hsv;
	Mat m_model3d;
	Rect m_rc;
	// for HSV colorspace
	float hrange[] = { 0,180 }; // Hue
	float srange[] = { 0,255 }; // Saturation
	float vrange[] = { 0,255 }; // Brightness

	const float* ranges[] = { hrange, srange, vrange }; // hue, saturation, brightness
	int channels[] = { 0, 1, 2 };
	int hist_sizes[] = { 16, 16, 16 };

	// check if we succeeded
	if (!cap.isOpened()){
		cout << "can't open video file" << endl;
		return 0;
	}

	// click and drag on image to set ROI
	cap >> frame;
	imshow("Tracker", frame);
	param.frame = frame;
	param.drag = false;
	param.updated = false;
```

Tracking

- Example code

```
// if mouse event is occured, it calls onMouse function
setMouseCallback("Tracker", onMouse, &param);

bool tracking = false;
while (true){
        // image acquisition & target init
        if (param.drag){
                if (waitKey(33) == 27) break; // ESC key
                continue;
        }
        // convert image from RGB to HSV
        cvtColor(frame, hsv, COLOR_BGR2HSV);
        if (param.updated){
                Rect rc = param.roi;
                Mat mask = Mat::zeros(rc.height, rc.width, CV_8U);
                ellipse(mask, Point(rc.width / 2, rc.height / 2), Size(rc.width / 2, rc.height / 2), 0, 0, 360, 255, CV_FILLED);
                Mat roi(hsv, rc);
                //histogram calculation
                calcHist(&roi,          // The source array(s)
                        1,              // The number of source arrays
                        channels,       // The channel (dim) to be measured.
                        mask,           // A mask to be used on the source array (zeros indicating pixels to be ignored)
                        m_model3d, // The Mat object where the histogram will be stored
                        3,              // The histogram dimensionality.
                        hist_sizes,     // The number of bins per each used dimension
                        ranges          // The range of values to be measured per each dimension
                );
```

Tracking

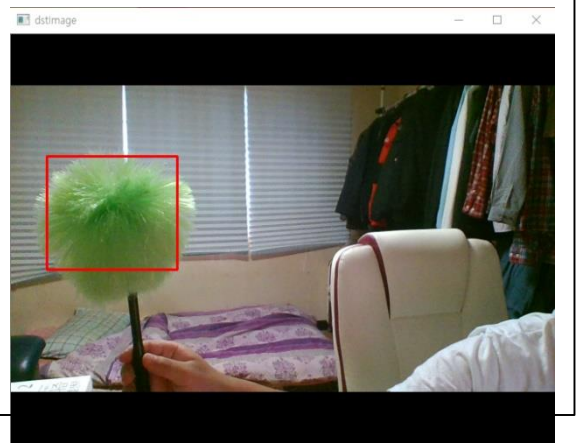
- Example code

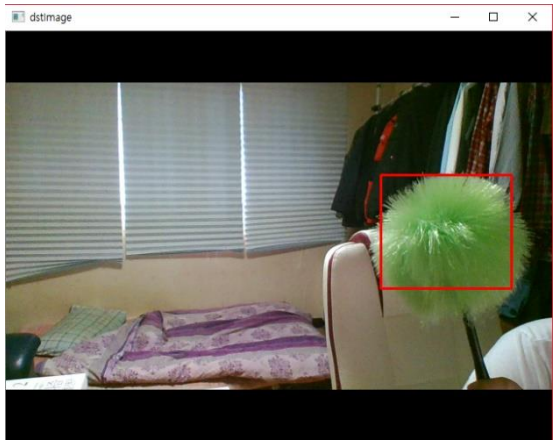
```
            m_rc = rc;
            param.updated = false;
            tracking = true;
    }

    // get a new frame from webcam
    cap >> frame;
    if (frame.empty()) break; // if there are no frames to read, quit.

    // image processing
    if (tracking){
            // histogram backprojection.
            // all the arguments are known (the same as used to calculate the histogram),
            // only we add the backproj matrix,
            // which will store the backprojection of the source image (&hue)
            calcBackProject(&hsv, 1, channels, m_model3d, m_backproj, ranges);
```

Meanshift

- Example code

```
                    // tracking[meanShift]
                    // obtain a window with maximum pixel distribution
                    meanShift(m_backproj, // dst
                            m_rc, // initial location of window
                            TermCriteria(TermCriteria::EPS | TermCriteria::COUNT, 10, 1) // termination criteria
                    );
                    rectangle(frame, m_rc, Scalar(0, 0, 255), 3);
            }
            // image display
            imshow("Tracker", frame);

            // user input
            char ch = waitKey(33);
            if (ch == 27) break; // ESC Key (exit)
            else if (ch == 32){ // SPACE Key (pause video)
                    // exit from while loop if user input is SPACE or ESC key
                    while ((ch = waitKey(33)) != 32 && ch != 27);

                    // if user input is not SPACE but ESC, quit program.
                    if (ch == 27) break;
            }
        }
        return 0;
}
```

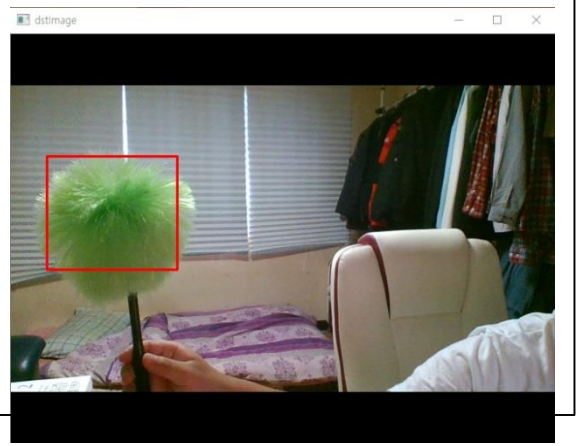


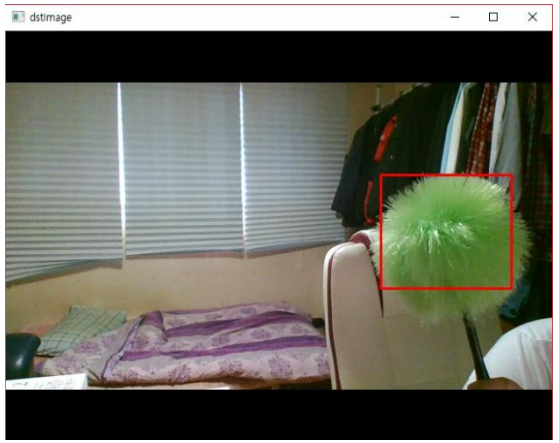


Camshift

- Example code

```
                    // tracking [CamShift]
                    // rotated rectangle(result) and
                    // box parameters(used to be passed as search window in next iteration)
                    CamShift(m_backproj, // dst
                            m_rc, // initial location of window
                            cvTermCriteria(TermCriteria::EPS | TermCriteria::COUNT, 20,1) // termination criteria
                    );
                    rectangle(frame, m_rc, Scalar(0, 0, 255), 3);
                }
                // image display
                imshow("Tracker", frame);

                // user input
                char ch = waitKey(33);
                if (ch == 27) break; // ESC Key (exit)
                else if (ch == 32){ // SPACE Key (pause video)
                    // exit from while loop if user input is SPACE or ESC key
                    while ((ch = waitKey(33)) != 32 && ch != 27);
                    // if user input is not SPACE but ESC, quit program.
                    if (ch == 27) break;
                }
        }
        return 0;
}
```





Optical Flow

Sung Soo Hwang

Optical Flow

- KLT algorithm with pyramids
  - Extract features first
    - Use goodFeaturesToTrack function

```
goodFeaturesToTrack(prevImage, prevPoints, maxCorners, qualityLevel, minDistance, Mat(), blockSize, useHarrisDetector, k);
```

    - Perform tracking of the extracted features

```
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 );
```

Optical Flow

- Example code

```
#include <iostream>
#include <opencv2/opencv.hpp>

using namespace std;
using namespace cv;

struct feature {
        Point2f pt;
        int val;
};
bool initialization = false;
void DrawTrackingPoints(vector<Point2f> &points, Mat &image);

int main(int argc, char *argv[]){
        // open the webcam
        VideoCapture cap(0);
        if (!cap.isOpened()) {
                cout << "Cannot open cap" << endl;
                return 0;
        }
        double fps = cap.get(CV_CAP_PROP_FPS);
        Mat currImage, prevImage;
        Mat frame, dstImage;

        double qualityLevel = 0.01;
        double minDistance = 10;
        int blockSize = 3;
        bool useHarrisDetector = false;
        double k = 0.04;
        int maxCorners = 500;
```

Optical Flow

- Example code

```
TermCriteria criteria = TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, // The type of termination criteria
                                     10, // The maximum number of iterations or elements to compute.
                                     0.01 // The desired accuracy or change in parameters at which the iterative algorithm stops.
                                     );
        Size winSize(11, 11);

        vector<Point2f> prevPoints;
        vector<Point2f> currPoints;
        vector<Point2f> boundPoints;

        int delay = 1000 / fps;
        int nframe = 0;

        while(1) {
                // get a new frame
                cap >> frame;
                if (frame.empty()) break;
                // copy the source image
                frame.copyTo(dstImage);
                // convert image from RGB to GRAY scale
                cvtColor(dstImage, currImage, CV_BGR2GRAY);
                // conver image using filter(Gaussian Blur)
                GaussianBlur(currImage, currImage, Size(5, 5), 0.5);
```

Optical Flow

- Example code

```
// feature detection
if (initialization) {

        // Determines strong corners on an image.
        // The function finds the most prominent corners in the image or in the specified image region
        goodFeaturesToTrack(prevImage, // Input 8-bit or floating-point 32-bit, single-channel image.
                            prevPoints, // Output vector of detected corners.
                            maxCorners, // Maximum number of corners to return.
                            qualityLevel, // Parameter characterizing the minimal accepted quality of image corners
                            minDistance, // Minimum possible Euclidean distance between the returned corners.
                            Mat(), // Optional region of interest
                            blockSize, // Size of an average block for computing a derivative covariation matrix over each pixel neighborhood
                            useHarrisDetector, // Parameter indicating whether to use a Harris detector (see cornerHarris) or cornerMinEigenVal.
                            k // Free parameter of the Harris detector.
        );
        // Refines the corner locations.
        // The function iterates to find the sub-pixel accurate location of corners or radial saddle points
        cornerSubPix(prevImage, // Input single-channel, 8-bit or float image.
                        prevPoints, // Initial coordinates of the input corners and refined coordinates provided for output.
                        winSize, // Half of the side length of the search window
                        Size(-1, -1), // Half of the size of the dead region in the middle of the search zone over which the summation in the formula below is not done
                        criteria // Criteria for termination of the iterative process of corner refinement.
        );
        // drwa circles
        DrawTrackingPoints(prevPoints, dstImage);
        initialization = false;
}
```

Optical Flow

- Example code

```
if (prevPoints.size() > 0) {
        vector<Mat> prevPyr, currPyr;
        Mat status, err;

        // Constructs the image pyramid which can be passed to calcOpticalFlowPyrLK.
        buildOpticalFlowPyramid(prevImage, // 8-bit input image.
                                prevPyr, // output pyramid.
                                winSize, // window size of optical flow algorithm.
                                3, // 0-based maximal pyramid level number.
                                true // set to precompute gradients for the every pyramid level
        );
        buildOpticalFlowPyramid(currImage, // 8-bit input image.
                                currPyr, // output pyramid.
                                winSize, // window size of optical flow algorithm.
                                3, // 0-based maximal pyramid level number.
                                true // set to precompute gradients for the every pyramid level
        );

        // Calculates an optical flow for a sparse feature set using the iterative Lucas-Kanade method with pyramids.
        calcOpticalFlowPyrLK(   prevPyr, // first 8-bit input image or pyramid constructed by buildOpticalFlowPyramid.
                                currPyr, // second input image or pyramid of the same size and the same type as prevImg.
                                prevPoints, // vector of 2D points for which the flow needs to be found
                                currPoints, // output vector of 2D points (with single-precision floating-point coordinates)
                                status, // output status vector (of unsigned chars)
                                err, // output vector of errors
                                winSize // size of the search window at each pyramid level.
        );
```

Optical Flow

- Example code

```
                    //delete invalid correspondinig points
                    for (int i = 0; i < prevPoints.size(); i++) {
                        if (!status.at<uchar>(i)) {
                            prevPoints.erase(prevPoints.begin() + i);
                            currPoints.erase(currPoints.begin() + i);
                        }
                    }

                    // function for drawing detected corners
                    DrawTrackingPoints(currPoints, dstImage);
                    prevPoints = currPoints;
                }

                // print the output
                imshow("dstImage", dstImage);
                // copy the filtered image
                currImage.copyTo(prevImage);

                // user input
                int ch = waitKey(33);
                if (ch == 27) break; // 27 == ESC key
                if (ch == 32) initialization = true; // 32 == Space key
            }

            return 0;
}
```

Optical Flow

- Example code

```
void DrawTrackingPoints(vector<Point2f> &points, Mat &image) {
        // Draw corners detected
        for (int i = 0; i < points.size(); i++) {
                int x = cvRound(points[i].x);
                int y = cvRound(points[i].y);
                circle(image, Point(x, y), 3, Scalar(255, 0, 0), 2);
        }
}
```