Tracking

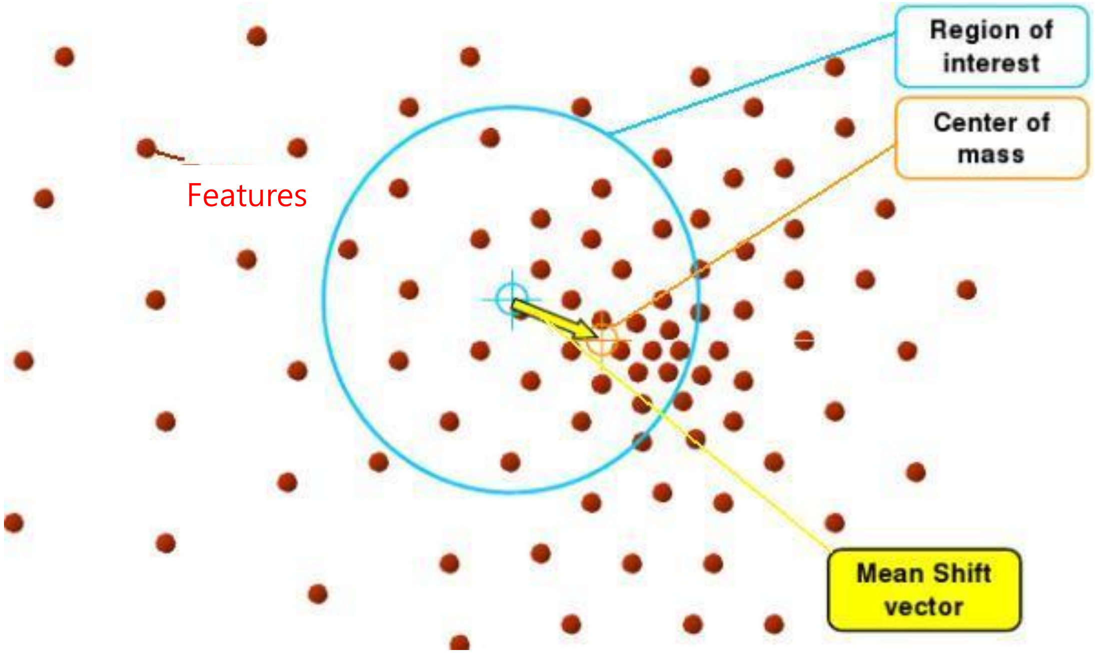
Sung Soo Hwang

Introduction

- Basic concept
  - First, a ROI is selected by user-interaction or detection
  - Represent the ROI with histograms or features
  - Find the best matching patch to the ROI at the next frame

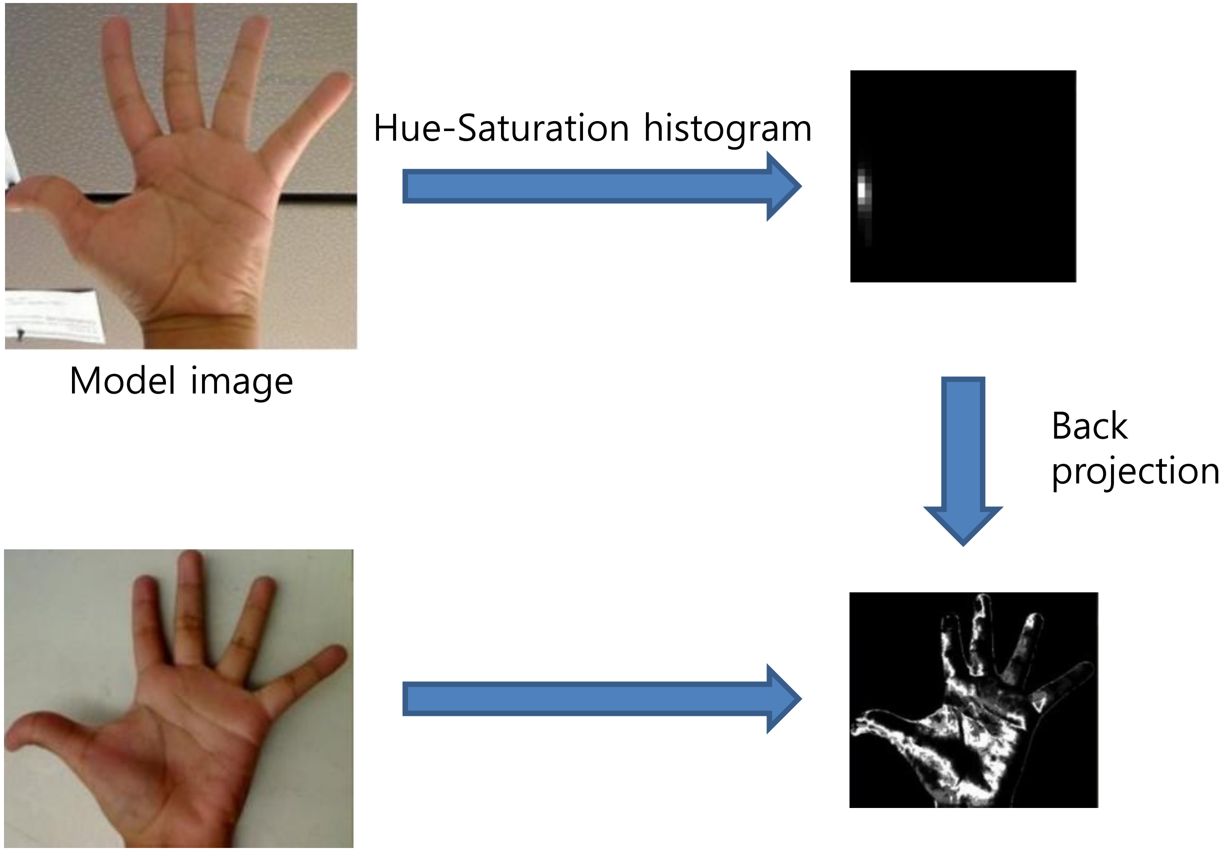
Meanshift

- It is a procedure for locating the maxima of a density function given discrete data sampled from that function
- It is an iterative method



Meanshift

- Histogram back-projection



Hue-Saturation histogram

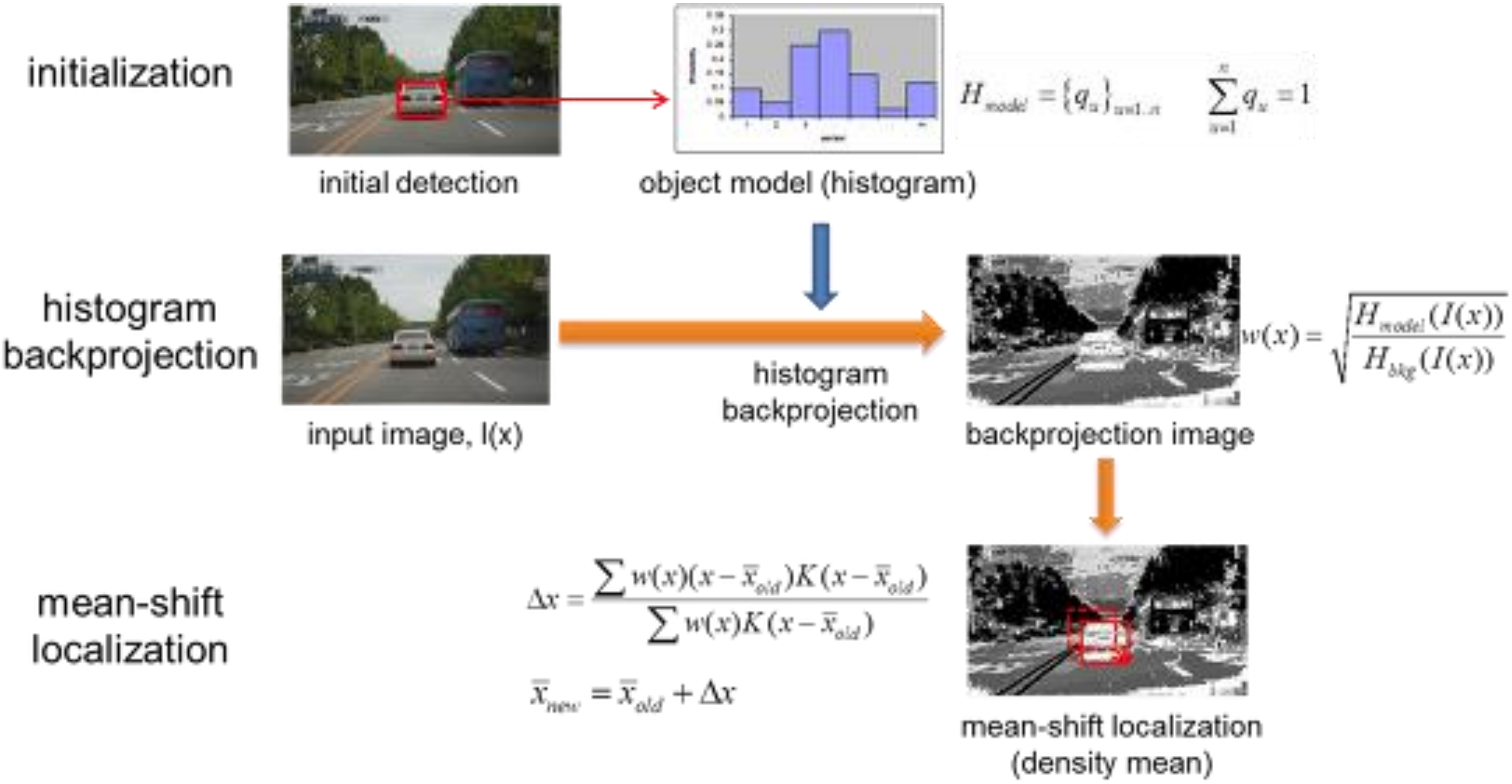
Model image

Back
projection

Target image

Meanshift

- Tracking using mean shift



initialization

initial detection

object model (histogram)

$$H_{model} = \{q_u\}_{u=1..n} \qquad \sum_{u=1}^{n} q_u = 1$$

histogram backprojection

input image, I(x)

histogram backprojection

backprojection image

$$w(x) = \sqrt{\frac{H_{model}(I(x))}{H_{bkg}(I(x))}}$$

mean-shift localization

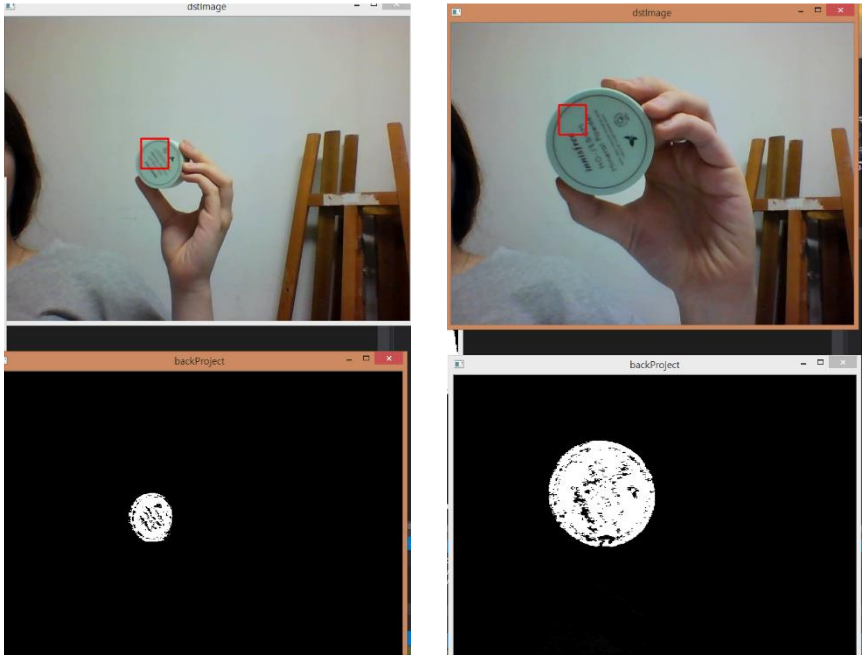
$$\Delta x = \frac{\sum w(x)(x - \bar{x}_{old})K(x - \bar{x}_{old})}{\sum w(x)K(x - \bar{x}_{old})}$$

$$\bar{x}_{new} = \bar{x}_{old} + \Delta x$$

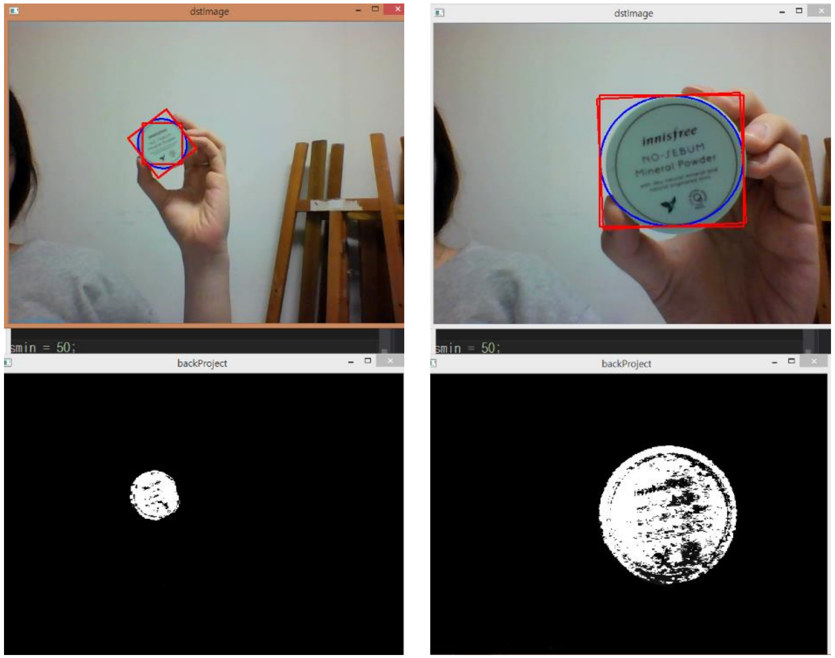
mean-shift localization (density mean)

Camshift

- Mean shift
  - Modified version of mean-shift
  - The size of search window can be changed



Mean-shift



```
smin = 50;
```

```
smin = 50;
```

Cam shift

Meanshift

- Example code

```
struct CallbackParam
{
        Mat frame;
        Point pt1, pt2;
        Rect roi;
        bool drag;
        bool updated;
};
void onMouse(int event, int x, int y, int flags, void* param)
{
        CallbackParam *p = (CallbackParam *)param;
        if (event == EVENT_LBUTTONDOWN){
                p->pt1.x = x;
                p->pt1.y = y;
                p->pt2 = p->pt1;
                p->drag = true;
        }
        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;
                }
        }
```

Meanshift

- Example code

```
    if (p->drag && event == EVENT_MOUSEMOVE){
        if (p->pt2.x != x || p->pt2.y != y){
            Mat img = p->frame.clone();
            p->pt2.x = x;
            p->pt2.y = y;
            rectangle(img, p->pt1, p->pt2, Scalar(0, 255, 0), 1);
            imshow("Tracker", img);
        }
    }
}

int main(int argc, char *argv[]){
    VideoCapture cap(0);
    CallbackParam param;
    Mat frame, m_backproj, hsv;
    Mat m_model3d;
    Rect m_rc;
    float hrange[] = { 0,180 };
    float vrange[] = { 0,255 };
    const float* ranges[] = { hrange, vrange, 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;
    }
```

Meanshift

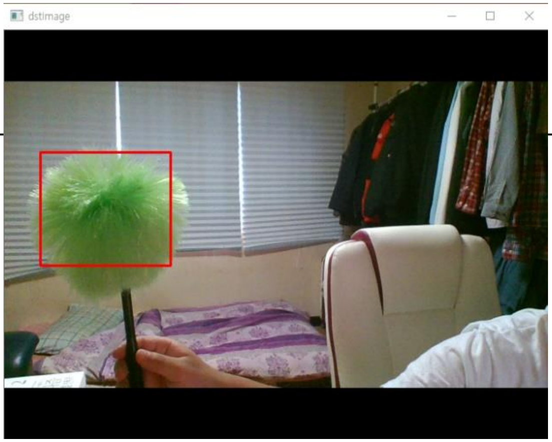
- Example code

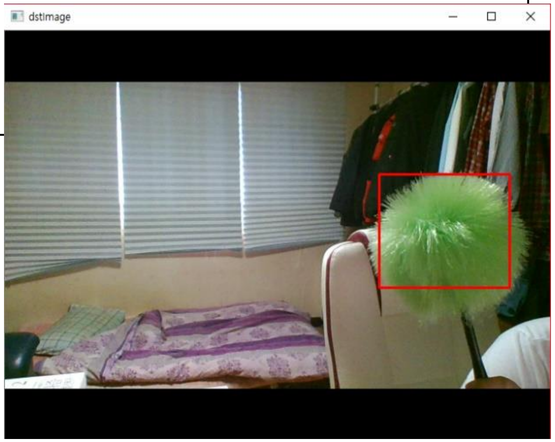
```
// click and drag on image to set ROI
cap >> frame;
imshow("Tracker", frame);
param.frame = frame;
param.drag = false;
param.updated = false;
setMouseCallback("Tracker", onMouse, &param);

bool tracking = false;
while (true){
        // image acquisition & target init
        if (param.drag){
                if (waitKey(33) == 27) break; // ESC key
                continue;
        }
        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);
                Mat roi(hsv, rc);
                calcHist(&roi, 1, channels, mask, m_model3d, 3, hist_sizes, ranges);
                m_rc = rc;
                param.updated = false;
                tracking = true;
        }
        cap >> frame;
        if (frame.empty()) break;
```

Meanshift

- Example code

```
        // image processing
        if (tracking){
            //histogram backprojection
            calcBackProject(&hsv, 1, channels, m_model3d, m_backproj, ranges);
            //tracking
            meanShift(m_backproj, m_rc, TermCriteria(TermCriteria::EPS | TermCriteria::COUNT, 10, 1));
            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)
            while ((ch = waitKey(33)) != 32 && ch != 27);
            if (ch == 27) break;
        }
    }
    return 0;
}
```

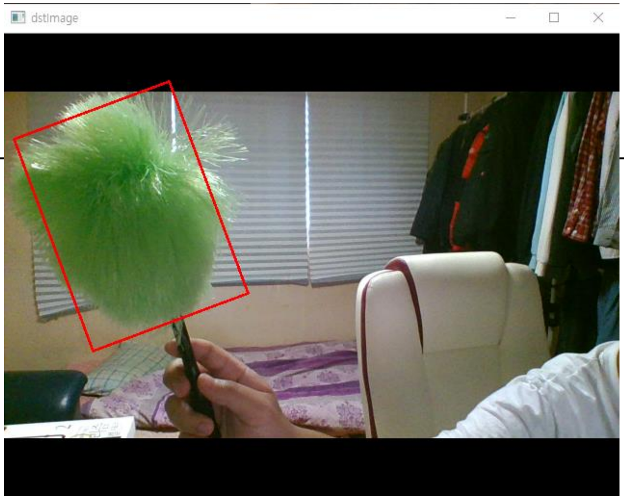


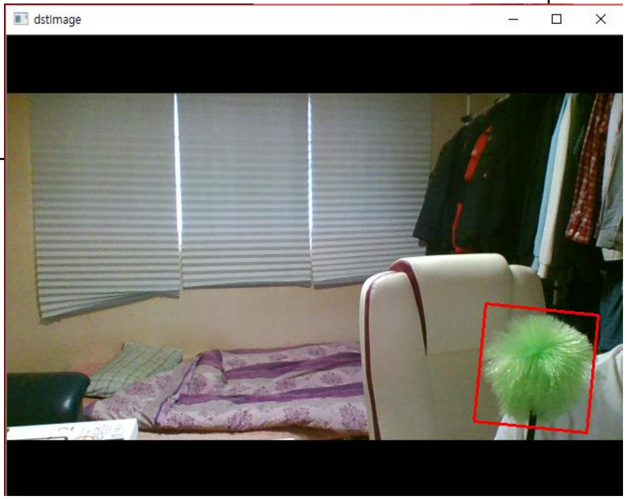


Camshift

- Example code

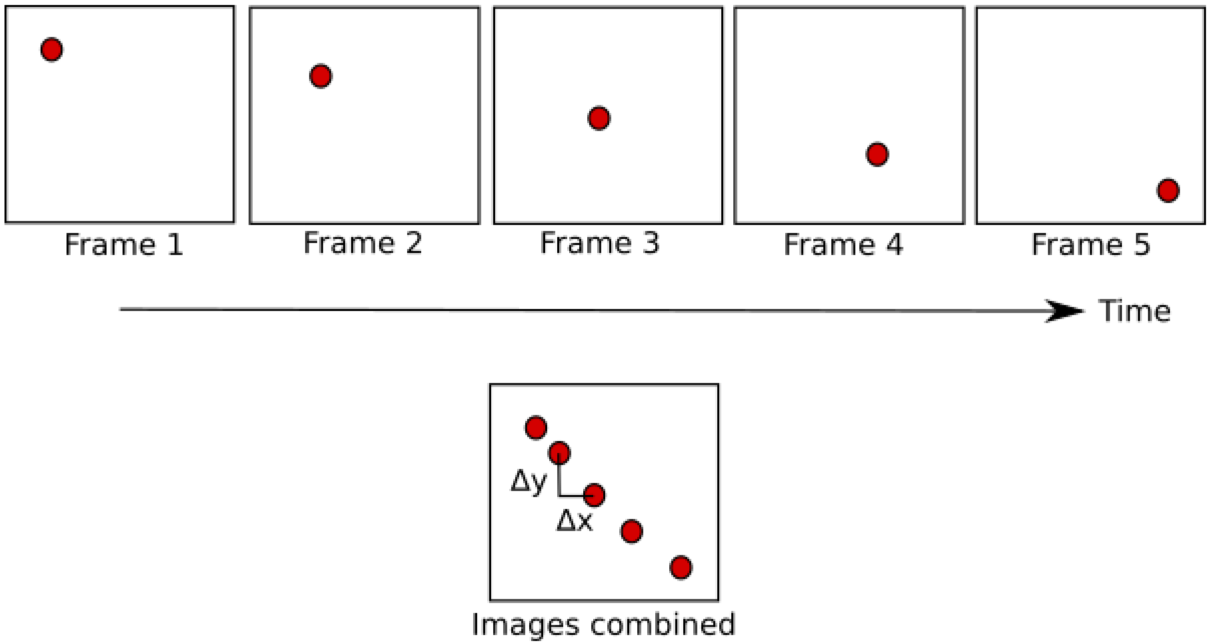
```
        // image processing
        if (tracking){
            //histogram backprojection
            calcBackProject(&hsv, 1, channels, m_model3d, m_backproj, ranges);
            //tracking
            CamShift(m_backproj, m_rc, cvTermCriteria(TermCriteria::EPS | TermCriteria::COUNT, 20,1));
            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)
            while ((ch = waitKey(33)) != 32 && ch != 27);
            if (ch == 27) break;
        }
    }
    return 0;
}
```

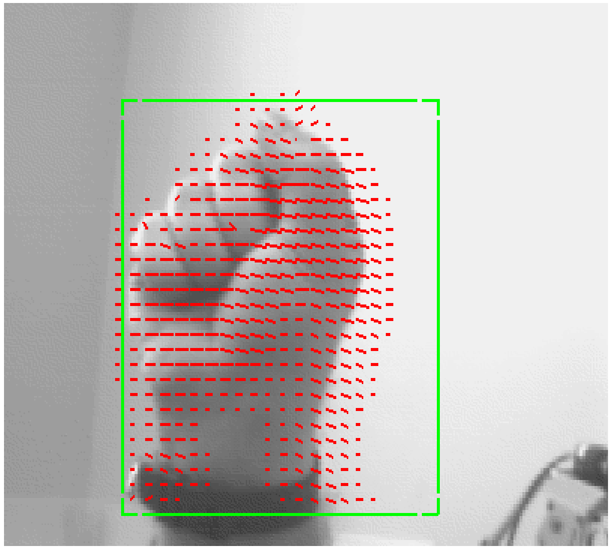




Optical Flow

- Optical flow is the apparent motion of brightness patterns in the image





Optical Flow

- KLT algorithm
  - Assumption
    - Intensity of objects are not changed over consecutive frames
    - Movement of pixels are similar to that of adjacent pixels

  ➔ $I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t)$

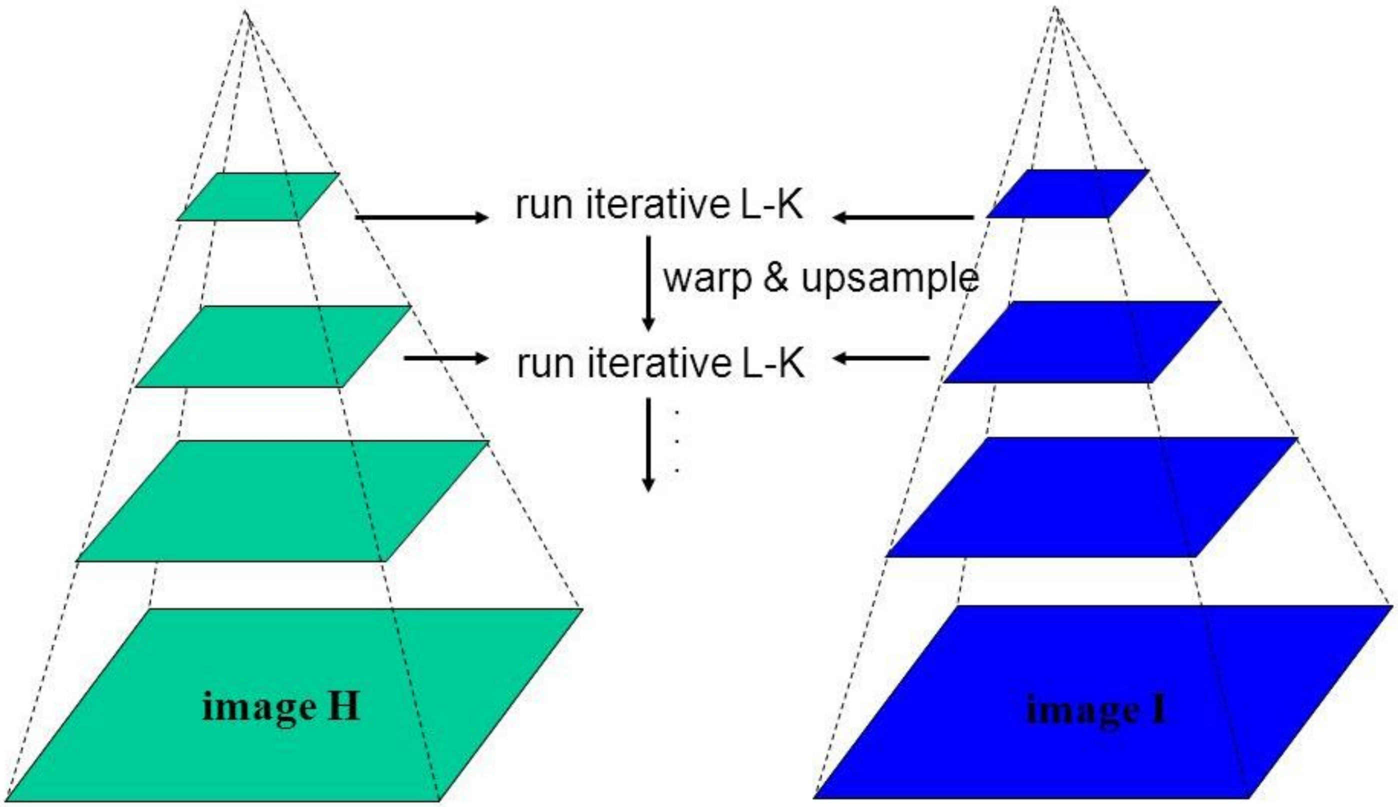
    - By applying Taylor series
      - $I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t$

      ➔ $\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0$

    - Extract features first and track the extracted features

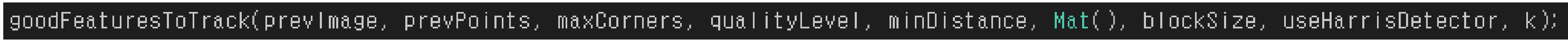
Optical Flow

- KLT algorithm with pyramids
  - Original KLT algorithm cannot handle large movement
  - To overcome this limitation, image pyramid is used



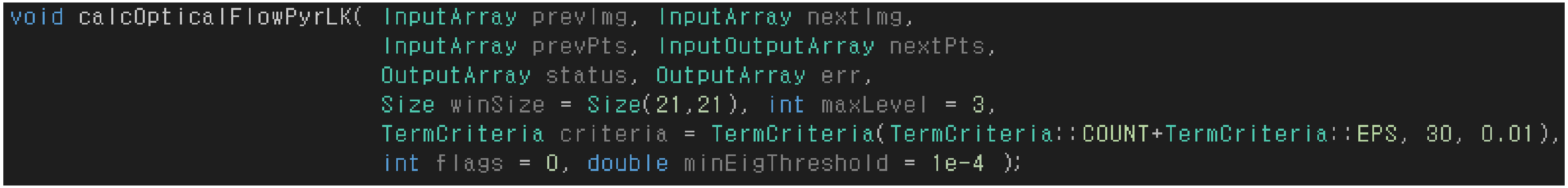
Optical Flow

- 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

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

int main(int argc, char *argv[])
{
        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;

        TermCriteria criteria = TermCriteria(TermCriteria::COUNT + TermCriteria::EPS, 10, 0.01);
        Size winSize(11, 11);

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

Optical Flow

- Example code

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

while(1) {
    cap >> frame;
    if (frame.empty()) break;
    frame.copyTo(dstImage);
    /// Copy the source image
    cvtColor(dstImage, currImage, CV_BGR2GRAY);
    GaussianBlur(currImage, currImage, Size(5, 5), 0.5);

    //feature detection
    if (initialization) {
        goodFeaturesToTrack(prevImage, prevPoints, maxCorners, qualityLevel, minDistance, Mat(), blockSize,
        useHarrisDetector, k);
        cornerSubPix(prevImage, prevPoints, winSize, Size(-1, -1), criteria);
        DrawTrackingPoints(prevPoints, dstImage);
        initialization = false;}

    if (prevPoints.size() > 0) {
        vector<Mat> prevPyr, currPyr;
        Mat status, err;
        buildOpticalFlowPyramid(prevImage, prevPyr, winSize, 3, true);
        buildOpticalFlowPyramid(currImage, currPyr, winSize, 3, true);
        calcOpticalFlowPyrLK(prevPyr, currPyr, prevPoints, currPoints, status, err, winSize);
        //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);
            }
        }
```

Optical Flow

- Example code

```
                    DrawTrackingPoints(currPoints, dstImage);
                    prevPoints = currPoints;
            }

            imshow("dstImage", dstImage);
            currImage.copyTo(prevImage);

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


    return 0;
}

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);
    }
}
```