COMPUTER VISION NOTES
OBJECT LOCALISATION AND TRACKING

MY PERSONAL NOTES ON

OBJECT LOCALISATION TECHNIQUES; COLOUR MATCHING, MEAN SHIFT TRACKING, OPTICAL FLOW, LUKAS KANADE

BY

0xLeo (github.com/0xleo)

APRIL 11, 2019

DRAFT X.YY
MISSING: . . .

Contents

1 Histogram-Based Methods

1.1 Histogram backprojection

1.1.1 Intuition - model and search image histogram

In image processing, we are usually interested in histograms of greyscale images. However, often the colour histogram can be used to identify an image region or object. RGB histograms are practically not good enough for matching as the R, G, B components are strongly correlation with the illumination hitting the object. In practice, objects are converted from RGB to HSV (Hue, Saturation, Value) domain. Hue represents the colour type (blue, yellow, etc.), saturation represents the vibrancy (how vivid or neutral it is) and value represents the brightness of the colour. Hence HSV decouples the brightness from the colour description. Therefore when performing colour matching we are only interested in the H and S components, which map to a 2D histogram. More about the HSV domain in A.1.

> ☞ **Comment:** The HS components are often but *not always* a good choice for colour-based detection. They may fail detecting black and white objects since black and white can have any colour (H) and in this case the SV components of the HSV or even the YUV domain are a better choice. However, in this article we stick to HS.

Histogram backprojection answers the question "where in the image are the colours that belong to the object being looked for?". We do this by defining a model image (the object we search for – a. k.a. target) and the search (the whole image where we search in), probing the model over search image and calculating their histogram similarity at each position.

Just to illustrate the idea, assume that we want to match the greyscale (instead of the 2D) histogram of the garlic in Fig. 1. A part of the top garlic has been chosen as the model. The histogram of the model is shown as well as that of two matching candidates. In this case, the histogram of "match 2" is more similar to the model's than one "match 1" so we want somehow to register that similarity. The question attempted to be answered in the next section is "how do we measure the similarity of the histogram of the matching candidate to that of the model?".

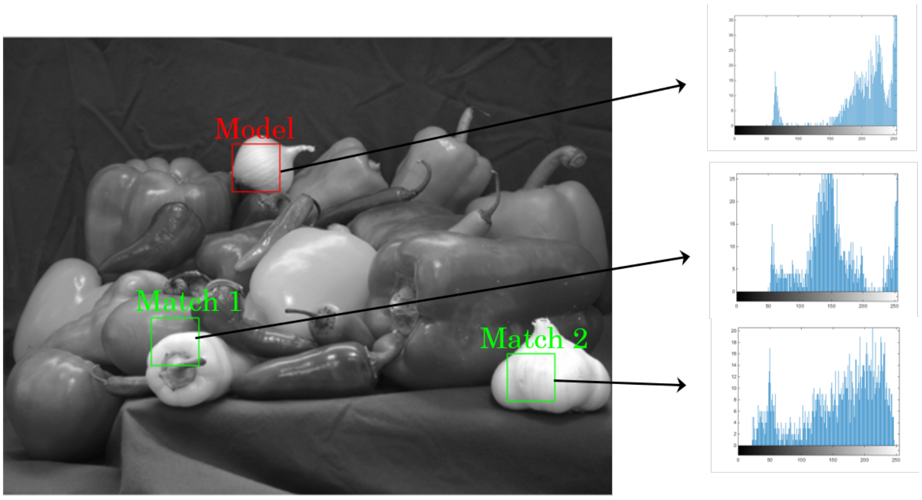


Fig. 1. Model and two matches' greyscale histograms.

1.1.2 (optional) The rationale behind defining the ratio histogram

We assume that

1. the model's histogram (the object we search for) is narrow and tall
2. and the scene's (whole input image) histogram is rather wide.

It has then been proven (it won't be discussed in this article how as the how is out of scope) that a good histogram similarity measure between the model M and a patch of the image we search it in I is the ratio histogram R. M and I need to be pre-calculated and can be divided element-wise, since they have the same range and bins to obtain R. The ratio histogram is a therefore function $R : \mathbb{Z}^2 \to \mathbb{R}$ that maps a colour (h,s) to some value. If $M(h,s) > I(h,s) \Rightarrow R(h,s) > 1$ that means the model has more pixels of colour (h,s)

relative to its total number of pixels compared to $I(h,s)$, and vice versa. If $R(h,s) = 1$, then that means that the input and model images contain (h,s) at the same degree.

However, because of assumptions (1), (2), $M(h,s) > I(h,s)$ can happen quite often and so long as $M(h,s) > I(h,s) \Rightarrow R(h,s) > 1$, e.g. $R(h,s) = 2,3,10$, the exact value does not give much useful information. To summarise, R de-emphasises pixels with colours that do not belong to the model and emphasises the rest.

1.1.3 A colour matching heuristic

From the previous section, the conclusion is that it is desirable to clip the ratio histogram to 1, as defined by Swain et al.

Definition 1.1. *For each bin j, the ratio histogram is defined as*

$$R_j = \min\left(\frac{M_j}{I_j}, 1\right) \tag{1.1}$$

, where M, I are the model's and input's histograms respectively.

Note that j does not necessarily have to be a pair (h,s), but it if a histogram bin (index) is quantised it can be a rectangle in the 2D space, such as $[20,39] \times [50,69]$. As mentioned before, R associates a colour with its probability of appearing in the model and the next step is the associate each pixel with that probability.

Each pixel of the original image at (x,y) maps to a 2D HS value, by a colour function $c : \mathbb{Z}^2 \leftarrow \mathbb{Z}^2$, by taking $c(x,y)$. Sometimes need an intermediate function $h : \mathbb{Z}^2 \rightarrow \mathbb{Z}^2$ that takes the output of c and quantises it (groups multiple colours in one bin), before it is fed to R. For example, h could convert $[0,1,\ldots,179] \times [0,1,\ldots,255]$ to $[0,19,39,\ldots,179] \times [0,24,49,\ldots,255]$. The output of h is bed to R, which divides M_j to I_j at each bin j.To summarise this paragraph we have defined the following functions in backpropagation:

- $c : \mathbb{Z}^2 \rightarrow \mathbb{Z}^2$: maps a pixel at (x,y) to an HS value (h_j, s_j).
- $h : \mathbb{Z}^2 \rightarrow \mathbb{Z}^2$ maps a set of values $(h_i, s_i, h_{i+1}, s_{i+1}, \ldots, h_n, s_n)$ to another (h,s) value by having quantised the range of h and s.
- $R : \mathbb{Z}^2 \rightarrow [0,1]$ maps an (h,s) value to a probablity.

We therefore want to create a new image b where each pixel (x,y) gets assigned its output of R - the measure of how much its colour appears in the model image.

$$b(x,y) := R\left(h\left(c(x,y)\right)\right) = \min\left(\frac{M\left(h\left(c(x,y)\right)\right)}{I\left(h\left(c(x,y)\right)\right)}, 1\right) \; \forall\, x,y \tag{1.2}$$

The final step is to find compact regions where b is high. If the shape of the object (model) to detect is generic, then this can be done by convolving b with binary disk mask D^r of radius r. Define:

$$D^r_{x,y} = \begin{cases} 1 & \sqrt{x^2+y^2} \le r \\ 0 & \text{otherwise} \end{cases} \tag{1.3}$$

Then the probability image b can be convolved with the mask:

$$b := D^r * b \tag{1.4}$$

The $\arg\max$ function to returns the pixel (x,y) with the maximum value of its argument, i.e. of the R matrix and the $*$ symbol denotes convolution. Then Histogram Backprojection algorithm can be then written

Algorithm 1 Colour matching by histogram backprojection according to Swain et al

procedure HIST-BACKPROJ(ImM, ImI) ▷ ImM: model, ImI: search image
  $M \leftarrow \text{histogram}(ImM)$
  $I \leftarrow \text{histogram}(ImI)$
  for each histogram bin j **do** ▷ a bin is a pair (h,s)
    $R_j = \min\left(\frac{M_j}{I_j}, 1\right)$ ▷ Divide element-wise
  $m \leftarrow rows(M)$
  $n \leftarrow cols(M)$
  $b \leftarrow empty_{m \times n}$
  for y in 0...m-1 **do**
    for x in 0...n-1 **do**
      $b_{x,y} \leftarrow R(h(c(x,y)))$ ▷ b matrix of colour probability
  $D^r \leftarrow$ binary disk of radius r
  $b \leftarrow D^r * b$ ▷ Group (by convolving) high probability pixels together.
  $x_{obj}, y_{obj} \leftarrow \arg\max_{x,y}(b)$
  return x_{obj}, y_{obj}

1.1.4 Histogram backprojection implementation from scratch

An implementation of Alg. 1 has been written in A.2. However, instead of finding the location of the object by the arg max function, it applies Otsu's threshold on the R matrix. This automatically selects a threshold T based on the statistics of the histogram of R for which if $R[x,y] < T$, then the pixel at (x,y) is classified as background, else as foreground. Instructions on how to run the implementation code are in A.2 and an output is shown below.

Fig. 2. Input image with a ROI of the objects to detect selected.

Fig. 3. Detected objects on the original image.

1.1.5 Histogram backprojection implementation using OpenCV's API

OpenCV implements the technique using the `cv2.calcBackProject(image, channels, histohram_array, channel_ranges, [scale = 1])` method (in Python). Its invocation looks like:
`cv2.calcBackProject(search_image, channels, model_histogram, channel_ranges, [scale = 1])`

- `search_image`: the input image, e.g. in HSV.
- `channels`: which channels of the original image and the model to select in order to draw its histogram, e.g. `channels = [0,1] -> H, S`.
- `model_histogram`: histogram of the model (ROI), needs to be pre-calculated.
- `channel_ranges`: set it to `[0,180,0,256]` to select the full range of H, S components.

The code listing in A.3 works similarly with the one in A.2, expecting two clicks from the user to define a bounding box around a sample of the object to detect. It also performs similarly on the same images, showing some black spots on roughly the same positions.

1.1.6 Histogram backprojection summary

✓ Fast – can easily be used in real time.

✓ Relatively immune to noise and illumination changes.

✓ Simple to implement.

✗ Not effective against non-compact objects.

✗ Does not use any knowledge about the shape or position of the detected object – only its colour.

1.2 Mean Shift Tracking

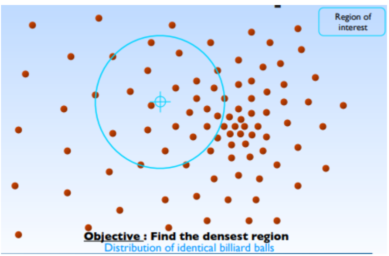
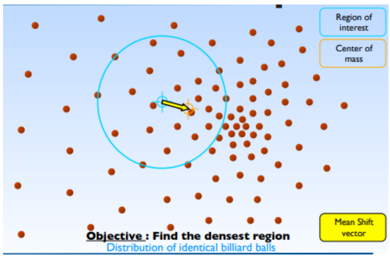
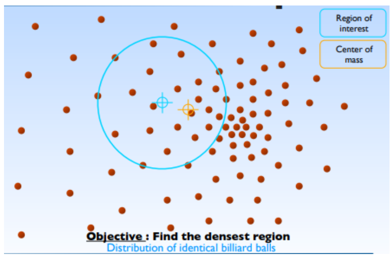
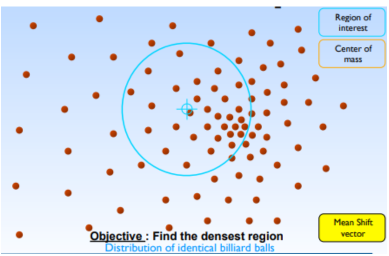
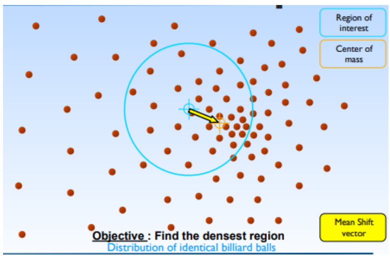
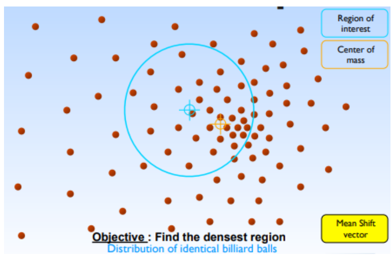
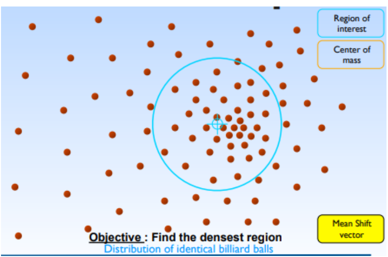
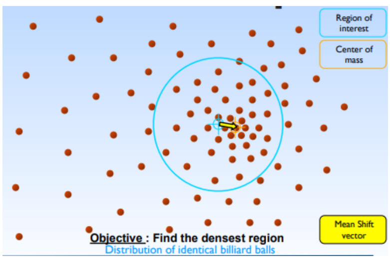
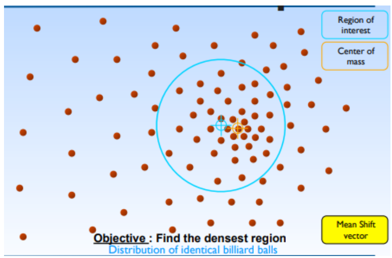
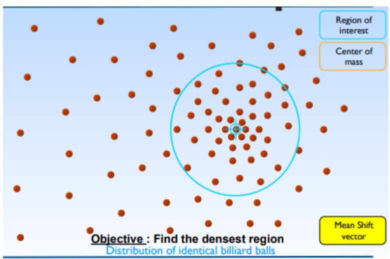
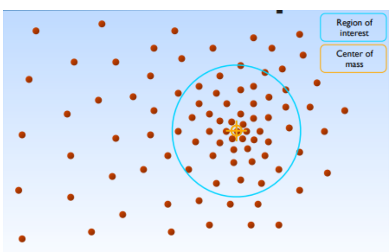
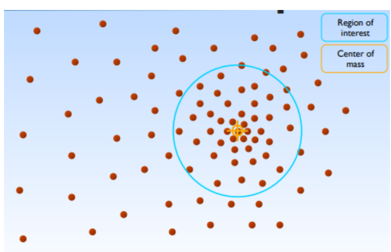
1.2.1 Mean Shift algorithm idea

Mean shift is a non-parametric feature-space analysis technique for locating the maxima of a density function, a so-called mode-seeking algorithm. In image processing, it's used for tracking.

Given some points $\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_n$ with d dimensions ("features"), for each data point, mean shift defines a window around it and computes the mean of data point. Then it shifts the centre of window to the mean and repeats the algorithm till the window stops moving. In image processing, feature space is often the colour space. Table 1 illustrates the iterations until the algorithm converges.

Mean shift is a nonparametric iterative algorithm. It considers each point sampled from a probability distribution, i.e. each point is most likely found at its actual measured position, but it can also be in a neighbourhood around it.

Table 1: Mean shift update steps shown on a very high level – in this case the new centre is simply the centroid, until converge (last row).

Current centre (blue)	Mean shift vector	New centre (yellow)
		
		
		
		

❓ How do we find the peak towards which the circle should move?

✅ The circle should move towards the densest point of the distribution of all points within the ROI. The peak is found by superimposing all the individual probability distributions around each point and finding the N highest maxima of the result (N is the number of classes we want to have).

❓ How do we convert a set of discrete input points to a continuous density function so that we can find the maxima (Fig. 4)?

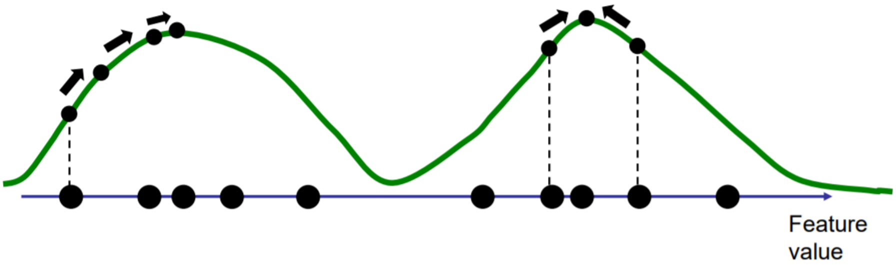


Fig. 4. The green curve is what we roughly want to generate from the 1D input points. The arrows simply show the path gradient ascent would follow to locate the maxima.

✅ Let us define a kernel function.

> **DEFINITION 1.2.** *A* k e r n e l *is a real-valued function of the points* $\boldsymbol{x}_1, \boldsymbol{x}_2, \ldots, \boldsymbol{x}_n$ *that satisfies the following properties:*
> 1. *K is maximum at $\mathbf{0}$, non increasing, and decays away from the maximum.*
> 2. *K is radially symmetric.*
> 3. *$K(\boldsymbol{x}) \geq 0$.*
> 4. *$\int_{\mathbb{R}^d} K(\boldsymbol{x})\, d\boldsymbol{x} = 1$*

Then for each input point, its kernel function should look roughly as follows.

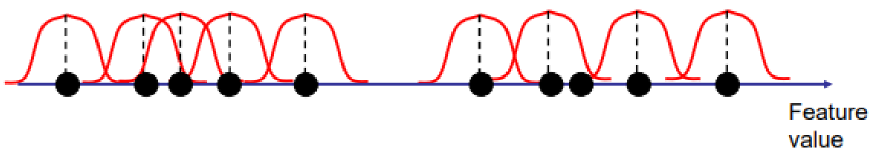


Fig. 5. The kernel functions $K(\mathbf{x} - \mathbf{x}_i)$, where $\mathbf{x}_i$ are the input points.

If we allocate each point $\mathbf{x}_i$ its own kernel $K(\mathbf{x} - \mathbf{x}_i)$, then by summing all N of them and diving by a constant C to normalise the result we can get a probability density function (PDF):

$$f(\mathbf{x}) = \frac{1}{C} \sum_{i=1}^{N} K(\mathbf{x} - \mathbf{x}_i) \tag{1.5}$$

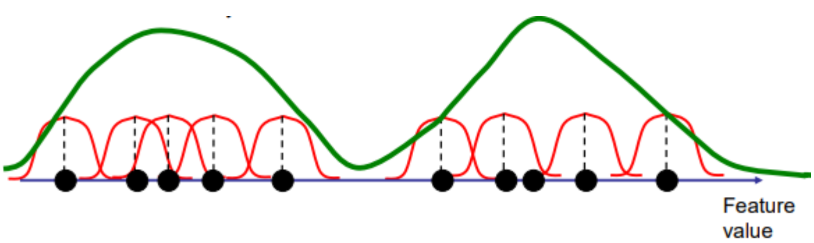


Fig. 6. The sum of individual kernel functions functions $K(\mathbf{x} - \mathbf{x}_i)$.

$f(\mathbf{x})$ approximates the probability that feature $\mathbf{x}$ is observed given the data points. The maxima of f (the "modes" of the pdf) correspond to the clusters in the data. As shown in Fig. 4, a way to reach the peak of the PDF f is by incrementing the mean by $\nabla f(\mathbf{x})$. A very rough algorithm for Mean Shift would therefore be.

Algorithm 2 Mean shift on a very high level

procedure MEANSHIFTIDEA($\mathbf{x}_1, \mathbf{x}_2, \ldots, \mathbf{x}_N$)
  for $i = 1, \ldots, N$ **do**
    $\mathbf{x} \leftarrow \mathbf{x}_i$
    while no convergence **do** ▷ gradient ascent for each point individually
      $\mathbf{x} \leftarrow \mathbf{x}_i + \nabla f(\mathbf{x}) = \mathbf{x}_i + \frac{1}{C}\sum_i \nabla K(\mathbf{x} - \mathbf{x}_i)$
  return $\mathbf{x}$

1.2.2 Mean Shift terminology and notation

Before the maths is presented, this is notation that will be used.

- d – the dimension of the input column vector, the entries of this vector are also called "features".
- $\mathbf{x}_i$ – the data points.
- "Kernel" $K(\mathbf{x})$ – the function that assigns weight to every point of interest. For example, it can be Gaussian, flat, etc.
- "Bandwidth" h – the radius of the region of interest (ROI).
- "PDF" f($\mathbf{x}$) – probability density function.

1.2.3 Mathematical analysis

A basic requirement for the kernel function, as stated in Def. 1.2 is radial symmetry, i.e.

$$K(\mathbf{x}) = c_d k(\|x\|^2) \tag{1.6}$$

, where c_d acts as the normalisation constant and is the volume of the d-dimensional sphere, such that $K(\mathbf{x})$ integrates to 1. Some functions that can serve as the basis k for the kernel are

$$\text{Epanechnikov} \quad k_E(\mathbf{x}) = \begin{cases} 1 - \|x\|^2 & \|x\| \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.7}$$

$$\text{Uniform} \quad k_U(\mathbf{x}) = \begin{cases} 1 & \|x\| \le 1 \\ 0 & \text{otherwise} \end{cases} \tag{1.8}$$

$$\text{Gaussian} \quad k_N(\mathbf{x}) = \exp(-\frac{1}{2}\|x\|^2) \tag{1.9}$$

It can be proven that the pdf function that approximates kernel density given inputs $\mathbf{x}_i, \ldots, \mathbf{x}_n$, where $\mathbf{x}_i \in \mathbb{R}^d$ is

$$\hat{f}(\mathbf{x}) = \frac{1}{nh^d}\sum_{i=1}^{n} K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{1.10}$$

Assuming that the kernel $K(\mathbf{x})$ is differentiable, the gradient of the kernel density is

$$\hat{\nabla} f(\mathbf{x}) := \nabla \hat{f}(\mathbf{x}) = \frac{1}{h^d}\sum_{i=1}^{n} \nabla K\left(\frac{\mathbf{x} - \mathbf{x}_i}{h}\right) \tag{1.11}$$

Computing the gradient, the derivative is

$$\nabla \hat{f}(\mathbf{x}) = \frac{2c_d}{nh^{d+2}}\sum_{i=1}^{n} (\mathbf{x} - \mathbf{x}_i) k'\left(\left\|\frac{\mathbf{x} - \mathbf{x}_i}{h}\right\|^2\right) \tag{1.12}$$

$$= \frac{2c_d}{nh^{d+2}}\left(\sum_{i=1}^{n} g_i\right)\underbrace{\left(\frac{\sum_{i=1}^{n}\mathbf{x}_i g_i}{\sum_{i=1}^{n} g_i} - \mathbf{x}\right)}_{\text{mean shift vector}}, \tag{1.13}$$

$$g(r) = k'(r), \quad r := \|\mathbf{x}\|, \quad g_i = g(\|(\mathbf{x} - \mathbf{x}_i)/h\|^2) \tag{1.14}$$

Definition 1.3 (mean shift vector)**.** *Referring to Eq.* (1.14)*,* $\sum_{i=1}^{n} g_i$ *is yet another kernel estimation and the second term* $M(\boldsymbol{x}) = \sum_{i=1}^{n} \boldsymbol{x}_i g_i / \sum_{i=1}^{n} g_i - \boldsymbol{x}$ *is the mean shift vector.*

It always points toward the direction of the maximum increase in the density therefore we want to shift the current estimation $\mathbf{x}$ by it in each iteration. The mean shift vector must be $\mathbf{0}$ at optimum, i.e. the algorirthm stops at $\mathbf{x} = \frac{\sum_{i=1}^{n} \mathbf{x}_i g_i}{\sum_{i=1}^{n} g_i}$. This is equivalent to draft Alg. 2 converging. Given this knowledge about the gradient, the latter algorithm is rewritten as follows.

Algorithm 3 Mean shift algorithm

```
1: procedure MeanShift(x_1, x_2, ..., x_n)
2:     for i = 1,...,n do
3:         x ← x_i
4:         while no convergence do                ▷ gradient ascent for each point individually
5:             x ← x + M(x) = [Σ_{i=1}^n x_i g(‖(x−bx_i)/h‖²)] / [Σ_{i=1}^n g(‖(x−x_i)/h‖²)],   g(‖x‖) = k′(‖x‖)
6:     return x
```

guaranteed to converge. The update step is not as complicated as it seems. For example for the kernel

$$k(\|\mathbf{x}\|) = \begin{cases} 1 - \|x\| & \|x\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

, its derivative w.r.t. the distance $\|\mathbf{x}\|$ is

$$g(\|\mathbf{x}\|) = \begin{cases} -1 & \|x\| \leq 1 \\ 0 & \text{otherwise} \end{cases}$$

Comparing the argument of g with 1 gives us the data points of interest therefore the "mean" part of $M(\mathbf{x})$ is:

$$\frac{\sum_{i=1}^{n} \mathbf{x}_i g\left(\left\|\frac{\mathbf{x}-bx_i}{h}\right\|^2\right)}{\sum_{i=1}^{n} g\left(\left\|\frac{\mathbf{x}-\mathbf{x}_i}{h}\right\|^2\right)} = \frac{-\sum_{\|\mathbf{x}-\mathbf{x}_i\|<h} \mathbf{x}_i}{-\sum_{\|\mathbf{x}-\mathbf{x}_i\|<h} 1} = \frac{\sum_{\|\mathbf{x}-\mathbf{x}_i\|<h} \mathbf{x}_i}{n_h} \tag{1.15}$$

n_h is simply the number of inside the kernel, for which $\|\mathbf{x} - \mathbf{x}_i\| < h$ and $\sum_{\|\mathbf{x}-\mathbf{x}_i\|<h} \mathbf{x}_i$ is just the average of the data points within a radius h of $\mathbf{x}$! Regarding the convergence of the algorithm the following can be proved.

Theorem 1.1. *If the kernel function* $k(\boldsymbol{x})$ *is convex and monotonically decreasing then the update of* $\boldsymbol{x}$ *converges and the pdf* $\hat{f}(\boldsymbol{x})$ *increases.*

For the Epanechnikov kernel, convergence is reached in finite number of steps. Finally, mean shift runs in $\mathcal{O}(n^2 T)$, where n is the number of input points and T the number of iterations.

1.2.4 Mean Shift as a tool for segmentation

When we want to segment an image, it is usually converted from RGB to another colour space such as HSV or LUV. In this case, suppose the image is in LUV.

Then the feature space is (l, u, v, x, y). To perform segmentation, we apply *two different* mean shifts in the 5-dimensional space as we want to segment pixels based on their location and colour. For each pixel (x_i, y_i) of intensity color (l_i, u_i, v_i), find the corresponding mode c_{col}. All of the pixels (x_i, y_i) corresponding to the same mode c_{col} are grouped into a single region. At the same time, mean shift is performed in the 2D space xy and all of the corresponding pixels (x_i, y_i) are grouped into a single mode c_{pos}. The kernel in this case is the product of the position kernel and the colour kernel,

$$K_{pos,col}(\mathbf{x}) = \frac{c}{h_{col}^3 h_{pos}^2} k\left(\frac{\|\mathbf{x}_{pos}\|^2}{h_{pos}^2}\right) k\left(\frac{\|\mathbf{x}_{col}\|^2}{h_{col}^2}\right) \tag{1.16}$$

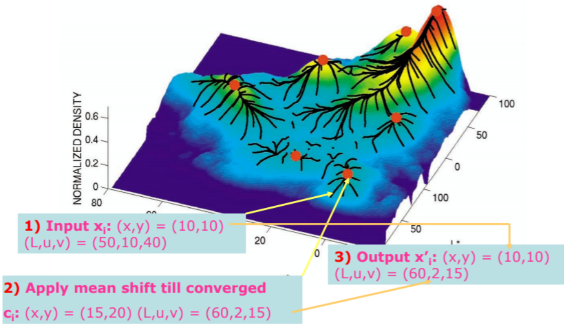


Fig. 7. Mean shift in the LUV space.

1.2.5 Mean Shift as a tracker

The idea behind using it as a tracker is the following. Given a frame that contains the object of interest (OOI), we want to successfively move the window of radius h towards the "centre" of the object. How is the centre defined?

After selecting a sample of the OOI, backprojection from the previous section helps create a greyscale. salience (likelikehood) frame (see b matrix in backprojection section). In this frame, the whiter the pixel, the more likely it is to belong to the OOI.

Then, mean shift can be performed in 3D - (x, y, i), where i is the salience intensity to chase the object and output its (x, y).

1.2.6 Implementation from scratch

Below are the main features of my own adaptation of mean shift.

- Mean shift is performed in the xy space so the "mode" (converge) centre is 2D.
- If it performed in xy space, which points are considered? The candidate points are the result of backprojecting a sample of the OOI to the frame, create a likelihood frame b, as described in Section 1.1. Then they are automatically thesholded, which yields a BW frame. For that frame, the white points are likely to belong to the OOI and the black are irrelevant. So we want to generate a pdf from those white points and find its centre of density.
- The chosen mean shift kernel if $k(r) = 1 - r$, $r \leq 1$, $r = \|x\|$. The radius of interest can be chosen by the user but defaulted to some small value anyway. Therefore the update step of the m.s. vector x is simple – it is simply the centroid of all points up to distance h around x as derived in 1.15.

My implementation code is listed in A.2. The program is standalone and expect a video path and a radius from the user. Below are some representative outputs for a football sequence. For that particular scane, the player was successfully tracked for the majority of the frames.

Fig. 8. Initial step; grabbing a sample of the player to track (green box).

Fig. 9. An intermediate tracking step (blue circle).

Fig. 10. Player keeps being tracked right after being blocked by another player.

The algorithm is robust given that the sample represents reasonably well the OOI. It is not robust when the scale of the OOI changes, e.g. in car scenes when a car to be tracked car is initially in the background and in the end approaches the camera. Then mean shift variations that are able to adapt the radius h need to be considered and there's a good amount of literature on that.

1.2.7 Implementation in OpenCV

Mean shift in OpenCV consists of the following main stages:

1. Set up a target; i.e. grab a sample of the OOI. Once again, the OOI shouldn't be in RGB, but in a colour space that decouples hue and illumiation.
2. Provide an initial window – essentially initialise the mean shift tracking vector x and provide a radius h to define the area around $\mathbf{x}$ where points for the update of $\mathbf{x}$ will be considered.
3. Perform backprojection of the target on the current frame, generating a greyscale image. Perform mean shift on the greyscale backprojected image.

After grabbing the ROI (in HSV), the OpenCV approach also filters only certain H (0 to 180 – maximum is 180), S (60 to 255 – maximum is 255), V (32 to 255 – maximum is 255) bands in order to remove potential noise:

```
mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)), np.array((180.,255.,255.)))
```

Mean shift is performed, for instance, as:

```
cv2.meanShift(probImage, track_window, criteria) -> retval, track_window
```

- `probImage`: greyscale backprojected frame.
- `track_window`: the updated tracking window as `x, y, w, h` (essentially the updated m.s. vector).
- `term_crit`: termination criteria, e.g. to set them to 10 iterations or window displacement no more than 1 pixel do `( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1)`.
- `retval`: A boolean whether the algorithm was successful.
- `track_window`: The updated tracking window in the same format as the old one.

The code provided by OpenCV's documentation has been lazily re-written and made interactive as listed in A.5. The user can define the bounding box around the object of interest and then press q to finalise the selection and start the algorithm.

1.2.8 Mean Shift pros and cons

- ✓ Does not assume spherical clusters.
- ✓ Just a single parameter (window size).
- ✓ Finds variable number of modes.
- ✓ Robust to outliers.

- ✗ Output depends on window size.
- ✗ Computationally expensive (however, not impossible to use in real time).
- ✗ Does not scale well with dimension of feature space.

1.3 Camshift (Continuously Adaptive Mean Shift)

2 Motion-based methods

2.1 Optical Flow

2.2 Lukas-Kanade tracking

A Appendices

A.1 HSV domain

A.2 Histogram backprojection implementation from scratch – source code

Listing 1: Histogram backprjection from scratch (src/hist_backproj/backproj.py).

```
import cv2, numpy as np
import sys
import pdb

g_clicks_xy = []

def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
        Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))

"""
@im: The input image as read (in BGR)
"""
def get_model(im):
    """
        Process user input (clicks) and Extract the target image
        (model) in hsv.
    """
    global g_clicks_xy
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    cv2.namedWindow('input')
    cv2.setMouseCallback('input', on_click)
    while True:
        cv2.imshow('input', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    # for clicks, index 0 = x, index 1 = y
    g_clicks_xy = sorted(g_clicks_xy,
            key = lambda x: x[0]**2 + x[1]**2,
            reverse = True) [-2:]
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    hsvt = hsv[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    rect = cv2.rectangle(im.copy(), g_clicks_xy[0], g_clicks_xy[1],
            color = (0,255,0))
    qimshow(rect, wname = 'selection', delay = 3)
    return hsvt


"""
@hsv: The whole input (search) image in hsv
@hsvt: The target (model), i.e. the ROI, in hsv
@<return>: The ration histogram of hsvt by hsv, clipped from 0 to 1
"""
def ratio_histogam(hsv, hsvt):
    # see doc: HS histograms, [0, 180] as in OpenCV 0 <= H <= 179
    M = cv2.calcHist([hsv], [0, 1], None, [180, 256], [0, 180, 0, 256])
```

```python
    I = cv2.calcHist([hsvt], [0, 1], None, [180, 256], [0, 180, 0, 256])
    R = np.divide(np.array(M, np.float),
            np.array(I, np.float),
            out = np.zeros_like(I),
            where = I != 0)
    R[R > 1.0] = 1.0
    return R


"""
@hsv: the whole input image in HSV
@R: the ratio histogram as returned by the ratio_histogram function
@r: the radius of the disk backprojection convolves with
"""
def backproject(hsv, R, rad = 15):
    """
        Generate a 2D binary image where ones are probably the object(s)
        of interest and zeros the background
    """
    b = np.zeros((hsv.shape[0], hsv.shape[1]), np.uint8)
    for r in range(hsv.shape[0]):
        for c in range(hsv.shape[1]):
            b[r,c] = R[hsv[r,c][0], hsv[r,c][1]]
    b = np.uint8(b)
    disk = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (rad, rad))
    # convolve with disk
    b = cv2.filter2D(b, -1, disk, b)
    b = np.uint8(b)
    b = np.array(cv2.normalize(b, b, 0, 255, cv2.NORM_MINMAX), np.uint8)
    _, thresh = cv2.threshold(np.uint8(b), 0, 255,
            cv2.THRESH_BINARY | cv2.THRESH_OTSU )
    return thresh


def usage():
    str_usage = 'Run with $ python <program_name> <input_image>\n'\
    'When your input image shows up, click 2 times to define\n'\
    'a bounding box around  a sample of your object of interest.\n'\
    'Then press "q" to finish the selection.'
    print str_usage
    sys.exit(0)


def main():
    assert len(sys.argv) > 1,\
        "Run the program with -h or --help to print usage"
    if sys.argv[1] == '-h' or sys.argv[1] == '--help':
        usage()
    else:
        im = cv2.imread(sys.argv[1])
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    hsvt = get_model(im)
    R = ratio_histogam(hsv, hsvt)
    thresh = backproject(hsv, R)
    # final processing - AND the 2D binary threshold image with the
    # original RGB input image to show the result
    thresh = cv2.merge((thresh, thresh, thresh))
    res = cv2.bitwise_and(im, thresh)
    qimshow(res)


if __name__ == '__main__':
    main()
```

A.3 Histogram backprojection implementation using OpenCV - source code

Listing 2: Histogram backprjection using OpenCV's `calcBackProject` (src/hist_backproj/backproj_cv.py).

```
import cv2
import numpy as np
import sys

g_clicks_xy = []

def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
        Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))


"""
@im: The input image as read (in BGR)
"""
def get_model(im):
    """
        Process user input (clicks) and Extract the target image
        (model) in hsv.
    """
    global g_clicks_xy
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    cv2.namedWindow('input')
    cv2.setMouseCallback('input', on_click)
    while True:
        cv2.imshow('input', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    # for clicks, index 0 = x, index 1 = y
    g_clicks_xy = sorted(g_clicks_xy,
            key = lambda x: x[0]**2 + x[1]**2,
            reverse = True) [-2:]
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    hsvt = hsv[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    rect = cv2.rectangle(im.copy(), g_clicks_xy[0], g_clicks_xy[1],
            color = (0,255,0))
    qimshow(rect, wname = 'selection', delay = 3)
    return hsvt


def backproject(hsv, hsvt, rad = 9):
    # Calculating object histogram
    # In OpenCV, hue lies within [0,179]!
    hsvt_hist = cv2.calcHist([hsvt],# image
            [0, 1],                 # channel selection
            None,                   # mask
            [90, 128],              # no of bins
```

```
            [0, 180, 0, 256]        # channel ranges
            )
    # normalize histogram to [0,255] and apply backprojection
    # to get R  (ratio histogram) matrix -> R between 0 and 1
    cv2.normalize(hsvt_hist, hsvt_hist, 0, 255, cv2.NORM_MINMAX)
    R = cv2.calcBackProject([hsv],  # image
            [0,1],                  # channel selection
            hsvt_hist,              # histogram array
            [0,180,0,256],          # channel ranges
            scale = 1)
    # convolve with circular disc
    disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (rad, rad))
    cv2.filter2D(R, -1, disc,R)
    # Otsu's threshold
    _, R_thresh = cv2.threshold(R, 0, 255,  cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    # Make it 3D to AND it with the search image
    R_thresh = cv2.merge((R_thresh, R_thresh, R_thresh))
    return R_thresh


def usage():
    str_usage = 'Run with $ python <program_name> <input_image>\n'\
    'When your input image shows up, click 2 times to define\n'\
    'a bounding box around  a sample of your object of interest.\n'\
    'Then press "q" to finish the selection.'
    print str_usage
    sys.exit(0)


def main():
    assert len(sys.argv) > 1,\
        "Run the program with -h or --help to print usage"
    if sys.argv[1] == '-h' or sys.argv[1] == '--help':
        usage()
    else:
        im = cv2.imread(sys.argv[1])
    cv2.namedWindow('select area then press q')
    cv2.setMouseCallback('select area then press q', on_click)
    while True:
        cv2.imshow('select area then press q', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    hsv = cv2.cvtColor(im,cv2.COLOR_BGR2HSV)
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    rbgt = im[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    hsvt = cv2.cvtColor(rbgt, cv2.COLOR_BGR2HSV)
    R_thresh  = backproject(hsv, hsvt)
    res = cv2.bitwise_and(im, R_thresh)
    qimshow(res)


if __name__ == '__main__':
    main()
```

A.4 My mean shift implementation from scratch – source code

Listing 3: Mean shift using OpenCV's `meanShift` (src/mean_shift/my_mean_shift.py).

```
from __future__ import print_function
import cv2
import numpy as np
import sys


g_clicks_xy = []

######### UI #########
def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
    Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))

######### Maths #########
def is_in_circle(centre, r, point):
    return (centre[0] - point[0])**2 + (centre[1] - point[1])**2 <= r**2

def centroid(pts2D):
    x, y =zip(*pts2D)
    N = len(x)
    return int(sum(x)/N), int(sum(y)/N)

def dist(x, y):
    x, y = np.asarray(x), np.asarray(y)
    return np.linalg.norm(x - y)

######### Mean Shift process #########
def backproject(hsv, hsvt, rad = 9):
    # Calculating object histogram
    # In OpenCV, hue lies within [0,179]!
    hsvt_hist = cv2.calcHist([hsvt],# image
        [0, 1], # channel selection
        None, # mask
        [90, 128], # no of bins
        [0, 180, 0, 256] # channel ranges
        )
    # normalize histogram to [0,255] and apply backprojection
    # to get R (ratio histogram) matrix -> R between 0 and 1
    cv2.normalize(hsvt_hist, hsvt_hist, 0, 255, cv2.NORM_MINMAX)
    R = cv2.calcBackProject([hsv], # image
        [0,1], # channel selection
        hsvt_hist, # histogram array
        [0,180,0,256], # channel ranges
        scale = 1)
    # convolve with circular disc
    if rad:
        disc = cv2.getStructuringElement(cv2.MORPH_ELLIPSE, (rad, rad))
        cv2.filter2D(R, -1, disc,R)
        # Otsu's threshold
    _, R_thresh = cv2.threshold(R, 0, 255, cv2.THRESH_BINARY | cv2.THRESH_OTSU)
    # Make it 3D to AND it with the search image
```

```
    #R_thresh = cv2.merge((R_thresh, R_thresh, R_thresh))
    return R_thresh

"""
@bw: thresholded backprojected frame (M x N)
@x: current mean shift vector estimation (tuple)
@h: mean shift ROI radius (int)
@<return>: new mean shift vector estimation (tuple)
"""
def my_mean_shift(bw, x, h, conv_thresh = 2.0):
    """
    This mean shift works as follows:
    @bw is black and white image obtained as the  result of thesholding
    the backprojected frame with the model, model being a small sample
    of the object of interest extracted by the user.
    The domain is simply the xy, and for the update of m.s. vector x only
    white pixels up to h pixels around the current x are considered. White
    pixels are most likely to be part of the OOI. The updated x is returned
    """
    x_old = (np.inf, np.inf)
    while dist(x_old, x) > conv_thresh:
        started = True
        x_old = x
        points_in = []
        for r in range(bw.shape[0]):
            for c in range(bw.shape[1]):
                if bw[r, c] and is_in_circle(x, h, (c, r)):
                    points_in.append((c, r))
                    x = centroid(points_in)
    print (x)
    #qimshow(cv2.circle(bw.copy(), x, 6, 126, 4),  delay = 0)
    #qimshow(bw)
    return x


"""
@im: The input image as read (in BGR)
"""
def get_model(im):
    """
    Process user input (clicks) and Extract the target image
    (model) in hsv.
    """
    global g_clicks_xy
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    cv2.namedWindow('2 clicks, then press q')
    cv2.setMouseCallback('2 clicks, then press q', on_click)
    while True:
        cv2.imshow('2 clicks, then press q', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    # for clicks, index 0 = x, index 1 = y
    g_clicks_xy = sorted(g_clicks_xy,
            key = lambda x: x[0]**2 + x[1]**2,
            reverse = True) [-2:]
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    hsvt = hsv[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    rect = cv2.rectangle(im.copy(), g_clicks_xy[0], g_clicks_xy[1],
            color = (0, 255, 0))
```

```python
    qimshow(rect, wname = 'selection', delay = 3)
    return hsvt


def usage():
    str_usage = \
    "Instructions:\n\
    Run the program with\n\
    $ python <prog_name> <video> <roi_radius>\n\
    When the \"input\" window appears, click 2 times to create a \n\
    bounding box around a sample of the object to be tracked.\n\
    Then press \"q\" to continue.\n"
    print(usage)


######### Driver #########
def main():
    """
    Instructions:
    Run the program with
    $ python <prog_name> <video> <roi_radius>"
    When the "input" window appears, click 2 times to create a
    bounding box around a sample of the object to be tracked.
    Then press "q" to continue.
    """
    assert len(sys.argv) > 1,\
            "\nUsage: $ python <prog_name> <video> <roi_radius>.\n\
            Run with $ python -h to print detailed instructions"
    if sys.argv[1] in ['-h', '--help']:
        usage()

    vid_path = sys.argv[1]
    h = 20 if len(sys.argv) == 2 else int(sys.argv[2])
    cap = cv2.VideoCapture('soccer2.mp4')
    _, frame1 = cap.read()
    # Get model (target) image
    hsv = cv2.cvtColor(frame1, cv2.COLOR_BGR2HSV)
    hsvt = get_model(frame1)

    bw = backproject(hsv, hsvt, rad = 0)

    # Initialise mean shift (x) vector from user input
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    x = (click_tl[0] + w/2, click_tl[1] + h/2)
    valid = True
    # frame by frame processing
    while(valid):
        valid, im = cap.read()
        hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
        bw = backproject(hsv, hsvt, rad = 0)
        x = my_mean_shift(bw, x, h)
        cv2.imshow('detection', cv2.circle(im.copy(), x, 1, (0, 255, 0),
            6))
        if cv2.waitKey(1) & 0xFF == ord('q'):
            break
    # Release the capture
    cap.release()
    cv2.destroyAllWindows()


main()
```

A.5 Mean Shift implementation using OpenCV - source code

Listing 4: Mean shift using OpenCV's `meanShift` (src/mean_shift/mean_shift_cv.py).

```python
import numpy as np
import cv2
import sys

g_clicks_xy = []

######### UI #########
def qimshow(im, delay = 10, wname = 'display'):
    cv2.imshow(wname,im)
    cv2.waitKey(delay * 1000)
    cv2.destroyAllWindows()


def on_click(event, x, y, flags, param):
    """
    Mouse callback function - write to global list of clicks
    """
    global g_clicks_xy
    if event == cv2.EVENT_LBUTTONDOWN:
        g_clicks_xy.append((x, y))

"""
@im: The input image as read (in BGR)
"""
def get_model(im):
    """
    Process user input (clicks) and Extract the target image
    (model) in hsv.
    """
    global g_clicks_xy
    hsv = cv2.cvtColor(im, cv2.COLOR_BGR2HSV)
    cv2.namedWindow('2 clicks, then press q')
    cv2.setMouseCallback('2 clicks, then press q', on_click)
    while True:
        cv2.imshow('2 clicks, then press q', im)
        k = cv2.waitKey()
        if k == ord('q'):
            break
    cv2.destroyAllWindows()
    # for clicks, index 0 = x, index 1 = y
    g_clicks_xy = sorted(g_clicks_xy,
            key = lambda x: x[0]**2 + x[1]**2,
            reverse = True) [-2:]
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    hsvt = hsv[click_tl[1]: click_tl[1] + h,
            click_tl[0]: click_tl[0] + w]
    rect = cv2.rectangle(im.copy(), g_clicks_xy[0], g_clicks_xy[1],
            color = (0, 255, 0))
    qimshow(rect, wname = 'selection', delay = 3)
    return hsvt

def main():
    cap = cv2.VideoCapture(sys.argv[1])
    # take first frame of the video
    ret,frame = cap.read()
    # setup initial location of window
    r,h,c,w = 250,90,400,125  # simply hardcoded the values
    track_window = (c,r,w,h)
    # set up the ROI for tracking
```

```python
    roi = frame[r:r+h, c:c+w]
    hsv_roi = get_model(frame)
    click_br, click_tl = g_clicks_xy[0], g_clicks_xy[1]
    w = click_br[0] - click_tl[0]
    h = click_br[1] - click_tl[1]
    x = click_tl[0]
    y = click_tl[1]
    track_window = (x, y, w, h)
    #hsv_roi =  cv2.cvtColor(roi, cv2.COLOR_BGR2HSV)
    # Filter out noise
    mask = cv2.inRange(hsv_roi, np.array((0., 60.,32.)),
            np.array((180.,255.,255.)))
    roi_hist = cv2.calcHist([hsv_roi],[0],mask,[180],[0,180])
    cv2.normalize(roi_hist,roi_hist,0,255,cv2.NORM_MINMAX)
    # Setup the termination criteria,
    #either 10 iteration or move by atleast 1 pt
    term_crit = ( cv2.TERM_CRITERIA_EPS | cv2.TERM_CRITERIA_COUNT, 10, 1 )
    while(1):
        ret ,frame = cap.read()
        if ret == True:
            hsv = cv2.cvtColor(frame, cv2.COLOR_BGR2HSV)
            dst = cv2.calcBackProject([hsv],[0],roi_hist,[0,180],1)
            # apply meanshift to get the new location
            ret, track_window = cv2.meanShift(dst, track_window, term_crit)
            # Draw it on image
            x,y,w,h = track_window
            img2 = cv2.rectangle(frame, (x,y), (x+w,y+h), 255,2)
            cv2.imshow('img2',img2)
            k = cv2.waitKey(30) & 0xff
            if k == ord('q'):
                break
        else:
            break
    cv2.destroyAllWindows()
    cap.release()

main()
```