Implementation and Evaluation of the Mean Shift Tracker

Roberto Valenti
Id: 0493198

Felix Hageloh
Id: 0425257

Introduction

Real-time object tracking has been an important field of research over recent years as it has many applications in domains such as: robot vision, automated surveillance etc.
The basic approach to tracking and can be described as follows:

We have some representation of our target object (based on color, pca etc) and are trying to find the same object in subsequent video frames. However, tracking is not an easy task as it encounters many problems as follows:

- target object is often cluttered/occluded
- changing lighting conditions
- target changes in size/scale
- computation cost: dealing with video data is computational expensive, but we would like to only use minimal resources for tracking as more are required for higher level semantic computations (recognition, trajectory interpretation, and reasoning)

The last requirement rules out exhaustive search where for each frame we search the entire image for our target object. Since we know that in each frame the target object will be close to its position in the previous frame we can exploit this fact to enhance this exhaustive approach. The most basic form is called brute force tracking where instead of searching the entire image of the target, we only search in a specified region around the targets previous position. However, this approach is still very computational expensive so ideally we would like to make better use of the information learned from previous frames to limit the number of positions to search for the target.
Formally, we can describe the subsequent video frames as a sequence of states $\{x_k\}_{k=0,1,\dots}$. In addition we have a corresponding series of measurements $\{z_k\}_{k=1,\dots}$, so we would like to predict the new state x_k given all previous measurements $z_{1:k}$, or in probabilistic terms we would like to find the probability density function $p(x_k \mid z_{1:k})$ [1].
Using this approach several solutions have been presented, such as:

The Kalman filter which has been used by Boykov and Huttenlocher [2] to track vehicles in an adaptive framework.
The Extended Kalman Filter used by Rosales and Sclaroff [3] to estimate a 3D object trajectory from 2D image motion.
A Hidden Markov Model formulation for tracking was proposed by Chen *et al.* [4]

The mean-shift tracker has been very recent development that has proven to be very effective and efficient. The aim of this project is to implement a mean-shift tracker to follow players in a football video and to evaluate its strengths and weaknesses.

The paper first describes the model we chose for our target object (a football player) and then outlines the theoretical background of the mean-shift tracker. Next our implementation will be described and finally the results obtained discussed.

Target Model

It has been shown [5] that color probability distribution functions (pdf) can be used as a feature for object discrimination. In our case this seems to be a suitable feature, since football players normally have a distinctly colored tricot that is very different from the green background of the field and from that of the other team's players. Nevertheless, color is subjected to several the variations [5], such changes in intensity and saturation which are due to changes in the light source, the objects geometry and reflections. So we would like to choose a color system that is insensitive to those kinds of variations.
We can note that for a football match video lighting conditions are normally constant and can be approximated by a white light source [5]. Moreover, the players and the grass can be seen as matte surfaces. Hence in the standard RGB system the R value for an infinitesimal surface patch at location x is given by [5]

$$\beta_R(x) = G_b(\vec{x}, \vec{n}, \vec{s}) E(x) \int_\lambda B(\vec{x}, \lambda) F_R(\lambda) d\lambda \tag{1}$$

where G_b is the geometric term of the object body which is dependant on the surface normal $\vec{n}$ and the direction of the illumination source $\vec{s}$. Furthermore, $E(x)$ is the spectral power distribution of the incident light and $F_R(\lambda)$ is the cameras spectral sensitivity for R given a wavelength λ. We can see that this value is still affected by a lot of factors and thus RGB is not a suitable color system for this application. However, the normalized R value r is given by

$$r = \frac{R}{R+G+B} = \frac{G_b(x,n,s)E(x)k_r}{G_b(x,n,s)E(x)(k_r + k_g + k_b)} = \frac{k_r}{k_r + k_g + k_b} \tag{2}$$

where k_r, k_g and k_b are the terms $\int_\lambda B(\vec{x}, \lambda) F_k(\lambda) d\lambda$ for F_r, F_g and F_b respectively. Likewise g and b are defined as

$$g = \frac{k_g}{k_r + k_g + k_b} \quad \text{and} \quad b = \frac{k_b}{k_r + k_g + k_b} \tag{3}$$

Hence, we can see that normalized RGB (rgb) only depends on the surface albedo and the spectral sensitivity of the camera. Obviously both of them are constant in our case since we are tracking the same object and the camera used for each video frame is the same.

Hence we can use the color probability density function q in rgb space to represent our target object. We will refer to q also as *target model* from now on [1]. To find our target in

subsequent frames we will measure the pdf p about different locations y, where $p(y)$ is referred to as *target candidate*. To keep computational costs low we will represent those pdfs as m-bin histograms of only two rgb channels. However, in order not to loose too much information we will not have separate histograms for each channel, but a two dimensional histogram. Again, to reduce computation cost we assume at this point that a 16 x 16 bin histogram will be sufficient.

Hence our target model and target candidate will be a two dimensional rgb histogram taken over a specified region around a location y. The only problem with rgb is that it becomes unstable for low intensities, as we divide by very small numbers or potentially zero. This issue has to be addressed during the implementation by using appropriate masks.

Mean-Shift-tracking

The very basic idea of mean shift tracking is the same as brute force tracking, namely we can assume that in subsequent video frames the target will not change much in position and appearance. Since we measure our target candidate over a region, we can assume that part of the target object will still fall under the region of the previous position. Hence, when measuring the pdf about the target's previous position, we will detect at least some similarity with our target distribution. So essentially we want to include special information in the distance measure between our target and current distribution. This way we can calculate a gradient that points into the spatial direction where the two distributions are most similar and hence to where we are most likely to find our target object. To link our color pdf with spatial information we can use a kernel, which will define our target model as follows

Thus, in order to be able to estimate the most probable position of our target in the next frame we must:

a) Find an appropriate metric: The paper [REF] suggests using the Bhattacharyya coefficient for several reasons stated there. Given two discrete and normalized distributions $\hat{p}$ and $\hat{q}$ the Bhattacharyya Coefficient is defined as

$$\rho[\hat{p},\hat{q}] = \sum_{u=1}^{m} \sqrt{\hat{p}_u \hat{q}} \tag{4}$$

Seeing the two distributions as m-dimensional unit vectors, this is simply the cosine between [blalala] and [balabla]. However, we want to express the distribution of our candidate object as a spatial function, so we let $\hat{p}(y)$ be the distribution of the candidate object at position y. Thus, our coefficient becomes

$$\rho[\hat{p}(y),\hat{q}] = \sum_{u=1}^{m} \sqrt{\hat{p}(y)_u \hat{q}} \tag{5}$$

b) Define an appropriate kernel: Any isotropic kernel will do, but the Epanechnikov kernel is recommended [REF], as it defines an ellipsoidal region and gives more weights to pixel closer to the center of the kernel. This is useful because pixels far from the center of the object are more likely to be occluded or cluttered. The kernel function is defined by

$$k(x) = \frac{1}{2} c_d^{-1} (d+2)(1-x) \quad if \;\; x \le 1$$
$$k(x) = 0 \quad otherwhise \tag{6}$$

Note that d is the number of dimensions and c_d is the area of a unit circle in d dimensions, so $d = 2$ and $c_d = \pi$ in our case.

c) Calculate the gradient in respect to y based on the choices we made for a) and b): We want to minimize the distance between target and candidate distribution, which, for our choice of metric, is the same of maximizing the Bhattacharyya coefficient. Reference for the exact derivation can be found in [1], but it is important to know that we can estimate the gradient by using the mean shift procedure [1]. In essence, the mean shift procedure utilizes the fact that for any density function the mean of a set of neighboring samples is always biased towards a local mode (local maximum). This is quite intuitive since samples closer to the local mode will have bigger values thus 'attract' the mean towards them. So if we want to estimate the gradient at a point x we can calculate the mean vector $\bar{x}$ from a sample set about x. From that we can then calculate the mean-shift vector $\bar{x} - x$.
Using this technique we finally come to the result that given the target object's current position $\hat{y}_0$ we can estimate its next position by

$$\hat{y}_1 = \frac{\sum_{i=1}^{n_h} w_i x_i}{\sum_{i=1}^{n_h} w_i} \tag{7}$$

where n_h is the number of pixels under our kernel and

$$w_i = \sum_{u=1}^{m} \sqrt{\frac{\hat{q}_u}{\hat{p}_u(\hat{y}_0)}} \delta[x_i - u] \tag{8}$$

which is basically the square root of log-likelihood, but only for the values of the two distributions that correspond to the location (or pixel) x_i inside the kernel. Note that (7) holds in our case, because the derivative of our kernel (6) is a constant. Otherwise (7) should also include the derivative of the kernel as described in [REF].

Knowing all this we can now informally outline the mean-shift algorithm as follows (a more formal outline can be found in [1]):

Given:
the distribution of our target object $\hat{q}$ and its location in the previous frame $\hat{y}_0$.

1. Measure the distribution around the objects previous location $\hat{y}_0$ and check how well it matches the target distribution, by calculating the Bhattacharyya coefficient (5)

2. Estimate the objects next location $\hat{y}_1$ using (7).

3. Measure the color distribution around the new position and also check how well it matches the target distribution (Bhattacharyya again)

4. If the color distribution at $\hat{y}_0$ matches $\hat{q}$ better than at $\hat{y}_1$ then the step calculated in 2. was too big. So keep measuring the color distributions at locations closer to $\hat{y}_0$ by iteratively halving the step size until we find a better match (or reaches $\hat{y}_0$ again).

5. If the distribution at $\hat{y}_1$ matches $\hat{q}$ well enough (their distance is smaller than ε) we found our target object. Otherwise set $\hat{y}_0$ to $\hat{y}_1$ and go back to 2.

Implementation

The basic implementation of the tracker without enhancements is fairly straight forward and closely follows the outline given in the previous section. The code can be found in `meanshift.m`. Presented here are for each step an overview of the matlab implementation and deviations from the general outline.

I. To implement the target model we need to be able to construct a rgb histogram around a point y given a kernel k. To achieve this we created a function cropAt(y, width, height) that crops a rectangular region of size width x height centered at y from an image channel. Now we can simply pass the cropped region to a histogram function (implemented in `pHist2d`) that constructs a 16 x 16 bin histogram for two rgb channels.
However, this is not taking into account the kernel yet. Since the kernel doesn't change throughout the execution we only have to calculate once and store it in a matrix. The function `epanechKernel` constructs a kernel with a given with and height using the Epanechikov profile in (6). Note that this formula is defined for an Euclidian space around the center of the kernel, which we constructed by creating a matrix of the same size as the kernel and setting the center entry (ay height/2, width/2) to zero. The rest of the entries are then the Euclidian distances from the center.
Having our kernel which defines a weight for each pixel inside it, we now don't construct our histogram not by adding a one to corresponding bin for each pixel, but the pixels corresponding weight, which we can look up in our kernel matrix.
We then choose an arbitrary player from the first frame of our test video and measure his first location `Y0` manually. Also the kernel size is chosen manually by ensuring that the kernel covers most of the player without containing any background pixels. Then by using all the functions defined above we can measure the target distribution (histogram) which we store as `Ht`.

II. Just like getting our target histogram Ht we now measure our target candidate histogram `Hc` at `Y0` in the next frame. The Bhattacharyya coefficient is calculated in `bhatCoef.m` which is implemented directly from its mathematical definition. The only thing to note is that we first have to normalize `Ht` and `Hc` by dividing them by their internal sum (sum of all bin entries). We then store this coefficient as `C0`

III. This is the beginning of the mean-shift loop. We use (7) and (8) to calculate `Y1`. As mentioned before, to calculate the weights we have to retrieve for each pixel its corresponding bin values from the histograms which is implemented in `histValue.m`. In order not to divide by zero we ignore all zero values in `Hc`. Also it is important to note that $x_{i:n}$ are the relative coordinates centered at zero and not the true matrix coordinates. So we have to subtract `H` from the true coordinates each time.
Also since (7) assumes the image to be centered at `Y0`, our newly calculated `Y1` is not an absolute position, but an offset from `Y0`.

IV. The same as step 1 but now we measure `Hc` at `Y0+Y1` and call the new coefficient `C1`.

V. To make sure we didn't step too far we loop while `C1` is smaller than `C2` and half the step size each time as described 4. Note that if we reached `Y0` again for machine precision we should also exit the loop, otherwise we might loop infinitely.

VI. After we found our new location `Y0 + Y1` we can calculate the step size as the Euclidian distance between `Y0` and the new location. If it is smaller than eps (build-in value in Matlab) we are close enough to our target and thus exit the mean-shift loop. Otherwise we continue as described in 5.

Finally we include this mean shift algorithm inside a loop that reads the video frame by frame and normalizes it using (2) and (3). In order to smooth out the instability of rgb we first create a mask over the frame, considering only pixels with a RBG sum greater than 70. The rest of the pixels get set to zero. From this we can predict that there might a great number of "false" black pixels when measuring the color distributions. Thus we ignore the smallest bin in our target and candidate histograms by setting it to zero.

To show the result of the tracker we wrote a function `showFrame()` that displays one frame and indicates the position of the kernel by inverting the image over the kernel region. Optionally `showFrame()` can also be used to plot the path of the target player.

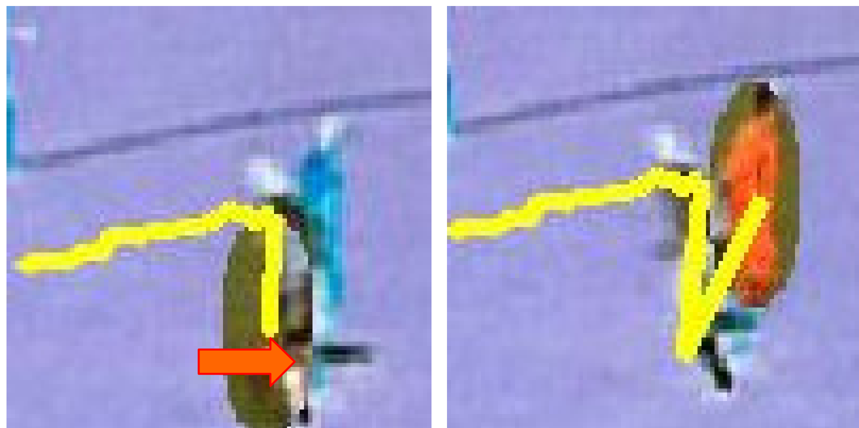
Experiments

Before testing the tracker we first have to decide on which two rgb channels to use. The best channels should be the ones that give the best discriminative power in respect to the background. We can investigate this through comparing our target distribution to the distribution of the background (taken under the same kernel) by calculating the Bhattacharyya coefficient. The results for our target player are given below:

Channels:	red-blue	green-blue	red-green
Bhat. Coefficient:	0.2617	0.3801	0.2823

We can see that for a combination of red and blue the target distribution is the most different from the background (the Bhattacharyya coefficient is the smallest).

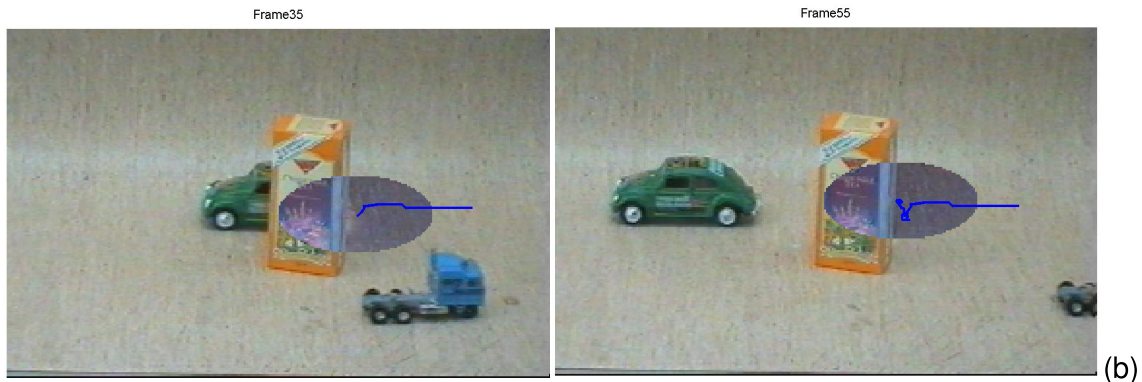
In general we can say the mean-shift tracker works very well and is reasonably fast even written in Matlab. An example video of the tracker can be found at http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/soccerbad.avi, where we can see that the tracker even works for the player being occluded by another for one frame. However, the very good performance is partly due to the fact that this is an easy example. Except for being occluded briefly the tacked player doesn't impose any other difficulties: He is well visible throughout the video, he stays at approximately the same depth most of the time, which means he doesn't change in scale and he moves at a reasonable speed. Also we have to note that the player covering him is from the opposite team and has thus a completely different color distribution. If the was player was covered by another from the same team we should expect the tracker to follow the other player afterwards.
The next two frames are (inverted) samples taken from the video that highlight the moment the player gets occluded.



(a)

We can see that actually the tracker is able to follow the player because a part of him stays visible all the time. Thus the tracker always follows the visible orange parts of the player, effectively moving around the occluding player. This works well in this example, but if the player was completely occluded we should expect the tracker to get lost. Moreover, if we would like to do some kind of trajectory analysis after tracking, this result is not useful as we can see from the plotted path, which should continue straight.

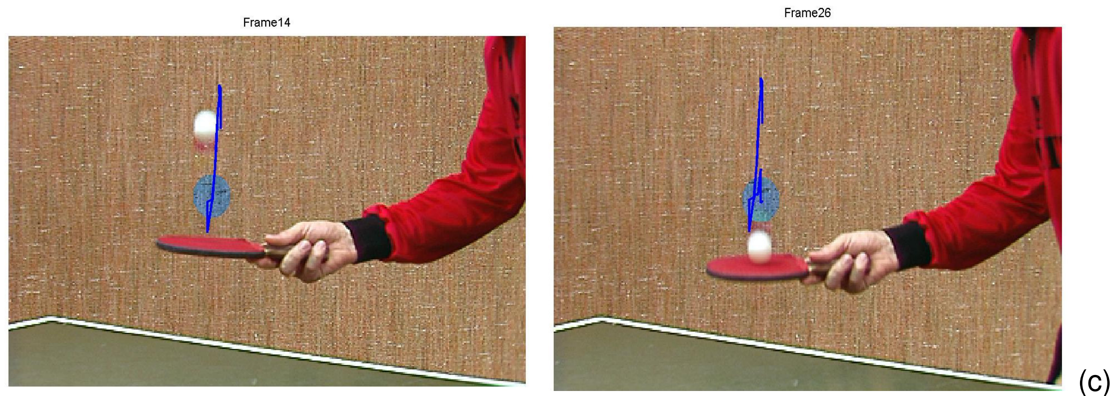
An example video where the target object gets occluded completely, or at least no part of the target falls under the kernel anymore, can be found here http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/truck.avi.
As expected the tracker gets lost after in can not find any colors matching the car after it gets occluded.



(b)

This example was also taken using the red and blue channel. Since the car is green we should naturally include the green channel. However, this was not suitable for this example since by chance the occluding object also has some green in it, which the tracker finds and thus is ultimately able to follow the car again. This is pure luck though and can not be counted as a valid example.

Another problem not addressed so far is a very fast moving object. An example of this can be found here http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/tennisbad.avi. Since the position of the to ball changes to much in subsequent frames, no part of the ball falls under the kernel at its previous position - the major underlying assumption of the mean-shift tracker. Some critical video frames are show here:



(c)

Another problem in this video is also that the ball is white while the background also has lots of white noise in it. In our 16 x 16 the color for the ball and the noise get mapped to the same bin, especially for rgb. We tried the same video with RGB but it didn't give any better results either.
Moreover, due to optical distortion the ball also appears to be stretched and discolored when it moves very fast. It thus also changes its scale and and color distribution, which oppose even more difficulties to the tracker.

This leads to the last problem of tracking, namely noisy and cluttered images especially where the noise has the same color as the target object. A good example is shown in here http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/car.avi.

Tracking a white car in this video was impossible. The tracker just got lost very quickly until it found another white car and kept jumping from car to car. Following the red car as of course possible, as it is not affected by noise. And example frame is shown here

(d)

Enhancements

Facing those problems covered previously. We came up with several enhancements to the tracker.

Trajectory Guessing:
The main problem is that the object is not under the kernel anymore when it reappears. Hence, after we lost the target object we would like to keep the kernel moving with the same velocity (includes direction) that the target had before it got occluded. By this we are most likely to have the kernel in the correct position when the object becomes visible again, unless of course it changes direction while occluded.
Our approach to solve this problem is pretty simple: If the Bhattacharyya coefficient is smaller than a certain threshold, we can assume that the object is lost. By analyzing the objects last location and its location five frames before, we can calculate a velocity vector for the target. Hence we let the kernel move with the same velocity until the target object is found again.
Of course, if the object is not found the kernel will just keep moving until he reaches the end of the image.

Trajectory Guessing is a fairly limited improvement that should only work in certain cases. However, for our cases is led to some satisfactory results posted here

http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/soccer.avi
http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/truck.avi

Adaptive Scaling:
As mentioned in [ref], the object can change its scale during the video sequence. This can lead to strange results during tracking, like focusing on details of the tracked object. To avoid this we should scale the kernel whenever the target object changes its scale. Details on the theory and implementation can be found in [1]. Our Implementation is taken directly from this description also a filter to avoid over sensitive scale adaptation.

By testing the adaptive scaling on the table tennis sequence, we notice a major performance improvement. As mentioned before, due to distortion the ball changes its scale, which is now accounted for with this enhancement, but the improvement is surprisingly large. The main problem identified for this video was the objects high velocity and not its changing scale. However, since the kernel gets stretched through the adaptive scaling it is now also big enough to follow the ball even in cases when its location in a new frame is so far away from its pervious position that our initial kernel would not have been able to detect it. The results for the table tennis video can be found here

http://student.science.uva.nl/~rvalenti/UVA/MIR/movies/tennis.avi

Several other improvements could be made to the tracker, many of them mentioned in [1], so that we would also be able to follow a white car in example (d). However, due to time constraints we were not able to implement other enhancements or run experiments with different parameters, such as different color models or kernel profiles.

Conclusion

The mean-shift tracker can be a very effective and efficient solution for video tracking. While its mathematical foundation can be complicated it is very easy to implement. It gives very good tracking results while being computationally inexpensive which allows for real time tracking and higher level computations. It is very flexible in a sense that it has many parameters that can be tuned. Many different target models can be used and enhancements be made, which makes the tracker adaptive to several different domains. However, its performance also strongly depends on the correct tuning of these parameters and the correct set of enhancements used. Thus some work and testing is required before using the tracker in different application areas.

References

[1] Dorin Comaniciu, Visvanathan Ramesh and Peter Meer; ”Kernel-Based Object Tracking”
[2] Y. Boykov and D. Huttenlocher, “Adaptive Bayesian recognition in tracking rigid objects,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* Hilton Head, SC, 2000, pp. 697–704.
[3] R. Rosales and S. Sclaroff, “3D trajectory recovery for tracking multiple objects and trajectory guided recognition of actions,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* Fort Collins, CO, 1999, pp. 117–123.
[4] Y. Chen, Y. Rui, and T. Huang, “JPDAF-based HMM for real-time contour tracking,” in *Proc. IEEE Conf. on Computer Vision and Pattern Recognition,* Kauai, Hawaii, volume I, 2001, pp. 543–550.
[5] Theo Geevers “Color in Image Search Engines”, Principals of visual information retrieval 2001