# Mean-Shift tracking

Radoslav Atanasoski

## I. Introduction

In this assignment, a mean-shift tracker is implemented, updating the bounding box position iteratively to the average density of the data points in the neighborhood. First, the mean-shift mode seeking function is implemented and tested with a variety of parameters and images. After which, the tracker is implemented which uses the mean-shift function for updating its position.

## II. Experiments

### A. Mean-shift mode seeking

The implemented mean-shift function takes several parameters, two of which mostly affect the final results:

- **Starting position** (x, y)
- The neighborhood **h (kernel size)**

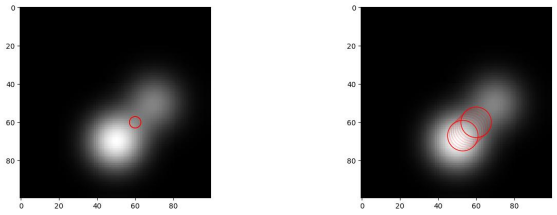
The function updates the position within the size of the frame, by using the numpy function clip. And the stopping criteria is when the rounded shift vector is equal to 0.

In table I, we can see a few examples of testing the function on an image generated from the provided function generate_responses_1.

| start (x, y) | h | **end (x, y)** | **steps** | **function value** |
|---|---|---|---|---|
| (33, 18) | 11 | (33, 18) | 0 | 0.0 |
| (33, 18) | 17 | (66, 51) | 13 | 0.000807386 |
| (15, 33) | 15 | (48, 67) | 14 | 0.0015149676 |
| (15, 33) | 21 | (49, 68) | 9 | 0.0015767945 |
| (34, 33) | 7 | (34, 33) | 0 | 0.0 |
| (34, 33) | 15 | (50, 68) | 14 | 0.0015885793 |
| (66, 62) | 11 | (62, 62) | 4 | 0.0008476697 |
| (66, 62) | 17 | (53, 67) | 13 | 0.001505647 |
| (82, 98) | 15 | (53, 72) | 13 | 0.0015149676 |
| (82, 98) | 25 | (52, 70) | 7 | 0.0015885793 |

Table I: Mean-shift function tested with different parameter values.

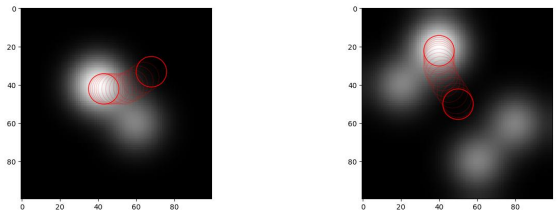
We can see that in some cases, if we are far away and have all zeros in the neighborhood, the algorithm does nothing and stops. However, if the kernel size is increased enough, the function will find the peak. If the kernel size is set too high, the function will not find the most accurate position, but it will come close and in fewer iterations. Setting the optimal kernel size depends on the environment, and is usually done empirically.



(a) Using a kernel size of 7. (b) Using a kernel size of 17.

Figure 1: Kernel size affect when in saddle point.

In figure 1, we can see an example where by increasing the kernel size, the function finds the peak and does not get stuck in a saddle point.



(a) Example for image with peak of 1.0 at (40, 40) and another peak of 0.5 at (60, 60).

(b) Example for image with peak of 1.0 at (20, 40), at other peaks of 0.5 at (40,20),(80,60),(60,80).

Figure 2: Examples on self generated images.

In figure 2, we can see two examples on images generated by a self made function. The first image, is the same as the provided one, but the coordinates of the peaks at different locations. However, for the second example, there are 4 peaks, one of which is with higher value than the rest, and the algorithm finds it. When all peaks were set with the same value, the algorithm didn't move as it was stuck in a saddle point, which makes sense as the whole image would be symmetric and the starting position is at the center of the image. Even by changing the kernel size, the results will be the same.

### B. Mean-shift tracker

The tracker uses the mean-shift algorithm described above for updating the bounding box position. The tracker takes several parameters as input:

- **number of bins**: used in extracting histograms
- **epsilon**: a small constant used when calculating V
- **h**: the neighborhood that is used for mean-shift
- **sigma**: used when creating the kernel
- **alpha**: used for updating the model (q)

The size of the Epanechnikov kernel for the tracker is set to the same size as the patch extracted from the image in the initialization step.

In the tables bellow, we can see examples of how the implemented tracker preformed, using different parameters on the VOT(14) sequences.

| seq. | #bins | $\varepsilon$ | h | $\sigma$ | $\alpha$ | **FPS** | **failed** |
|---|---|---|---|---|---|---|---|
| torus | 16 | 1e-4 | 25 | 2 | 0.0 | 646.4 | 0 |
| torus | 32 | 1e-4 | 25 | 2 | 0.0 | 368.2 | 0 |
| torus | 16 | 1e-4 | 25 | 2 | 0.1 | 565.0 | 2 |
| torus | 16 | 1e-4 | 11 | 2 | 0.0 | 493.9 | 3 |
| torus | 16 | 1e-4 | 25 | 1 | 0.0 | 438.7 | 9 |

Table II: Tracker tested on torus sequences with different parameters.

In table II, we can see how each parameter effected the results. By increasing the number of bins, the tracker slows down. By lowering the neighborhood h to much, the tracker starts to fail. Even when setting alpha to be greater than 0, the

tracker starts to fail. However, the worst performance happens when sigma is decreased, focusing more on the center of the extracted patch. Setting the number of bins to 16 (optimal for speed), the neighborhood h to 25 (optimal in most cases), sigma to be greater than 1, meaning that the weights are more equally spread out throughout the patch, and without updating the mode, gave best performance overall on this set of sequences.

| seq. | #bins | $\varepsilon$ | h | $\sigma$ | $\alpha$ | **FPS** | **failed** |
|---|---|---|---|---|---|---|---|
| hand2 | 32 | 1e-4 | 25 | 2 | 0.0 | 330.0 | 0 |
| hand2 | 32 | 1e-3 | 33 | 3 | 0.0 | 328.9 | 0 |
| hand2 | 32 | 1e-3 | 21 | 2 | 0.0 | 311.9 | 2 |
| hand2 | 16 | 1e-5 | 33 | 4 | 0.1 | 374.2 | 4 |

Table III: Tracker tested on hand2 sequences with different parameters.

In table III, using higher number of bins, gave best results, as there is a lot of background noise and rapid movement.

| seq. | #bins | $\varepsilon$ | h | $\sigma$ | $\alpha$ | **FPS** | **failed** |
|---|---|---|---|---|---|---|---|
| polarbear | 8 | 1e-6 | 33 | 4 | 0.0 | 683.0 | 0 |
| polarbear | 16 | 1e-4 | 17 | 1 | 0.0 | 681.0 | 0 |
| polarbear | 16 | 1e-4 | 31 | 2 | 0.0 | 632.5 | 0 |
| polarbear | 16 | 1e-5 | 33 | 3 | 0.4 | 634.7 | 1 |

Table IV: Tracker tested on polarbear sequences with different parameters.

In table IV, we can see that most of the parameters give 0 fails. Only when increasing alpha, we get more fails than usual. This is because the bear moves a short distance and very slowly (we can use very low number of bins), so there isn't really any change that the model needs to adapt. Furthermore, if the kernel moves slightly off the bear, but still within bounds, the model will update more on the background, causing more fails.

| seq. | #bins | $\varepsilon$ | h | $\sigma$ | $\alpha$ | **FPS** | **failed** |
|---|---|---|---|---|---|---|---|
| gymnastics | 16 | 1e-4 | 25 | 2 | 0.0 | 461.3 | 0 |
| gymnastics | 16 | 1e-6 | 33 | 2 | 0.0 | 407.4 | 0 |
| gymnastics | 16 | 1e-4 | 25 | 5 | 0.1 | 459.8 | 1 |
| gymnastics | 32 | 1e-3 | 31 | 3 | 0.0 | 374.5 | 1 |

Table V: Tracker tested on gymnastics sequences with different parameters.

In table V, the gymnast we are tracking moves fast and does a lot of different movements. Here, increasing alpha should give better performance, but it give slightly worse. The reason could be that because the initial boundary box is static, and when the gymnast jumps and twists, there is a lot of noise introduced in the window, which the model learns, instead of the gymnast.

| seq. | #bins | $\varepsilon$ | h | $\sigma$ | $\alpha$ | **FPS** | **failed** |
|---|---|---|---|---|---|---|---|
| basketball | 16 | 1e-3 | 25 | 3 | 0.0 | 237.7 | 4 |
| basketball | 8 | 1e-4 | 17 | 1 | 0.0 | 642.8 | 3 |
| basketball | 8 | 1e-6 | 17 | 1 | 0.0 | 684.5 | 2 |

Table VI: Tracker tested on basketball sequences with different parameters.

In figure VI, we can see worse results than the previous examples. In this set of sequences (basketball), there is a lot of movement (slow and rapid), and there are a lot of sequences (just over 700). Because of this, we can truly test the capabilities of our tracker. We can see that by using a small number of bins and smaller neighborhood, with sigma of 1 (focusing more on the center of the window), we get better results. However, even when trying with a lot of different parameters, 2 fails was the best that was achieved.

Finally, one last set of examples using different color spaces can be seen in figure VII, where alpha is set to 0.

| seq. | space | #bins | $\varepsilon$ | h | $\sigma$ | **FPS failed** | |
|---|---|---|---|---|---|---|---|
| bicycle | RGB | 16 | 1e-3 | 21 | 1 | 582.7 | 3 |
| bicycle | HSV | 16 | 1e-3 | 25 | 2 | 462.1 | 1 |
| bicycle | LAB | 16 | 1e-3 | 25 | 4 | 544.1 | 3 |
| bicycle | YCrCb | 16 | 1e-3 | 25 | 4 | 580.6 | 4 |

Table VII: Tracker tested on bicycle sequences with different parameters.

In the example above, using HSV color space, achived better performance then the rest, with slightly different parameter values. Furthermore, the sequence set "david" was also tested with HSV against RGB, achieving 2x better results because of the change in lighting. By using HSV with 16 number of bins, 17 neighborhood, and sigma of 3, we get 2 fails for HSV and 4 fails for RGB.

HSV separates the color information into intuitive components which makes it easier to construct a color histogram that better represents the object being tracked. HSV color space is more robust to changes in lighting conditions, as changes in brightness or contrast affect only the Value component and not the Hue or Saturation components. This makes it more suitable for object tracking in real-world scenarios where lighting conditions can vary widely.

## III. Conclusion

In this assignment, the mean-shift tracker was implemented and tested on a variety of sequence sets. The optimal parameter values differed on the specifics of the video, such as the amount of noise in the window, the movement speed and object scaling. For the most part of the sequences that were tested, using 16 number of histograms, with around 20 neighborhood and sigma of 2 gave good results with 0-2 fails.