Object Tracking Using Mean Shift

COMP558 Project Report

School of Computer Science

McGill University
Montreal, Quebec, Canada

December 16, 2024

Katia Brenaut

Habiba Abdelrehim
Zina Kamel

Abstract

This report presents a classical computer vision technique for object tracking. The implemented method captures the target object in the initial video frame as a color histogram, which is then used by the Mean Shift algorithm to track the object across subsequent frames. The Epanechnikov kernel is used to construct the histogram, and the similarity between histograms is computed using the Bhattacharya's coefficient. Experimental results demonstrate reliable tracking performance and show promising directions for this algorithm. The potential of Mean Shift in other applications is further explored by implementing a segmentation algorithm based on Mean Shift. The report concludes by highlighting limitations of the Mean Shift algorithm such as difficulty handling fast object motion and brightness variations, while proposing areas of improvement and future work.

Contents

Introduction

Object tracking is a core computer vision problem that has been being tackled by a dynamic research community for decades. The reason behind such enthusiasm is an incredibly high variety of applications. To cite a few examples, tracking can be used for security and surveillance, as it was during the Covid19 pandemic in order to monitor crowds and check for social distancing. It can also be used in more critical situations such as autonomous vehicles, in order to detect obstacles and estimate a vehicle's speed. The common purpose of these examples is simple: from a given start position of an object on screen, we want to be able to automatically estimate the object's position through time.

While this problem may seem basic for a human being, it is actually far from trivial when it comes to implementing it on a computer. Indeed, the fundamental mathematics and algorithms behind this task that is natural to the human brain are yet to be understood, and many different models and approaches have been described in order to mimic them, each one coming with its set of benefits and drawbacks. The one we chose to focus on is called the Mean Shift algorithm, which comes with the advantages that it is robust, does not have a heavy computing cost, and does not require any training.

In this report, we shall discuss the theory behind the Mean Shift algorithm, and how we implemented it using color histograms, backprojection and Bhattacharya's coefficient for histogram comparison. Then, we shall see how we explored a few different applications such as image segmentation, and finally which benefits and weaknesses we encountered and how the latter could possibly be addressed in the future.

Statement of Contributions: We have collectively worked together on different parts of the projects. We each started with specific tasks whereby Katia implemented the Mean Shift and histogram functions, Zina implemented the back-projection, histogram helper function and segmentation while Habiba did the HSV implementation for the functions, histogram and video tracker code. After initial implementation, we have worked on all the parts collectively for debugging and improvements.

Literature

While the Mean Shift algorithm is widely used in object tracking, other methods have also been explored. Particle filters work well for non-linear and unpredictable motion but can be very slow and expensive to run, especially in high-dimensional data [DM96]. Optical flow methods track motion by analyzing pixel changes between frames and are good for dense motion tracking. These methods calculate the apparent motion of objects in a scene by detecting intensity changes in images. They are particularly useful for applications like video stabilization, object detection, and activity recognition [HS81]. However, optical flow often struggles with noisy data, rapid movements, or occluded objects where pixels are hidden or lost. Additionally, these methods require significant processing power, making real-time applications challenging, especially on devices with limited computational resources. Efforts to optimize optical flow, such as pyramid-based approaches, can reduce some of these issues, but the trade-offs in accuracy and speed remain. Deep learning approaches, like CNNs, Siamese networks, and R-CNNs, perform very well in complex situations, especially when the background is cluttered or objects are partially hidden. However, they need large labeled datasets, high computing power, and often do not work well in real-time on low-power devices [Chi20, ZLLS24].

Comaniciu and Meer (2002) in *Mean Shift: A Robust Approach Toward Feature Space Analysis* explain the Mean Shift algorithm as a powerful method for finding clusters and high-density areas in data [CM02]. The paper shows how the algorithm is good for tasks like object tracking and segmentation because it doesn't need labeled data or predefined clusters. Mean Shift works by moving points toward the areas of highest density using a kernel density estimation (KDE), which helps locate objects accurately. Compared to other methods, the Mean Shift algorithm is simpler and faster. It finds object positions step by step without needing labeled data or predefined cluster counts. It works well for both tracking and segmentation tasks. Recent improvements, like using the Bhattacharyya distance for better histogram matching and adaptive kernels to handle different features, have made it more reliable. OpenCV's version of the algorithm is a good example of how it can work in real-time. This report implements the Mean Shift algorithm for tracking and segmentation, based on Comaniciu and Meer (2002). We use ideas from the paper to test Mean Shift's performance and suggest improvements like adding more features and tuning parameters to make it work better for different tasks in computer vision.

Methods

In this section, we will cover the methods we have implemented in this project. We start by going over the steps to build the Mean Shift algorithm for object tracking namely, the histogram construction, back projection, and the Mean Shift tracking algorithm. Following the promising results we obtained, we were interested in testing the Mean Shift algorithm on applications other than tracking. Hence, we will also describe our implementation for a segmentation algorithm using Mean Shift.

Object Tracking with Mean Shift

Histogram Construction

The approach we used for this project requires comparing regions of an image to the object we are tracking. In order to perform these comparisons, we used color histograms.

A color histogram is a statistical tool used to analyze the repartition of the color values within the image. We divide the range of possible colors into a given number of intervals called bins, and for each interval, we count the proportion of pixels whose color falls into it.

The method described in the *Mean Shift: A robust approach toward feature space analysis* [CM02] paper also uses weights when computing the contribution of each point x_n to the corresponding bin. These weights are computed using the distance between x_n and y the center of the region, and a kernel profile k. This gives us the following formula for the value of each bin m in the histogram:

$$p_m = C_h \sum_n k\left(\left|\frac{y_0 - x_n}{h}\right|^2\right)\delta(b(x_n) - m)$$

where C_h is a normalization constant (so the elements of the histogram sum to 1), h is the radius of the region of interest and $b(x_n)$ is the index of the bin x_n falls into.

We are quite free regarding the choice of k. The paper suggests using the Epanechnikov kernel defined as follows:

$$K(x) = c(1 - |x|^2)$$

with c some constant.

This gives us the following profile:

$$k(x) = \begin{cases} 1 - x, & 0 \leq x \leq 1, \\ 0, & x > 1. \end{cases}$$

In order to have more accuracy when comparing images, instead of computing histograms over only one-dimensional grayscale values, we also took the colors into account. We could have used the RGB model and computed three-dimensional histograms, but this would

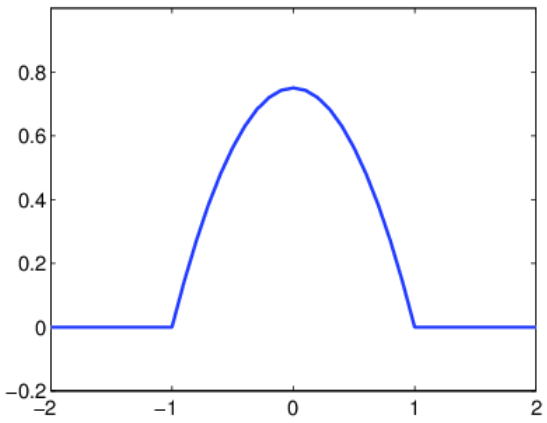


Figure 1: The Epanechnikov kernel [Pit06]

have been expensive. Instead, another model that is frequently used in this context is the HSV model (hue, saturation, value). By using only the hue and saturation components, we considerably increase accuracy (allowing us to make out objects that appear as bright but have different hues), while not increasing the cost of computation too much (having two-dimensional histograms).

Back Projection

After constructing the histogram for the object of interest, we will now use it in the subsequent frames to locate the target object. As will be described in details in the next subsection of the report, the Mean Shift algorithm iteratively converges towards the region of the highest likelihood. Hence, by creating a back-projected image where the pixels of the target object are highlighted, we can have the tracking window iteratively shift towards them. This is what the back-projection algorithm creates.

In a back-projected image, every pixel represents the probability that this pixel belongs to the target object. Hence, the intensity values for the pixels of the target object in this new frame will be higher than the other regions. By creating a normalized histogram for the region of interest in the previous step, we already have a function for the probability that a particular pixel belongs to this region. Hence, we can simply replace the pixel's value in the back-projected image with this probability.

The algorithm for the back-projection can be outlined as follows:

```
for every pixel (i,j) in the new frame:
    find the hue and saturation bin indices for this pixel in the histogram

    backprojected_image(i, j) = histogram value of the corresponding bins
```

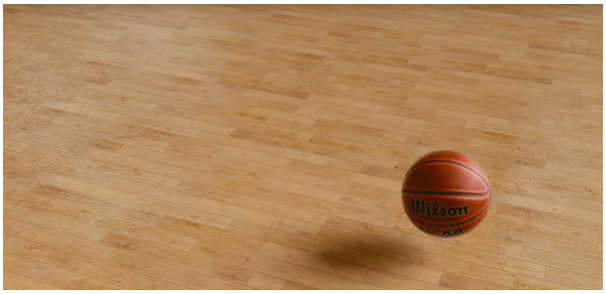
Following the algorithm above, we have a new image (backprojected_image) that stores the probability of every pixel belonging to the region of interest.

In order to test our implementation, we generate images for a few objects. We expect pixels for the region of interest to be brighter (higher intensity) than the objects that are less similar. We also compare the results with the back-projection method available in OpenCV. Sample results from one of the tests is shown below.

Back-Projection Test

In this test, we used the following images for a bouncing basketball. The first image shows the initial location of the ball, on which we create our region of interest histogram. We then apply the back-projection on the new location of the ball in the second image. Input images were taken from an online video [Ayo15]

(a) Initial Ball Location



(b) New Ball Location

Figure 2: Test Images Used

(a) Results from Our Backprojection Implementation

(b) Results from OpenCV Backprojection Implementation

Figure 3: Back-projected Images Generated

The results are well-aligned with what we would expect the back-projected image to look like: the pixels having the region of interest are of higher intensity than the remaining pixels. Results are also very close to the results obtained from OpenCV. Finetuning the parameters such as the boundaries of the region of interest and the number of bins of the histogram will further help in removing any outliers.

Mean Shift Tracking Algorithm

In this part we shall discuss the Mean Shift algorithm itself, which is the core of the method described by Comaniciu and Meer in their paper we based our approach on.

The idea behind Mean Shift is simple: we have a set of points spread within a region, and we look for the point of highest density. For this, we start from some guess point defining the center of a region of interest (ROI). This region of interest is smaller than the total region, which means only a few points will fall within. Then, we compute the mean of these points, and take this mean as the new guess. If we repeat this step enough times, the guess point will eventually converge towards the point of highest density of the distribution.

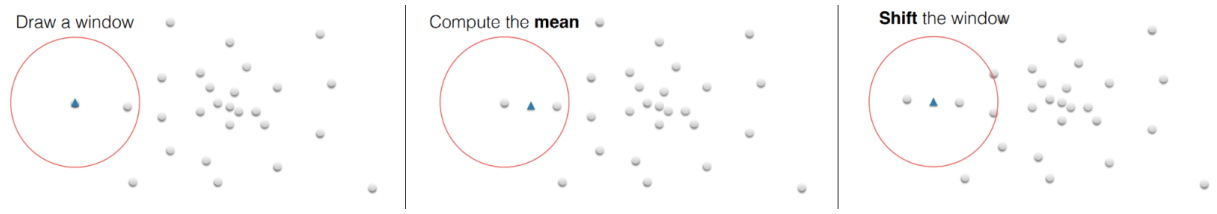


Figure 4: Illustration of the means shift algorithm. [Kit17]

We apply this simple algorithm to the context of tracking by applying weights to the points each time we compute the mean within the ROI. The weights are computed for each point x_n using both the distance of x_n to the center y_0 of the ROI, and a comparison between the histogram q of the tracked object (the one we got on the first frame) and the histogram p centered on y_0. For this, we use the following formula:

$$w_n = \sum_m \sqrt{\frac{q_m}{p_m(y_0)}} \delta(b(x_n) - m)$$

where the sum iterates over all the possible bin indexes m in the histogram. $b(x_n)$ is the bin index of x_n (based on its color and computed in the same way as the histogram). This formula was derivated using Bhattacharya's coefficient for comparing histograms.

Since the Kronecker delta will be equal to one only once, we can actually rewrite this formula as follows:

$$w_n = \sqrt{\frac{q_m}{p_m(y_0)}}$$

with m being the exact bin index of x_n. This removes the need for unnecessary loops in the code.

Next, we compute the new guess point y_1 by computing the weighted mean of the x_n:

$$y_1 = \frac{\sum_n w_n x_n g\left(\left|\frac{y_0 - x_n}{h}\right|^2\right)}{\sum_n w_n g\left(\left|\frac{y_0 - x_n}{h}\right|^2\right)}$$

with $g = -k'$ and h the radius of the ROI.

Finally, if $|y_1 - y_0| < \epsilon$, we stop iterating and consider the algorithm has converged. Otherwise, we repeat the same steps by replacing y_0 by y_1.

To summarize, this gives us the following algorithm:

```
y0 = initial guess
q = histogram of the tracked object
dist = infinity
while dist > epsilon
    py0 = histogram(y0)
    for each point xn
        m = bin index for xn
        wn = sqrt(q(m) / py0(m))
    y1 = sum(xn * wn * g(norm_squared((y0-xn)/h)) / sum(wn *
        g(norm_squared((y0-xn)/h))
    dist = distance between y1 and y0
```

Since we worked with the Hue and Saturation components of the HSV color models, our histograms were actually two-dimensional, so the bin index m would actually be two numbers. But the algorithm is pretty much the same as described above.

Mean Shift for Segmentation

Mean Shift algorithm is not specific for object tracking, and can be used for other applications such as Image Segmentation. Hence, after implementing it for object tracking, we were curious to adapt our algorithm for image segmentation.

A famous algorithm for clustering is the K-means Algorithm [JH10]. Although it is commonly used, it has few limitations such as the need for specifying the number of clusters "K" beforehand. On the other hand, segmentation with Mean Shift allows us to overcome this limitation.

We follow a similar approach to the Mean Shift algorithm used in tracking: we initialise a window at every pixel and iteratively compute the Mean Shift vector. This will shift the window towards the region of the highest density until convergence. Similar pixels will converge to the same "peak" or cluster. Hence, pixels are assigned to the cluster they converge to. The algorithm can be summarised in the following steps:

Before running the below algorithm, we set up a vector feature space whereby every vector is a combination of the color and spatial (coordinate values) of the pixel. This will allow us to compare the similarity between two feature vectors in the search space. This ensures that pixels that are both visually similar and spatially close are grouped together.

```
for every pixel:
    repeat until convergence or max interations:
            center a window at this pixel

            compute the normalised difference between current feature vector
               and all feature vectors, in both color and spatial domains

            compute Gaussian kernel weights

            compute the new mean for the current window using the weights
               computed

            shift the window to the new mean computed
    assign the pixel to the cluster it converged to
```

We test the algorithm on some sample images, one of which is shown below:

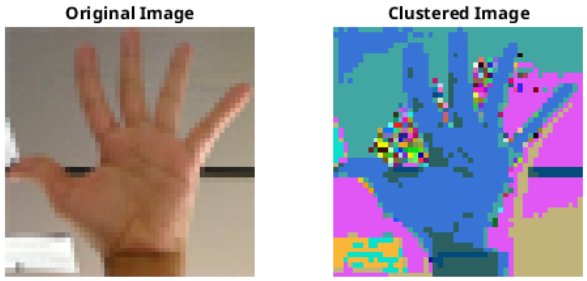


Figure 5: Segmentation Results

Based on the results, the algorithm seems to be working well, grouping similar pixels together. There are some inaccuracies especially in the background where we expect to have fewer clusters than what is in the result. One way to imporve the results would be to adjust the parameters such as the color and spatial bandwidth, number of iterations and the threshold. Performing a grid search over these parameters would help determine more optimal ones. Moreover, we can handle the difference in densities by assigning different bandwidth to different regions. For instance, we can reduce the bandwidth in dense regions to capture fine details and increase it in sparse regions. We could also merge similar clusters together after the segmentation is done, which would help improve the clustering for regions similar to the background in our test image. Finally, we are currently considering the spatial and color features only but it would also be interesting to experiment with including additional features such as textures or edges during clustering.

Results

In this section, we present the results for the Mean Shift algorithm for object tracking.

We first tested our implementation on single pairs of images, using the one-dimensional approach with only grayscale images.

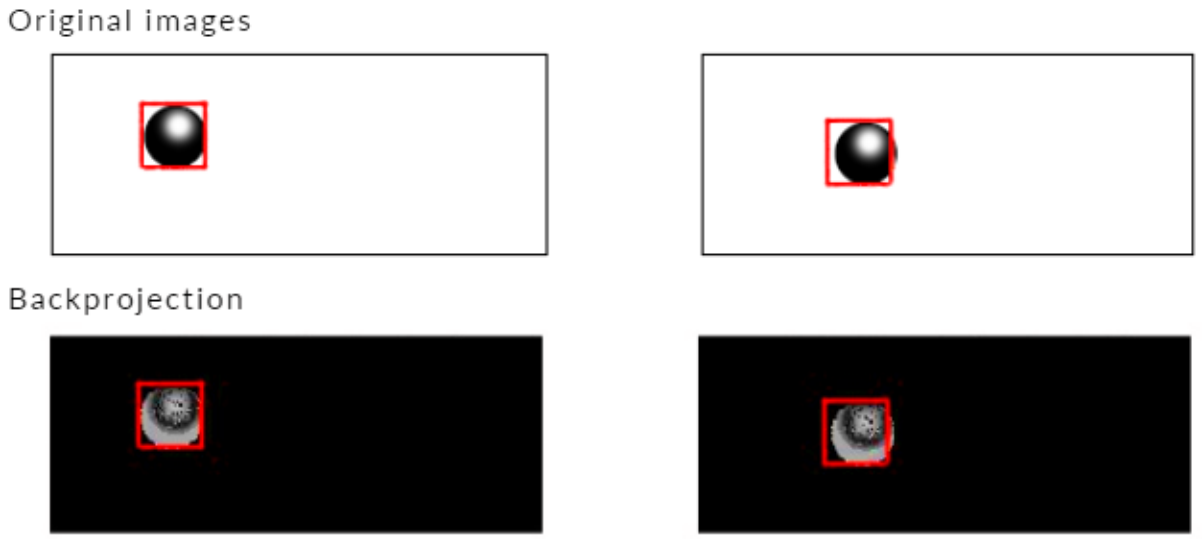


Figure 6: tracked ball between two frames, grayscale

Figure 7: tracked car between two frames, grayscale

We can see that the red square, which represents the computed position of the tracked object, satisfyingly moves in the same direction as the object, even if it does not match it perfectly.

For the second example, we had to transform the image of the car to make it as square as possible. Indeed, when the tracked object is too long, the region of interest (which, although it is represented as a square here for better visibility, is in reality a circle) includes too many pixels belonging to the background if we try to include the full object. On the other hand, if we make it smaller to include only a part of the object and no background, the quality of the tracking is altered. So this method works best when the tracked object is round or square (in other words, compact).

After having implemented tracking using HSV, this is what we got:

Figure 8: tracked ball between two frames, HSV

Figure 9: tracked car between two frames, HSV

We can see that the red square is following the object better when we take the colors into account.

Then, we implemented tracking on a video by iterating over the frames and comparing them two by two.

The first test video [1] is a simple moving ball we generated using Blender. Running our algorithm on it produced the following results [2]

[1]Input test video: `https://drive.google.com/file/d/1-c1AuhfoOGTEbxIcPUoC7ICjKeDRn2VF/view?usp=drive_link`

[2]Results on sample video test: `https://drive.google.com/file/d/1RyjX4jiFdGpefgaUTf8UCwz2nwatX_vv/view?usp=drive_link`

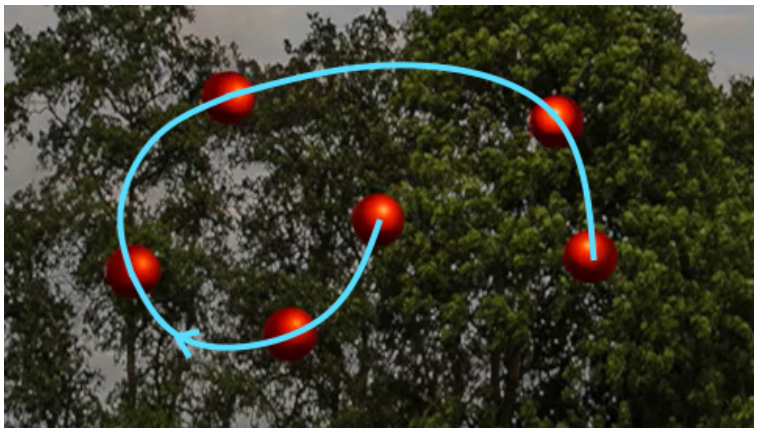


Figure 10: Video of a ball moving (in blue, the path of the ball, drawn for illustration only and not present in the video)

Figure 11: The result of our algorithm

We can see that the red square is following the ball perfectly, so the algorithm clearly works well on this simple example.

Discussion and Conclusion

Based on the results obtained, the algorithm seems to be working well. However, there are some limitations or areas of failure that we will touch upon in this section, while giving some suggestions for possible improvements and future work.

Firstly, the algorithm seems to fail when the object is moving fast. This can be seen in our test video for the tennis ball [3], during which the tracking worked accurately until the ball started moving very fast, as shown in the pictures below. And even when it slowed down, it was too far from the estimated position to allow recovery, and the algorithm got stuck at an incorrect estimation until the ball came close to the estimated position again.

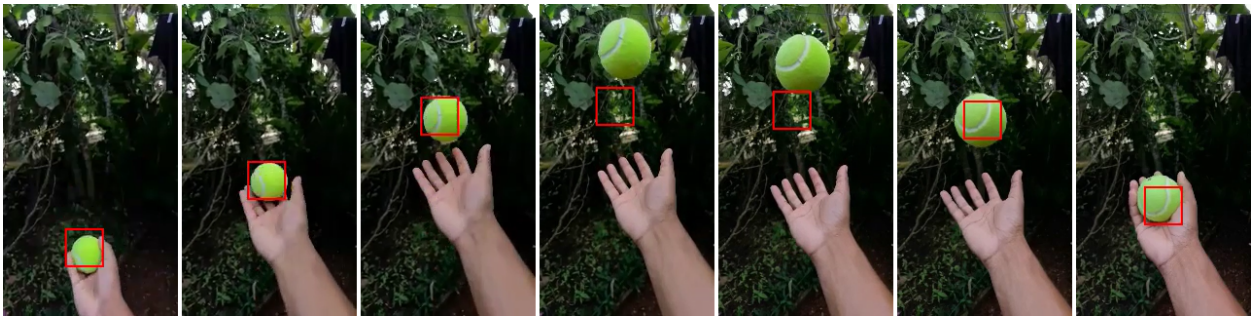


Figure 12: The result of our algorithm. Input Video Source: [Cha22]

We wanted to confirm this limitation by comparing the results we obtained with OpenCV implementation. Hence, we ran the same video using OpenCV's built-in methods for Mean Shift. The results obtained [4], also had the failed scenario we observed at the same instance as our results, confirming the limitation in the algorithm. Mean Shift relies on small, iterative updates and assumptions of smooth motion, so when the object moves fast, the algorithm may fail to correctly shift the window to the object's new position as the object might be outside the region where it searches for peaks. Moreover, when the object moves fast, Mean Shift might get stuck in a local maximum rather than the true target location. Some improvements would be to tune the parameters such as the search window size or the number of bins for the histogram. We could also integrate methods such as Kalman Filter to predict the ball's likely position in the next frame based on its velocity and trajectory. This can help us have a better estimate for the initialization of the Mean Shift search window closer to the true new location. This would also help us deal with occlusion, another limitation with the current approach. Additionally, we could increase the frame rate of the video to reduce the displacement of the object between consecutive frames.

[3]Our results on the tennis ball video: `https://drive.google.com/file/d/1Fa7evPL-6bD6zjmmhjbr0hKaub0gThTs/view?usp=sharing`

[4]OpenCV's results on the tennis ball video: `https://drive.google.com/file/d/1jekzjWzFM67JFwoBTY9XXRGG7-kvd9iA/view?usp=sharing`

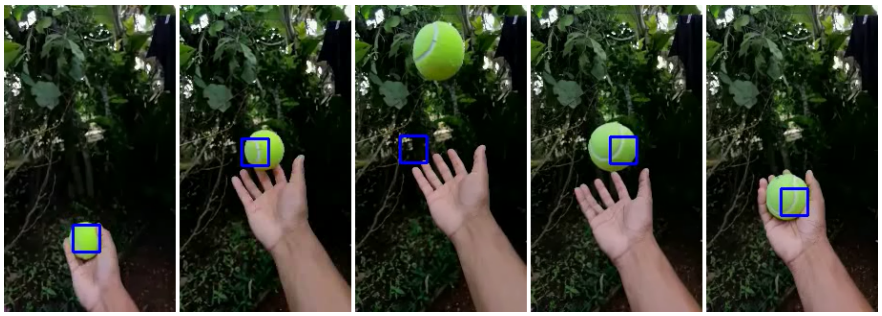


Figure 13: The result of MeanShift tracking with OpenCV. Input Video Source: [Cha22]

Another limitation we recognised is that the algorithm does not perform very well when the objects are of similar hue (which is the case with this green tennis ball in front of a green background) or when the brightness is varying across frames. One way to overcome this would be to incorporate additional features, such as brightness (which would mean working with three-dimensional histograms as mentioned in the histogram section), edges, or texture, in combination with the color histograms to have a better representation of the object and be able to distinguish it from similarly looking objects.

In conclusion, although the method has some limitations, it has proved to give accurate and promising results in several instances, especially when the object movement is moderate, and the color contrast is clear. Its effectiveness and computational efficiency make it a practical choice for real-time applications where more advanced and resource-intensive algorithms may not be feasible.

Bibliography

[Ayo15] Z. Ayoub. Basketball bouncing animation in blender. https://www.youtube.com/watch?v=m1p6AuY6wLU, 2015.

[Cha22] Unic Slow Motion Channel. Tennis ball slow motion video. https://www.youtube.com/watch?v=f_FYoHI4yes, 2022.

[Chi20] Davide Chicco. Siamese neural networks: An overview. In *Artificial Neural Networks: Methods and Applications*, volume 2190 of *Methods in Molecular Biology*, pages 73–94. Springer Protocols, Humana Press, 2020.

[CM02] Dorin Comaniciu and Peter Meer. Mean shift: A robust approach toward feature space analysis. https://courses.csail.mit.edu/6.869/handouts/PAMIMeanshift.pdf, 2002.

[DM96] Pierre Del Moral. Non linear filtering: Interacting particle solution. *Markov Processes and Related Fields*, 2(4):555–580, 1996.

[HS81] Berthold K. P. Horn and Brian G. Schunck. Determining optical flow. *Artificial Intelligence*, 17(1-3):185–203, 1981.

[JH10] Xin Jin and Jiawei Han. *K-Means Clustering*, pages 563–564. Springer US, Boston, MA, 2010.

[Kit17] Kris Kitani. Mean-shift tracker. https://www.cs.cmu.edu/~16385/s17/Slides/15.2_Tracking__Mean_Shift.pdf, 2017.

[Pit06] F. Pitié. Statistical signal processing techniques for visual post-production. https://www.researchgate.net/figure/Epanechnikov-Kernel-This-is-the-function-Ku-3-41-u-2-for-1-u-1-and-zero_fig5_277288545, 2006.

[ZLLS24] Aston Zhang, Zachary Lipton, Mu Li, and Alexander J. Smola. *Dive into Deep Learning*. Cambridge University Press, 2024.