Improving DroTrack: Insights and Advancements for Drone-based Tracking

Alexandre St-Aubin, Alexis Viau-Cardinal, Rafid Saif

December 27, 2024

Abstract

The increased deployment of Unmanned Aerial Vehicles (UAVs) has amplified the demand for robust object tracking methods, essential for applications like search and rescue, surveillance, and agriculture. State-of-the-art feature-based trackers such as DroTrack integrate optical flow and segmentation to achieve real-time performance in drone-based scenarios. However, they suffer from significant limitations, including poor resilience to occlusion, unreliable recovery, and inefficient handling of size and orientation changes. In this work, we propose an improved tracking pipeline that combines feature-based tracking with DroTrack's optical flow framework, enhancing its robustness to occlusions, high object velocity, and orientation changes. Our experiments on the VisDrone dataset compare our approach against DroTrack and other baseline methods, evaluating tracking accuracy and runtime performance. Results demonstrate that our method improves tracking robustness, particularly in scenarios with occlusion and fast object speeds, albeit at the cost of reduced frame rate. Videos of our approach are available at Improving_Drotrack. The code is available at this url.

I. Introduction

THE use of Unmanned Aerial Vehicles (UAVs), or drones, has increased significantly in recent years across various fields due to their affordability, ability to access remote areas, and minimal risk. A key feature in drones is object tracking, with applications in search and rescue [1], [2], agriculture [3], military operations [4], and aerial surveillance [5].

At its core, object tracking involves determining the trajectory of an object within the image plane as it moves through a scene. Essentially, a tracker ensures that consistent labels are assigned to objects across different frames in a video. In this report, we focus on *single object tracking*. Typically, bounding boxes are used to locate regions of interest such as people or vehicles across image sequences, starting from an initial target location [6]. This technology is key in UAV applications such as ground strikes, criminal vehicle tracking, UAV landings [8], and search and rescue [2]. It can also be used in aerial surveillance to estimate highway traffic flow, enabling proactive traffic management [5], and much more.

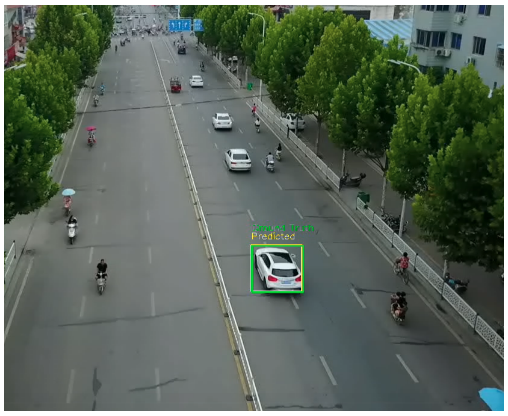

Fig. 1: Tracking a car with *DroTrack* on a video from the *VisDrone* dataset [7].

While single object tracking is a crucial task in many applications, it remains highly challenging in the field of computer vision [9]. Classical object tracking (i.e. with fixed cameras), faces challenges such as illumination variation, background clutter, low resolution, scale variation, occlusion, target rotation or deformation, and fast motion of the object [10]. These challenges are further amplified in drone-based tracking. The high-altitude perspective of UAVs makes low-resolution inevitable, making object detection and tracking more difficult. Additionally, the rapid movements and rotations of UAVs introduce frequent scale variations and distortions, with objects often appearing rotated or deformed. As if it could not get worse, real-time processing is often needed for practical applications, requiring algorithms to maintain both high accuracy and efficiency.

Object tracking methods can be separated in 4 main classes: *Learning-based*, *Estimation-based*, *Segmentation-based*, and *Feature-based*, as seen in Figure 2. *Learning-based* methods involve training large models to recognize the appearance of specific targets, allowing them to detect and track these objects during testing. While they offer

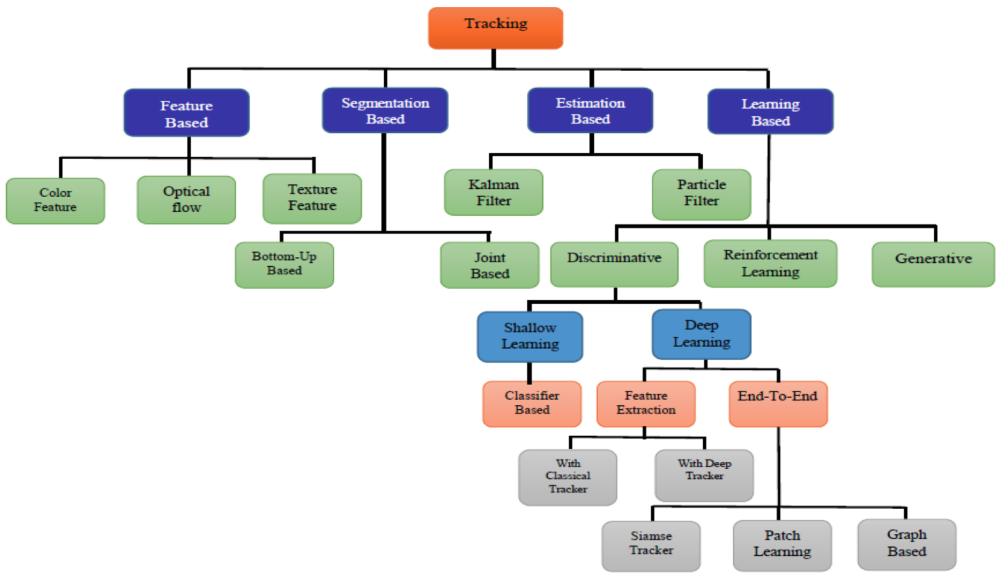

Fig. 2: Classification of object tracking methods, reproduced from [10].

higher accuracy, they are more complex, require extensive training data and computational resources, and often face challenges with real-time tracking. Some of the first works using learning based methods defined tracking as a binary classification problem [11]. *Estimation-based* methods approach tracking as a dynamic state estimation problem, often using Bayesian filters to recursively predict and update an object's position based on sensor data in a two-step process: prediction and updating [10]. An example of this process is the use of Kalman filters for tracking, as in [12]. *Segmentation-based* methods use algorithms to separate objects of interest from the background to track them [13], [14]. Finally, *feature-based* object tracking extracts distinctive features like color, texture, or *optical flow* to identify the most similar object in the next frame. Its main challenge is ensuring the extracted features are precise and reliable to differentiate the target from others [10].

The method we present here, inspired from *DroTrack* [15], leverages both feature-based and learning-based tracking. We choose *DroTrack* as it is one of the recent works using mostly classical methods specifically on drone-based tracking. From this method, we tune parts of it and incorporate methods from [16] in order to improve robustness. Overall, we see promising results in terms of tracking accuracy over several frames, however, the time taken to process each frame is currently too long.

II. Background

In this section, we first give a formal statement of the problem and performance expectations. Subsequently, we cover a few tracking algorithms and associated computer vision theory.

A. Problem Definition

In this paper, we address the problem of Single Object Tracking (SOT), which involves tracking an individual target over time. The core definition remains the same, whether tracking is performed using a static camera or a drone. In the first frame I_0, the user specifies a target bounding box, $\beta_0 = [x_0, y_0, h_0, w_0]$, that contains the object to be tracked in subsequent frames. At each iteration k, the algorithm receives the previous bounding box β_{k-1} and the current frame I_k. The goal is to compute the bounding box β_k reflecting as accurately as possible the same target in the new frame. In the case of done footage, the camera itself can move with 6 degrees of freedom. The camera movement induces many issues such motion blur, background noise, fast object motions in the projected plane, etc. This means that traditional trackers like the optical-flow based Lucas-Kanade tracker are more likely to fail. In this case, the failure is likely associated to the discrepancy between the first order approximation used in Lucas-Kanade and the ground truth.

B. *Expectations*

Having defined the problem, we now outline key performance expectations for a tracking algorithm:

1) **Robustness**: The tracker must handle challenges such as occlusions, illumination changes, background clutter, scaling, and deformation, which are especially prevalent in drone-based tracking scenarios.
2) **Real-time processing**: Given the real-time nature of many drone-based tracking applications, the algorithm should process and output a sufficient number of frames per second to ensure timely and reliable performance.

Meeting these expectations is essential for using tracking algorithms in real-world applications. A robust tracker ensures reliable performance in different environments and conditions. For example, in a search and rescue mission at sea, a tracker designed to follow a drowning person should not lose focus if the person briefly goes underwater. Real-time processing is equally important, as it allows the system to respond quickly to dynamic situations and unpredictable situations. These factors are key to assessing the effectiveness and practicality of a tracking system.

C. *Feature-Based Object Tracking*

Feature-based object tracking is the concept that forms the foundation of our approach. It involves two main steps: *feature extraction* and *feature matching* [17]. Extracting good features is extremely important, as no feature-based algorithm can be effective if the features it is tracking aren't robust. Matching features is particularly challenging because a feature in one image may have many similar counterparts in another, causing confusion. To address this, some algorithms use exhaustive search and correlation over a wide area, but these methods are computationally expensive and not suitable for real-time tracking. On the other hand, the Kanade-Lucas-Tomasi (KLT) feature tracker [18]–[20], limits the search to a small set of potential matches, reducing computational complexity. However, this approach may lose track of feature points over long image sequences. To mitigate this, the authors of [17] suggest combining feature tracking with a Sum-of-Squared Difference criterion. In *DroTrack* [15], feature tracking is integrated with classification to help maintain object tracking, as discussed in section II-H.

1) Feature Extraction: There are various types of features used in object tracking, but they all share the requirement of being as unique as possible to ensure easy distinction across frames. The first feature we discuss is *color*, which can be represented using a histogram, as described in [21]. This method begins by specifying a bounding box, computing its color histogram, and then searching for a similar histogram in the next frame. A drawback of using color as a feature is that it doesn't account for shape, meaning that two shapes may have an almost identical histogram, leading to potential confusion [10].

Edges detect strong changes in image intensities, typically occurring at object boundaries. Edges are less sensitive to illumination changes compared to color features, making them particularly useful for tracking object boundaries [22]. The most popular edge detection approach is the Canny Edge detector [23], known for its simplicity and accuracy.

Another commonly used feature in object tracking is *corners*, which are points in an image where two *edges* intersect, making them highly distinctive and reliable for tracking. The Harris corner detector [24] is one of the most widely used methods for detecting these points. It is based on the concept that corners can be identified by analyzing the local image structure using second-order image derivatives. Harris' method evaluates the eigenvalues of the autocorrelation matrix of the image gradient to find regions where both eigenvalues are large, indicating a corner. Similar methods include the Shit-Tomasi corner detector [25]. The advantage of corner-based tracking is that it is less sensitive to changes in illumination and can be effective in dynamic environments.

Texture is a feature that measures the intensity variation across a surface, capturing properties such as smoothness and regularity. Unlike color, texture requires additional processing to generate descriptors. Various texture descriptors exist, including Gray-Level Cooccurrence Matrices [26], or wavelets [27]. Like edges, texture features are less sensitive to illumination changes compared to color. They are also invariant to illumination, rotation, scale, and translation [10].

Beyond the features discussed, advanced feature detection techniques like *SIFT* (Scale-Invariant Feature Transform) [28], *SURF* (Speeded-Up Robust Features) [29], *BRISK* (Binary Robust Invariant Scalable Keypoints) [30], and *ORB* [31]. These methods are particularly robust for detecting and matching features across frames under challenging conditions, such as changes in scale, rotation, and illumination. These techniques not only detect features, but also characterizes them with a feature descriptor vector. SIFT is known for its precision and robustness, though it is computationally expensive, making it less suitable for real-time applications. SURF is a faster alternative to SIFT, trading off some accuracy for improved speed, it is used in [32] for object tracking. BRISK, on the other hand,

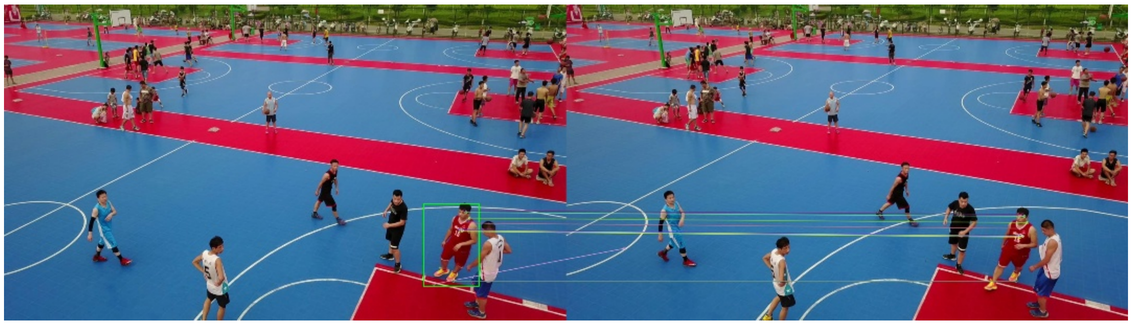
Fig. 3: SIFT matching to track a target across frames

focuses on computational efficiency, using binary descriptors for feature matching, making it well-suited for real-time systems. Finally, *ORB*, the feature descriptor that we opted for in our method, claims to be two orders of magnitude faster than *SIFT*, while maintaining similar performance, a behavior that we also observed.

2) Feature Matching (tracking): Once good features are extracted, the next step is to establish correspondences between those features across consecutive frames. This process, known as feature tracking or matching, ensures that the algorithm can reliably follow an object as it moves through the scene. Effective feature tracking forms the backbone of many object tracking systems, enabling accurate localization of targets over time.

A widely used method for feature tracking is *optical flow*. It represents a dense field of displacement vectors, indicating the translation of each pixel between consecutive frames. In other words, Optical Flow is an approximation of the 2D motion field, a projection of the 3D velocities in the world scene [33]. It is computed using the brightness constraint, which assumes that corresponding pixels maintain their brightness over time [22], described by the equation

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

By applying the first-order Taylor expansion, the equation becomes

$$I(x,y,t) + I_x\Delta x + I_y\Delta y + I_t\Delta t = I(x,y,t),$$

allowing the displacement vectors $(\Delta x, \Delta y)$ to be calculated for each pixel. Unlike feature-based methods, which provide displacements only for selected feature points, *optical flow* estimates dense motion across the entire image. The concepts of *optical flow* are widely used in tracking algorithms such as in the KLT algorithm [18], [19], which leverages optical flow, but in local regions of the image, rather than all pixels.

SIFT Flow, building on the principles of optical flow, adapts its framework by matching SIFT descriptors rather than raw pixel intensities [34]. In this approach, each pixel is represented by a SIFT descriptor, capturing local image structure and contextual details. A discrete algorithm that preserves discontinuities is employed to estimate the flow and match these descriptors across two images. By leveraging SIFT features, this method enables robust correspondence, even when there are variations in the appearance of scenes or objects. It is also possible to use other features like *SURF*, *ORB*, or *BRISK* for this.

D. Histogram-Based Tracking (Mean Shift and CAM Shift)

Mean Shift [35] is a classical computer vision method enabling the tracking of color distinctive targets. First, a color-space histogram is created from the labeled region of interest in the first frame. This histogram is generally normalized for robustness to changes in illumination. For the next frames, the algorithm creates a probability map of the new bounding box location by comparing color distribution. This is achieved by back projecting the subsequent frame into the previously computed histogram distribution The mean is then iteratively shifted to the regions of higher probability. This process is repeated until the shift is smaller than some pre-defined threshold ε or a maximum number of iterations. The last mean position becomes the new center for the bounding box.

While this algorithm deals well with many problems such as motion blur, mean shift does not update the scale of the predicted bounding box or adjust the predicted orientation. These issues are handled by Continuously Adaptive Mean (CAM) Shift [36]. In this method, Mean Shift is first used to find a predicted target bounding box as before. An ellipse is then fitted to the region to find the appropriate orientation and size of the new bounding box. The rescaling of the bounding box is governed by variation in the probability map computed previously.

E. Efficient Subwindow Search (ESS)

Efficient sub-window search (ESS) [37] is a method that speeds up the computation of a score-maximizing sub-window. Traditionally, given a square search area of size n, testing all sub-windows would take $O\left(n^4\right)$ iterations, which is not feasible, especially in a scenario requiring real-time tracking. To remedy this issue, the ESS method adopts a branch-and-bound approach, allowing to reduce the number of iterations to $O\left(n^2\right)$, a significant performance improvement. The method described in Lampbert et al. [37] goes over the creation of an upper bound function from an arbitrary objective function allowing to bound the performances of multiple sub-windows.
Let $\mathscr{B}$ be the set of all bounding boxes. Given an arbitrary score function $f : \mathscr{B} \mapsto \mathbb{R}$, we need to define a function $\hat{f} : 2^{\mathscr{B}} \mapsto \mathbb{R}$ to apply the ESS algorithm. This function needs to have the following two properties: (1) for any set of bounding boxes $B \subset \mathscr{B}$, we get the upper bound $\hat{f}(B) \geq \max_{b\in B} f(b)$, and (2) whenever $B = \{b\}, b \in \mathscr{B}$, the equality $\hat{f}(B) = f(b)$ holds. In some cases, it is possible to explicitly define $\hat{f}$ using the definition of f. In their paper, Lampbert et al. chose f to be a support vector machine (SVM) classifier defined as follows.

$$f(I) = \beta + \sum_i \alpha_i \langle h, h_i \rangle \tag{1}$$

Here, h is the histogram of the image region I, and h_i are the histograms of the (quantized) training data. The α_i and β are the learned weights. Then, $\hat{f}$ is defined as follows

$$\hat{f}(B) = f^+(b_{max}) - f^-(b_{min}) \tag{2}$$

In the above equation, b_{max} is the largest area contained in B whereas b_{min} is the smallest area. The function $f^+(x)$ is defined as the sum of positive terms in $f(x)$, while $f^-(x)$ is the sum of negative terms. It is easy to see that this $\hat{f}$ indeed respects the two prescribed properties. This summation trick for many functions f and will come into play in section II-F.
To find the score-maximizing subwindow, the algorithm works with a set of bounding boxes defined as $BBOX = [B, T, L, R]$, namely a 4-tuple of intervals representing the range of possible values for the bottom, top, left, and right boundaries respectively. Splitting such set of bounding boxes is achieved by splitting in half the largest spanning interval (choosing one when multiple are qualifying). The full algorithm is given below.

Algorithm 1 Efficient Subwindow Search

```
1: procedure FINDING SCORE-MAXIMIZING WINDOW(image $I \in \mathbb{R}^{n\times m\times 3}$, quality bound function $\hat{f}$)
2:     $P \leftarrow$ Empty Priority Queue
3:     $[B,T,L,R] \leftarrow [0,n] \times [0,n] \times [0,m] \times [0,m] \times$
4:     while $[B,T,L,R]$ isn't a single bounding box do
5:         $\text{split}_1, \text{split}_2 \leftarrow$ Split the set $[B,T,L,R]$
6:         Push $\left(\hat{f}(\text{split}_1), \text{split}_1\right)$ to $P$
7:         Push $\left(\hat{f}(\text{split}_2), \text{split}_2\right)$ to $P$
8:         $[B,T,L,R] \leftarrow$ pop best state from $P$
9:     return $[B,T,L,R]$
```

F. Online Nearest Neighbor Classifier

In our approach, we leveraged the work of Gu et al. [16]. This is a feature-based tracker that uses SIFT features (generalizes to any descriptor-based feature detector). When receiving a frame, the algorithm classifies feature points by comparing the point descriptions (not location). Points with description closer to the descriptors associated with the tracked object than to descriptors associated with the background are assumed to be part of the same target. The optimal bounding box is found using the Efficient Subwindow Search (ESS) described in section II-E. The classifier is constantly updated with the new target descriptors. This method handles very well occlusion but fails with dealing with motion blur.
Before presenting the algorithm's procedure, let us first define a few functions. Let the function $V : R^{n\times m} \mapsto \mathbb{R}^2 \times \mathbb{R}^{128}$ be mapping a grayscale image I to its associated pair of key points location ($\mathbb{R}^2$) and SIFT key points descriptors ($\mathbb{R}^{128}$). We call $\Theta(B,I) = \left\{v \in \mathbb{R}^{128} | (x,v) \in V(I) \wedge x \in B\right\}$ the function that maps a bounding box B and an image I to the points descriptors of the picture contained within the bounding box. The function F_λ is used to discriminate points that share more similarities with tracked object than with the background. Let $A, B, C \subset \mathbb{R}^{128}$ be sets of key

points descriptors. Let $NN_X : \mathbb{R}^{128} \mapsto \mathbb{R}^{128}$ be the function mapping a key point descriptor to the closest value in X. This can be done efficiently using KD trees.

$$F_\lambda(A,B,C) = \{v \in A | \, \|v - NN_B(v)\| < \lambda \, \|v - NN_C(v)\|\} \tag{3}$$

The above function finds the points in A that are λ closer to points in B than they are to points in C. Finally, we define the performance function of our tracker as follows. Below, we refer to the frame that we are currently processing as I_k. The set O_{k-1} represents the points associated with the tracked object in the previous frame. Effectively, points are given a lifetime of τ frames, so O_{k-1} corresponds to the feature descriptors of the tracked object in the last τ frames. The set T represents the points associated with the background. The variable b represents the bounding box for which we want to evaluate the performance function, while b_{k-1} represents the bounding box of the previous frame.

$$\mu\left(b; O_{k-1}, T, b_{k-1}, I_k\right) = -\kappa\left(b, b_{k-1}\right) + \sum_{v \in \Theta(W, I_k)} \operatorname{sign}\left(F_\lambda\left(\{v\}, O_{k_1} T\right)\right) \tag{4}$$

In the above, the sign(A) function takes value 1 if $A \neq \emptyset$ and -1 otherwise. In other words, points v exhibiting greater similarity to the tracked object are assigned a value of 1, while those associated with the background are assigned a value of -1, Furthermore, the function κ defines a penalty based on the change in bounding box.

$$\kappa(b_1, b_2) = \gamma(\|O_1 - O_2\| + |h_1 - h_2| + |w_1 - w_2| + s(b_1, b_2)) \tag{5}$$

The O_i are the centroids of the respective bounding boxes b_i while the h_i and w_i are their height and width. The function s is a penalty on the change of aspect ratio defined as follows.

$$s(b_1, b_2) = \max\left\{\left|\frac{h_1}{w_1} - \frac{h_2}{w_2}\right|, \left|\frac{w_1}{h_1} - \frac{w_2}{h_2}\right|\right\}$$

The tracker defined in Gu et al. finds the bounding box b_k maximizing equation 4. This is achieved using ESS. Although the definition of the objective function differs from the one in equation 1, we can still use the trick of splitting sums demonstrated in equation 2. Our upper bound equation is thus given as follows.

$$\hat{\mu}(B) = \mu^+\left(b_{max}\right) + \mu^-\left(b_{min}\right) - \min_{b \in B} \kappa\left(b, b_{k-1}\right) \tag{6}$$

In the above, the μ^+ represents the sum of positive terms in equation 4, while μ^- is the sum of negative terms (without the κ penalty). With these functions in mind, we can define the full algorithm.

Algorithm 2 Object Tracking (Gu et al.)

```
1: procedure TRACKING WINDOW(image sequence $\{I_k\}_{k=1}^n$, initial bounding box $b_1$)
2:     $O_{\text{queue}} \leftarrow$ initialize empty double-ended queue of size $\tau$
3:     Append $\Theta(I_1, b_1)$ to $O_{\text{queue}}$
4:     $T \leftarrow V(I_1) \setminus \Theta(I_1, b_1)$
5:     for $k \in 2, \ldots, n$ do
6:         $b_k \leftarrow$ Compute $\text{argmax}_b\ \mu(b, O_{\text{queue}}, T, b_{k-1}, I_k)$ using ESS
7:         Append $F_\lambda\left(b_k, O_{\text{queue}}, T\right)$ to $O_{\text{queue}}$
```

G. Fuzzy C-Means Segmentation

A naive approach to picture segmentation is to use the fuzzy C-means algorithm [38]. Given N pixels x_i, the goal of this approach is to find n centroids c_i minimizing the following cost function.

$$J_{FCM} = \sum_{j=1}^{N} \sum_{i=1}^{n} u_{i,j}^m \|x_i - c_i\|^2 \tag{7}$$

In the above equation, the $u_{i,j} \in [0,1]$ represents our degree of confidence that the pixel x_j belongs to the group represented by the centroid c_i. In this case, 1 represents certain membership, while 0 represents certain exclusion. However, contrarily to the usual C-means algorithm, the method described in Tripathy et al. [38] incorporates pixel location in the value updates. More precisely, when updating the $u_{i,j}$, instead of simply considering the distance of the pixel x_j with the centroid c_i, the neighboring membership values are also included. This contribution is capped by the hesitation degree in each pixel value, which is induced by the intermediate values (between 0 and 1 excluded) of the membership values. The full algorithm is given below.

Algorithm 3 Image segmentation using Fuzzy C-Means

```
1: procedure SEGMENT IMAGE(image $I$ (with $N$ pixels), number of centroids $n$)
2:     Initialize the centroids $c_i$ for $i \in 1, \dots, n$
3:     Initialize the initial membership value $u_{i,j} \leftarrow 1$
4:     while Goal not reached do
5:         Compute the membership function $u_{i,j} \leftarrow \frac{1}{\sum_{k=i}^{n} \left( \frac{\|x_j - c_i\|}{\|x_j - c_k\|} \right)^{\frac{2}{(m-1)}}}$
6:         Compute the hesitation $\pi_{i,j} \leftarrow 1 - u_{i,j} - \frac{1 - u_{i,j}}{1 + \lambda \cdot u_{i,j}}$
7:         Compute the hesitant membership function $u_{i,j} \leftarrow u_{i,j} + \pi_{i,j}$
8:         Compute the neighboring weights $h_{i,j} \leftarrow \sum_{k \in \mathrm{NB}(x_j)} u_{i,k}$
9:         Compute the weighted average of neighboring scores $u_{i,j} = \frac{u_{i,j}^{p} \; h_{i,j}^{q}}{\sum_{k=1}^{n} u_{k,j}^{p} \; h_{k,j}^{q}}$
10:        Update the centroids with a weighted average $c_i \leftarrow \frac{\sum_{j=1}^{N} u_{i,j}^{m} x_j}{\sum_{j=1}^{N} u_{i,j}^{m}}$
```

In their paper, Tripathy et al. halt the process when the update to $u_{i,j}$ becomes smaller than some threshold. While this segmentation technique does not allow for tracking, it is a fast and decent image segmenter. This technique is at the heart of the DroTrack algorithm described in section II-H.

H. DroTrack

DroTrack [15] is a framework leveraging many tracking techniques designed to handle drone footage. In this framework, the first step is to compute feature points that can be tracked using optical flow. In this case, Shi-Tomasi [25] features are used to detect corners within the annotated bounding box from the first frame. In subsequent frames, these feature coordinates are tracked with optical flow. Here, the Lucas-Kanade optical flow method [19] is utilized.

After extracting these initial features, the framework adjusts the bounding box to better fit the detected points. This step is necessary because the distribution of detected corners may not be perfectly aligned with the center of the initial bounding box. To address this, DroTrack first computes the initial displacement $\Delta_0 = c_0 - c_p$, as the difference between the center of the annotated bounding box, c_0, and the center of mass of the detected points, c_p, excluding outliers. For subsequent frames, the inverse process is applied. More specifically, the center of the distribution of tracked points c'_p is computed. Then, the center of the new bounding box is computed as $c_k = c'_p + s_{k-1} \cdot \Delta$. Here, s is a scaling factor expressing the size change of the bounding box between the first frame and the last computed bounding box. Effectively, this value is given as $s_k = (h_k \cdot FH_0) / (FH_k \cdot h_0)$ where h_i is the height (vertical span) of the bounding box at frame i and FH_i is the height (number of vertical pixels) of the image.

While Lucas-Kanade optical flow does yield decent results, it is possible that it loses track of its target. Some of the possible cause of failures include occlusion and quick motion. Another issue is that this tracker only models translation, which might be too naive, especially when the camera is susceptible to move closer or farther from its target. This situation can fortunately be detected fairly easily by checking the number of points that are still contained in the bounding box. Whenever it loses track of its target, the DroTrack framework employs a two steps recovery process which first flags possible replacement regions of interests, and subsequently evaluates their similarity to the previous region. This is achieved by first segmenting the picture using the Fuzzy C-Means algorithm (see section II-G). The output of this process yields a segmented picture and large areas associated with a single centroid define candidate bounding boxes. Then, a CNN classifier (VGG16) [39] is run on the both the previously tracked frame and each candidate bounding box. Comparing the results using cosine similarity allows selection of the new region of interest.

Finally, the algorithm scales the bounding box according to its motion in the visual field of the drone. Simply put, vertical changes (Δ_y) in position affect the size of the box. The size is increased when the target box moves down in the picture and decreased when it moves up. Changes on the horizontal axis (Δ_x) do not incur any change of size. For intermediate values, the rate of change is linearly adjusted according to the angle formed by the (Δ_x, Δ_y) point and the negative Y-axis.

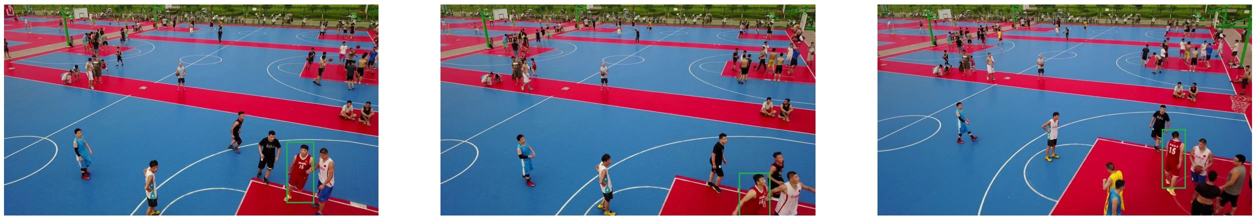
Fig. 4: Scenes with changing object appearance. Tracking is achieved as described in section III-A

III. Methods

To address the shortcomings of existing methods, we propose a novel tracking pipeline inspired by *DroTrack* but incorporating significant modifications to improve performance in challenging conditions such as oclusion and rapid velocity changes. Our method leverages a dual-tracker architecture that combines computationally efficient optical flow with a more robust, feature-based tracker, modified from Gu et al. This hybrid approach enhances the ability to handle occlusions, rotation, and changes in target appearance while minimizing computational overhead. We also introduce several adjustments, such as improved penalty functions, subwindow search optimizations, and a refined segmentation technique, to address key limitations in DroTrack. These changes are designed to improve target resilience, maintain tracking accuracy over longer time horizons, and adapt to drone-specific scenarios.

A. Key point Trackers

We created a tracker by matching SIFT key points. To improve the performances, we only limit key points search to a neighbourhood of pixels around the previous bounding box. The bounding box is chosen to cover the 10 closest key point matches to the tracked object key points set S. Two different approaches can be used to define the set S. For the first approach, we compute the key points at the initial frame to form S and compare the key points of every subsequent to this S. The second approach is to iteratively update the set S will all key points contained in updated bounding bounding boxes. This method should better handle rotation since minor angle changes can accumulate to form. This intuition is confirmed by the results with obtained in figure 4. However, we suspect that this method will likely add points associated with the background when performing the update step.

B. Histogram-Based Trackers

We compared performance with histograms-based trackers, more specifically Mean Shift and CAM shift. We used the implementations provided in OpenCV [40]. In our experiments we found that these methods require careful calibration of the color ranges when computing the feature histograms at the initial step. To do this, we use the k-means algorithm to segment the ROI into several clusters, and we pick the cluster with the highest bin count as the dominant color of the target. While this technique performs decently well, our experiments have shown that if the bounding box contains any of the background, then the resulting dominant color is often erroneous. To circumvent this issue, we use the intuition that the target is more likely to be in the center of the bounding box. Thus, we compute the dominant color by taking a subwindow half the size of the annotated bounding box at the first window.

C. DroTrack

To evaluate the improvements of our method, we compare its performance with the DroTrack pipeline. Fortunately, the authors of the paper have provided their code and we used their implementation on our dataset to judge its performances. They also provided the classifier network used for the target recovery step.

D. Our Method

In this paper, we introduce a tracking method based on the DroTrack framework. At its core, our tracker also uses Lucas-Kanade optical flow to track the target. We use the same method as DroTrack to adjust the bounding box location based on the distribution of the Shi-Tomasi key points. However, the recovery step is replaced with the online classifier tracker described in Gu et al. (see section II-F) We run this feature-descriptor based tracker (more computationally expensive) in parallel at a much slower frame rate.

We removed the angular scaling step from the pipeline. We judge that this approach is a naive way of handling size change, which is highlighted by the fact that in practice, the rate of change is multiplied by an arbitrary constant. Instead, we prefer relying on the classifier tracker which should handle size changes on its own. Since we do not add points to the optical flow tracker except when doing a recovery step, we are not worried about adding invalid points and thus scaling of the bounding box becomes less important.

We think that this method can outperform the one described in DroTrack since the usage of the feature descriptor based tracker allows to both handle occlusion and discriminate amongst similar entities. Indeed, comparing the CNN classification of candidate bounding boxes does not guarantee matching targets. A simple example arises from footage of a crowded environment. If the tracker is following an individual and looses track of its target due to occlusion, the classification approach is likely going to select another less-occluded individual in the crowd even though they might have completely different clothes, with different patterns.

1) Algorithm Description: Below is a brief overview of the proposed method.

Algorithm 4 Image tracking

```
1: procedure TRACK INITIAL BOUNDING BOX(image sequence $\{I_k\}_{k=1}^n$, bounding box $b_1$)
2:     $points \leftarrow$ Shi-Tomasi points of $I_1$ within $b_1$
3:     $\Delta \leftarrow$ Center difference between $points$ and $b_1$
4:     $gu \leftarrow$ Initialize online classifier tracker with image $I_1$ and bounding box $b_1$
5:     for $k \in 2, \dots, n$ do
6:         $points \leftarrow$ Lucas-Kanade optical flow with $points$
7:         $b_k \leftarrow$ Compute bounding box using $points$ and $\Delta$
8:         if $|points \cap b_k| < \varepsilon$ then
9:             $b_{k,\text{candidate}}$, $error \leftarrow$ Query (do not update internal values) $gu$ for bounding box
10:            $segmented \leftarrow$ Segment a neighborhood of $b_{k-1}$
11:            for $area \in$ Large single valued areas of $segmented$ do
12:                $b$, $e \leftarrow$ Query $gu$ assuming previous bounding box was $area$
13:                if $e < error$ then
14:                    $b_{k,\text{candidate}}$, $error \leftarrow b, e$
15:            $b_k \leftarrow b_{k,\text{candidate}}$
16:            $points \leftarrow$ Shi-Tomasi points of $I_k$ within $b_k$
17:        if mod $(k, n_{gu}) = 0$ then
18:            Update $gu$ with frame $I_k$
```

The method instantiates and updates two trackers, one computationally fast but less accurate (optical flow), and one computationally expensive but resilient (online classifier). Contrarily to the work of Gu et al., we used ORB keypoints [41] given they are much faster to compute The resilient tracker is queried only when we lose confidence in the output of the fast tracker, but occasionally updated. Although not strictly necessary, we used image segmentation to find candidate regions. These regions are used to alleviate the penalty κ induced by the change of window location and size (see equation 5), instead focusing more on feature matching. We still allow the online classifier tracker to edit the candidate region since the segmenter performs a naive division of the image.

2) Tweaks and Performances Improvements: Although this work mostly reapplied already proven methods, we performed a few tweaks to better suit our application.

Efficient Subwindow Search: Early tests showed that the Gu tracker is likely to choose a very narrow window when tracking relatively small targets. This is because the performance function 4 only requires key points center to be contained in the bounding box. However, SIFT or ORB points descriptor also include a size, roughly indicating the spatial frequency at which the detected feature occurred We altered the performance function so that a point gets credited only if fits entirely (including its size) in the bounding box.

Without this adjustment, the algorithm often returned windows which were too small in one direction and thus did not contain any Shi-Tomasi key points. While the algorithm could still recover at the next step by once again querying the online classifier tracker, we prefer using optical flow methods as often as possible since they are much less computationally expensive.

Online Classifier Tracker: We slightly adjusted the penalty associated with bounding boxes changes defined in equation 5. Indeed, we added a multiplicative coefficient to each term allowing better control over the penalty associated with each each change.

$$\kappa(b_1,b_2) = \gamma(\gamma_O \|O_1 - O_2\| + \gamma_h |h_1 - h_2| + \gamma_w |w_1 - w_2| + \gamma_s s(b_1,b_2)) \tag{8}$$

We ended up penalizing more changes in aspect ratio than changes of location and dimension. Indeed the high degree of freedom of a drone camera is likely going to induce large changes of location and size.
Furthermore, in their paper, Gu et al. do not update the background set, judging it unnecessary as it is unlikely to change overtime. We do update it since the background is bound to change with the movements of the drone. Although we have tried keeping the last τ background sets and perform the F_λ comparison (see equation 3) pairwise in term of background sets and tracked object sets, keeping a single updated background set seemed to perform more reliably.

Fuzzy C-Means Segmenter: Contrarily to the one used in DroTrack, our implementation of the Fuzzy C-Means segementer analyzed colour pictures and not grayscale ones. As described in Tripathy et al., the segmentor isn't limited to grayscale picture and in fact generalize to any dimension using the vector norm. We think that this change could improve the separation of regions having the same grayscale value, but completely different hues. Furthermore, we halted the loop after a certain number of iteration, not by monitoring the updates. This is done to more accurately cap the runtime of the algorithm.

IV. Experiments

Our experiments aim to answer the following research questions:

1) Does a drone-specific tracker such as DroTrack boast any advantages over generic trackers?
2) If so, what makes it better?
3) For failure cases, what are the causes for failure?
4) Is it possible to tweak certain modules of the chosen algorithm to obtain better results?

A. Experimental setups

For our experiments, we use the *VisDrone* labeled dataset [7]. The dataset contains sequences of drone footage with associated annotations. The tracking algorithm begins with as input the first ground truth bounding box.

B. Evaluation Criteria

To assess the performance of our method and compare it with *DroTrack*, we used the the following evaluation metrics. Let the ground truth bounding box be denoted by β^G and the predicted bounding box by β^P. Their respective areas are represented as $\mathscr{R}^G$ and $\mathscr{R}^P$.

1) Center Error: The Center Error (**CE**) computes the Euclidean distance between the ground truth bounding box center $C(\beta^G)$ and the predicted bounding box center $C(\beta^P)$. That is,

$$E_c(\beta^G,\beta^P) = \left\|C(\beta^G) - C(\beta^P)\right\|_2 \tag{9}$$

This metric is not very useful on its own, but will be used to compute the *Tracking Length* in section IV-B3.

2) Average Overlap Score (AOS): The *Intersection over Union* (IoU), or overlap score, is defined as,

$$\mathbf{IoU}(\mathscr{R}^G,\mathscr{R}^P) = \frac{|\mathscr{R}^G \cap \mathscr{R}^P|}{|\mathscr{R}^G \cup \mathscr{R}^P|}, \tag{10}$$

where $|\mathscr{R}^G \cap \mathscr{R}^P|$ is the area of overlap, and $|\mathscr{R}^G \cup \mathscr{R}^P|$ is the area of union between the two bounding boxes. This concept is illustrated in Figure 5. The *Average Overlap Score* (AOS) aggregates the IoU values across all frames. It is given by,

$$\text{AOS} = \frac{1}{n}\sum_{i=1}^{n} \mathbf{IoU}(\mathscr{R}_i^G,\mathscr{R}_i^P), \tag{11}$$

where n is the total number of frames. The AOS metric ranges from 0 to 1, providing a measure of the average alignment between the ground truth and predicted bounding boxes across all frames, with higher values indicating better performance.

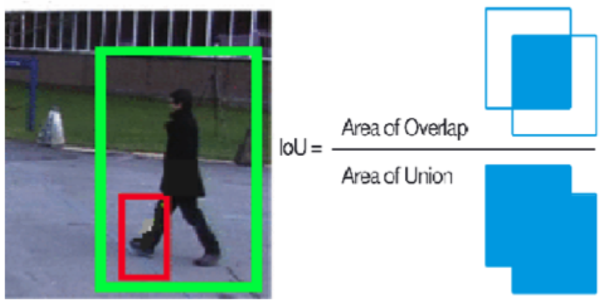

Fig. 5: Overlap score (IoU) between two boxes. Reproduced from [10].

3) Tracking Length: Tracking length (**TL**) measures the number of frames a tracker successfully processes before the first failure. Failure is defined when a specific threshold (τ) is exceeded, such as for the center error (E_c) or IoU (**IoU**) falling below the threshold:

$$\text{Failure: } E_c > \tau_{\textbf{CE}} \quad \text{or} \quad \textbf{IoU} < \tau_{\textbf{IoU}}. \tag{12}$$

The choice of threshold (τ) significantly impacts the results and must be carefully selected. A limitation of this metric is that it disregards frames after the first failure, focusing only on the initial tracking sequence (the tracker may find its way back to the object after having surpassed the threshold).

V. Results

We evaluate the models on a subset of the *VisDrone* dataset. The average runtime and frames per second (FPS) are given in Table I

Method	Runtime (s)	FPS
SIFT First Frame	0.040	24.90
SIFT Frame-to-Frame	0.043	23.33
Mean Shift	0.036	27.51
CAM Shift	0.032	31.02
DroTrack	0.055	18.11
Ours	0.049	20.55

TABLE I: Runtime and FPS performance of tested trackers

Method	AOS	TL
SIFT First Frame	0.137	107.00
SIFT Frame-to-Frame	0.083	103.16
Mean Shift	0.028	19.95
CAM Shift	0.010	3.465
DroTrack	0.175	200.77
Ours	0.124	329.71

TABLE II: AOS and TL of tested trackers

Given that the drone footage is captured at 30 FPS, we can assume that is the minimum required for any real-time tracker. If the tracker is used as feedback for a controller, we would need to perform more processing within between each frame, thus this is a conservative benchmark. Most of our methods fall short of this benchmark. Unfortunately, both robust detectors (DroTrack and Ours) have low FPS. While this may be optimized further to improve the performance, this is a clear limitation for both approaches currently.

Next, we compare *DroTrack* and **Ours** on the *VisDrone* dataset by evaluating the **CE** metric at each iteration. We use a threshold of $\tau_{\textbf{CE}} = 25$ to calculate the **TL**, and present the results in fig. 6. On average, **Ours** outperforms *DroTrack* with a difference of **128.94** in tracking length across all runs, demonstrating the superior robustness of our method.

We further evaluate the AOS by computing the average **IoU** for each run. The results, shown in table III, reveal a more nuanced comparison. While **Ours** achieves higher AOS in 15 out of 34 runs, *DroTrack* performs better in the majority of cases, particularly on sequences such as 043_00377_s and 049_00435_s. Notably, **Ours** demonstrates significant improvements on challenging runs like 070_02088_s, 071_00816_s, and 091_01035_s, where the AOS is substantially higher.

It is important to note a limitation of the **IoU** metric: it assigns a value of zero whether the estimated bounding box is slightly misaligned (e.g., hanging by a corner) or entirely off-target. This can obscure the fact that **Ours**

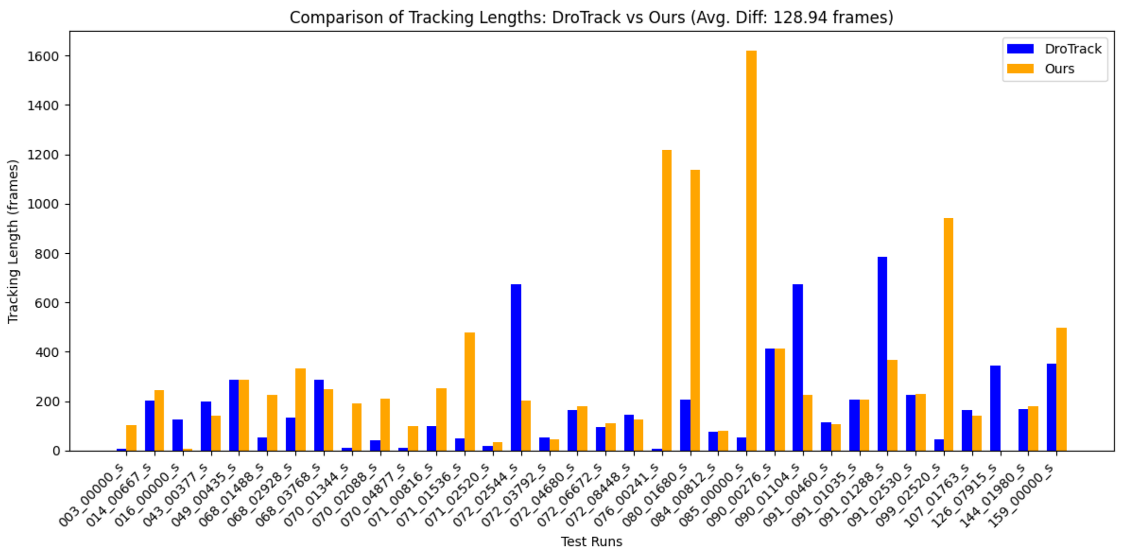

Fig. 6: Comparison of **TL**'s between **Ours** and *DroTrack* across tracking tasks from *VisDrone.*

often produces bounding boxes closer to the ground truth, as reflected by the superior **TL** metric 6. However, in scenarios where both methods successfully track the object, **Ours** may not always achieve higher precision than *DroTrack*.

TABLE III: AOS Comparison Between *DroTrack* and **Ours**. The highest AOS per run is in **bold**.

Run	DroTrack AOS	Ours AOS
003_00000_s	**0.010**	**0.010**
014_00667_s	0.188	**0.210**
016_00000_s	0.053	**0.192**
043_00377_s	**0.095**	0.019
049_00435_s	**0.323**	0.134
068_01488_s	**0.120**	0.083
068_02928_s	**0.134**	0.014
068_03768_s	**0.376**	0.165
070_01344_s	**0.058**	0.000
070_02088_s	0.057	**0.298**
070_04877_s	0.066	**0.122**
071_00816_s	0.126	**0.396**
071_01536_s	**0.107**	0.000
071_02520_s	0.011	**0.018**
072_02544_s	**0.229**	0.022
072_03792_s	0.076	**0.087**
072_04680_s	0.296	**0.296**
072_06672_s	**0.234**	0.031
072_08448_s	**0.368**	0.329
076_00241_s	**0.006**	**0.006**
080_01680_s	**0.114**	0.046
084_00812_s	0.050	**0.088**
085_00000_s	**0.089**	0.003
090_00276_s	**0.295**	0.002
090_01104_s	**0.254**	0.066
091_00460_s	0.088	**0.107**
091_01035_s	0.234	**0.388**
091_01288_s	**0.382**	0.112
091_02530_s	**0.307**	0.253
099_02520_s	0.020	**0.028**
107_01763_s	**0.258**	0.223
126_07915_s	0.279	**0.290**
144_01980_s	**0.681**	0.099
159_00000_s	**0.111**	0.080

VI. Conclusion

In this paper, we applied classical vision algorithms to address the SOT problem in drone-based tracking. By fine-tuning components of the algorithm, we achieved notable improvements in tracking performance, particularly in terms of TL. Across all test videos, our method demonstrated an average improvement in TL of 128.94. However, the results for AOS were mixed, indicating potential areas for refinement in handling challenging tracking scenarios.

Despite these advancements, both our method and DroTrack did not meet the stringent real-time requirements of 30 FPS. This limitation underscores the need for further optimization in algorithmic efficiency.

For future work, an important direction would be to focus on reducing computational overhead to meet real-time performance benchmarks. Investigating implementations in C or C++, which are better suited for time-critical applications, could provide valuable insights into runtime efficiency. Additionally, exploring hardware-accelerated solutions such as GPU or FPGA-based processing might unlock further improvements. Finally, expanding the dataset to include more diverse scenarios could help improve the generalization and robustness of the algorithm.

Contributions:

- Alexandre: Report, literature review, testing script, analysis of results, website.
- Alexis: Report, algorithm improvements, conception, implementation of our algorithm.
- Rafid: Report, testing, parts of algorithm, conception, implementation of compared algorithms.

References

[1] Y. Xu, X. Li, X. Meng, and W. Zhang, "An iterated greedy heuristic for collaborative human-uav search of missing tourists," *Knowledge-Based Systems*, vol. 286, p. 111409, 2024. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0950705124000443

[2] A. Chikwanha, S. Motepe, and R. Stopforth, "Survey and requirements for search and rescue ground and air vehicles for mining applications," in *2012 19th International Conference on Mechatronics and Machine Vision in Practice (M2VIP)*, 2012, pp. 105–109.

[3] X. Liu, G. Li, H. Yang, N. Zhang, L. Wang, and P. Shao, "Agricultural uav trajectory planning by incorporating multi-mechanism improved grey wolf optimization algorithm," *Expert Systems with Applications*, vol. 233, p. 120946, 2023. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0957417423014483

[4] J. Park and I. Guvenc, "Interference analysis for uav radar networks with guard zones based on stochastic geometry," *IEEE Transactions on Aerospace and Electronic Systems*, vol. 59, no. 4, pp. 4092–4104, 2023.

[5] E. Carroll and D. Rathbone, "Using an unmanned airborne data acquisition system (adas) for traffic surveillance, monitoring, and management," 01 2002.

[6] J. Zhao, G. Xiao, X. Zhang, and D. P. Bavirisetti, "A survey on object tracking in aerial surveillance," *Lecture Notes in Electrical Engineering*, 2018. [Online]. Available: https://api.semanticscholar.org/CorpusID:132210615

[7] P. Zhu, L. Wen, D. Du, X. Bian, H. Fan, Q. Hu, and H. Ling, "Detection and tracking meet drones challenge," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 44, no. 11, pp. 7380–7399, 2021.

[8] S. Yang, S. A. Scherer, K. Schauwecker, and A. Zell, "Onboard monocular vision for landing of an mav on a landing site specified by a single reference image," in *2013 International Conference on Unmanned Aircraft Systems (ICUAS)*, 2013, pp. 318–325.

[9] O. Abdelaziz, M. Shehata, and M. Mohamed, "Beyond traditional single object tracking: A survey," 2024. [Online]. Available: https://arxiv.org/abs/2405.10439

[10] Z. Soleimanitaleb and M. A. Keyvanrad, "Single object tracking: A survey of methods, datasets, and evaluation metrics," 2022. [Online]. Available: https://arxiv.org/abs/2201.13066

[11] S. Avidan, "Ensemble tracking," in *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, vol. 2, 2005, pp. 494–501 vol. 2.

[12] P. Gunjal, B. Gunjal, H. Shinde, S. Vanam, and S. Aher, "Moving object tracking using kalman filter," 02 2018, pp. 544–547.

[13] F. Schubert, D. Casaburo, D. Dickmanns, and V. Belagiannis, "Revisiting robust visual tracking using pixel-wise posteriors," 07 2015.

[14] V. Belagiannis, F. Schubert, N. Navab, and S. Ilic, "Segmentation based particle filtering for real-time 2d object tracking," in *European Conference on Computer Vision*, 2012. [Online]. Available: https://api.semanticscholar.org/CorpusID:3179336

[15] A. Hamdi, F. Salim, and D. Y. Kim, "Drotrack: High-speed drone-based object tracking under uncertainty," 2020. [Online]. Available: https://arxiv.org/abs/2005.00828

[16] S. Gu, Y. Zheng, and C. Tomasi, "Efficient visual object tracking with online nearest neighbor classifier," in *Computer Vision – ACCV 2010*, R. Kimmel, R. Klette, and A. Sugimoto, Eds. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011, pp. 271–282.

[17] B. Han, W. Roberts, D. Wu, and J. Li, "Robust feature-based object tracking," *Proceedings of SPIE - The International Society for Optical Engineering*, 04 2007.

[18] C. Tomasi, "Detection and tracking of point features," 1991. [Online]. Available: https://api.semanticscholar.org/CorpusID:238434334

[19] B. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision (ijcai)," vol. 81, 04 1981.

[20] J.-Y. Bouguet, "Pyramidal implementation of the lucas kanade feature tracker," 1999. [Online]. Available: https://api.semanticscholar.org/CorpusID:9350588

[21] M. Fotouhi, A. R. Gholami, and S. Kasaei, "Particle filter-based object tracking using adaptive histogram," in *2011 7th Iranian Conference on Machine Vision and Image Processing*, 2011, pp. 1–5.

[22] A. Yilmaz, O. Javed, and M. Shah, "Object tracking: a survey. acm comput surv," *ACM Comput. Surv.*, vol. 38, 12 2006.

[23] J. Canny, “A computational approach to edge detection,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. PAMI-8, no. 6, pp. 679–698, 1986.

[24] C. G. Harris and M. J. Stephens, “A combined corner and edge detector,” in *Alvey Vision Conference*, 1988. [Online]. Available: https://api.semanticscholar.org/CorpusID:1694378

[25] J. Shi and Tomasi, “Good features to track,” in *1994 Proceedings of IEEE Conference on Computer Vision and Pattern Recognition*, 1994, pp. 593–600.

[26] R. M. Haralick, K. Shanmugam, and I. Dinstein, “Textural features for image classification,” *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 6, pp. 610–621, 1973.

[27] S. Mallat, “A theory for multiresolution signal decomposition: the wavelet representation,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 11, no. 7, pp. 674–693, 1989.

[28] D. Lowe, “Distinctive image features from scale-invariant keypoints,” *International Journal of Computer Vision*, vol. 60, pp. 91–, 11 2004.

[29] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, “Speeded-up robust features (surf),” *Computer Vision and Image Understanding*, vol. 110, no. 3, pp. 346–359, 2008, similarity Matching in Computer Vision and Multimedia. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314207001555

[30] S. Leutenegger, M. Chli, and R. Y. Siegwart, “Brisk: Binary robust invariant scalable keypoints,” in *2011 International Conference on Computer Vision*, 2011, pp. 2548–2555.

[31] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.

[32] M. M. Monisha R, “Feature based moving object detection and tracking,” in *2017 International Journal of Innovative Trends and Emerging Technologies*, vol. 2, 2017.

[33] J. Barron, D. Fleet, S. Beauchemin, and T. Burkitt, “Performance of optical flow techniques,” in *Proceedings 1992 IEEE Computer Society Conference on Computer Vision and Pattern Recognition*, 1992, pp. 236–242.

[34] C. Liu, J. Yuen, and A. Torralba, “Sift flow: Dense correspondence across scenes and its applications,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 33, no. 5, pp. 978–994, 2011.

[35] D. Comaniciu and P. Meer, “Mean shift: a robust approach toward feature space analysis,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 24, no. 5, pp. 603–619, May 2002.

[36] G. R. Bradski, “Computer vision face tracking for use in a perceptual user interface,” 1998. [Online]. Available: https://api.semanticscholar.org/CorpusID:2928235

[37] C. H. Lampert, M. B. Blaschko, and T. Hofmann, “Beyond sliding windows: Object localization by efficient subwindow search,” in *2008 IEEE Conference on Computer Vision and Pattern Recognition*, 2008, pp. 1–8.

[38] B. K. Tripathy, A. Basu, and S. Govel, “Image segmentation using spatial intuitionistic fuzzy c means clustering,” in *2014 IEEE International Conference on Computational Intelligence and Computing Research*, 2014, pp. 1–5.

[39] K. Simonyan and A. Zisserman, “Very deep convolutional networks for large-scale image recognition,” 2015. [Online]. Available: https://arxiv.org/abs/1409.1556

[40] G. Bradski, “The OpenCV Library,” *Dr. Dobb’s Journal of Software Tools*, 2000.

[41] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, “Orb: An efficient alternative to sift or surf,” in *2011 International Conference on Computer Vision*, 2011, pp. 2564–2571.