ROBUST OBJECT TRACKING AND ADAPTIVE DETECTION FOR AUTO NAVIGATION OF UNMANNED AERIAL VEHICLE

Soumyaroop Nandi

Submitted to the Department of Electrical Engineering and Computer Science and the Graduate Faculty of the University of Kansas in partial fulfillment of the requirements for the degree of Master of Science.

Thesis Committee:

Chairperson Dr. Richard Wang

Dr. James Rowland

Dr. James Stiles

Date Defended: 01/09/2017

The Thesis Committee for Soumyaroop Nandi certifies

that this is the approved version of the following thesis:

ROBUST OBJECT TRACKING AND ADAPTIVE DETECTION FOR AUTO NAVIGATION OF UNMANNED AERIAL VEHICLE

Thesis Committee:

Chairperson Dr. Richard Wang

Dr. James Rowland

Dr. James Stiles

Date approved: 01/09/2017

Abstract

Object detection and tracking is an important research topic in the computer vision field with numerous practical applications. Although great progress has been made, both in object detection and tracking over the past decade, it is still a big challenge in real-time applications like automated navigation of an unmanned aerial vehicle and collision avoidance with a forward looking camera. An automated and robust object tracking approach is proposed by integrating a kernelized correlation filter framework with an adaptive object detection technique based on minimum barrier distance transform. The proposed tracker is automatically initialized with salient object detection and the detected object is localized in the image frame with a rectangular bounding box. An adaptive object redetection strategy is proposed to refine the location and boundary of the object, when the tracking correlation response drops below a certain threshold. In addition, reliable pre-processing and post-processing methods are applied on the image frames to accurately localize the object. Extensive quantitative and qualitative experimentation on challenging datasets have been performed to verify the proposed approach. Furthermore, the proposed approach is comprehensively examined with six other recent state-of-the-art trackers, demonstrating that the proposed approach greatly outperforms these trackers, both in terms of tracking speed and accuracy.

Acknowledgements

I want to express my heartfelt gratitude to my advisor Dr. Richard Wang, who has guided me throughout my Master’s program at the University of Kansas. I want to thank my committee members Dr. James Rowland and Dr. James Stiles for providing their valuable suggestions to construct this thesis. My sincere respect is shown towards all the faculty and staff members of the EECS department at the University of Kansas. All the coursework that I took in the EECS department has been extremely helpful for my research work. Overall, the two years of my Master’s study at the University of Kansas has been a great experience with regular ups and downs. But today I have become more matured both professionally and behaviorally with these experiences. I also want to thank my parents and family for constantly supporting me in this journey. It was pleasure to spent time with all my friends and colleagues at Lawrence. Without their support, staying away from home for such a long time would not have been possible. Finally, I want to thank all the students under Dr. Wang in Computer Vision group at the University of Kansas. I found great support working with them, learning new things and delivering at time of need.

Table of Contents

List of Figures

List of Tables

Chapter 1

Introduction

Automating visual detection and tracking of moving objects by intelligent independent systems has been an active research topic for the past decades in the computer vision field. The research has diverse applications extending from surveillance, military, security systems, auto-navigation, object recognition, to human-machine interactions [1]. In this research, we aim to develop an autonomous intelligent vision system that assists in autonomous navigation for unmanned aerial vehicles (UAV) with a forward-looking camera. The system can automatically localize and track the obstacles, which may be present in the path of the UAV. When the UAV is flying, it may collide with other UAVs, airplanes, birds, or some other flying objects. Therefore, it is essential to identify the obstacles and localize them in real-time throughout their trajectory for successful autonomous navigation and collision avoidance.

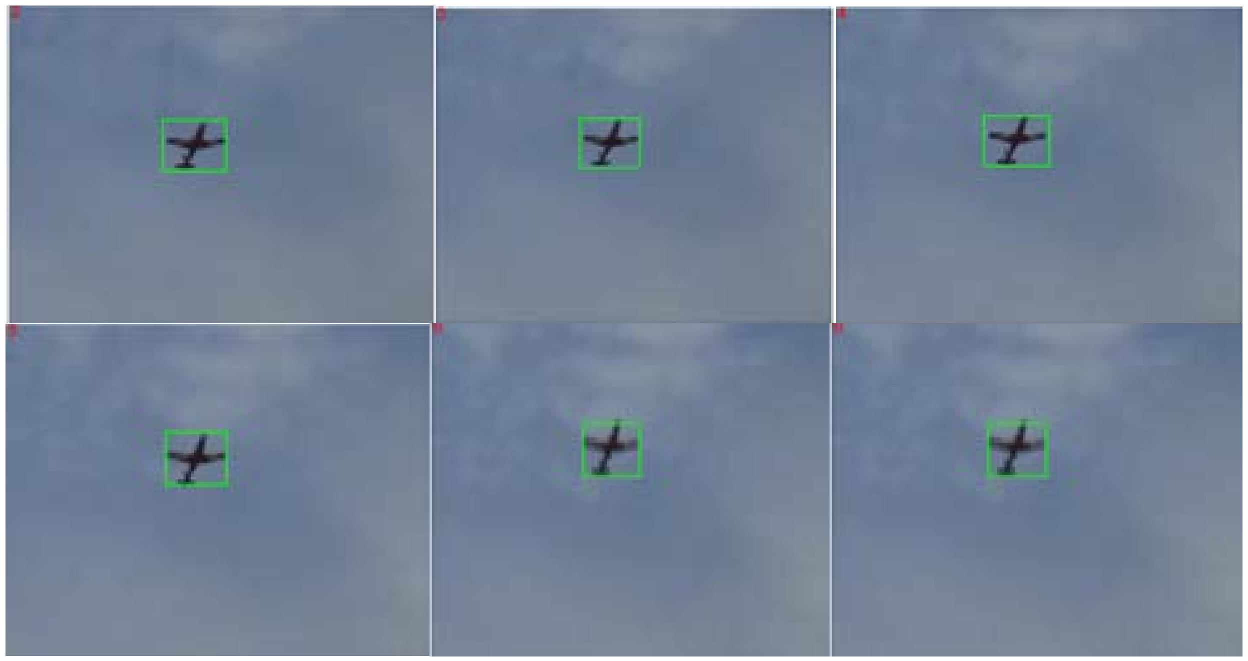


Fig 1.1: Object detection & tracking application of locating a flying object from *airplane_001* dataset [42]

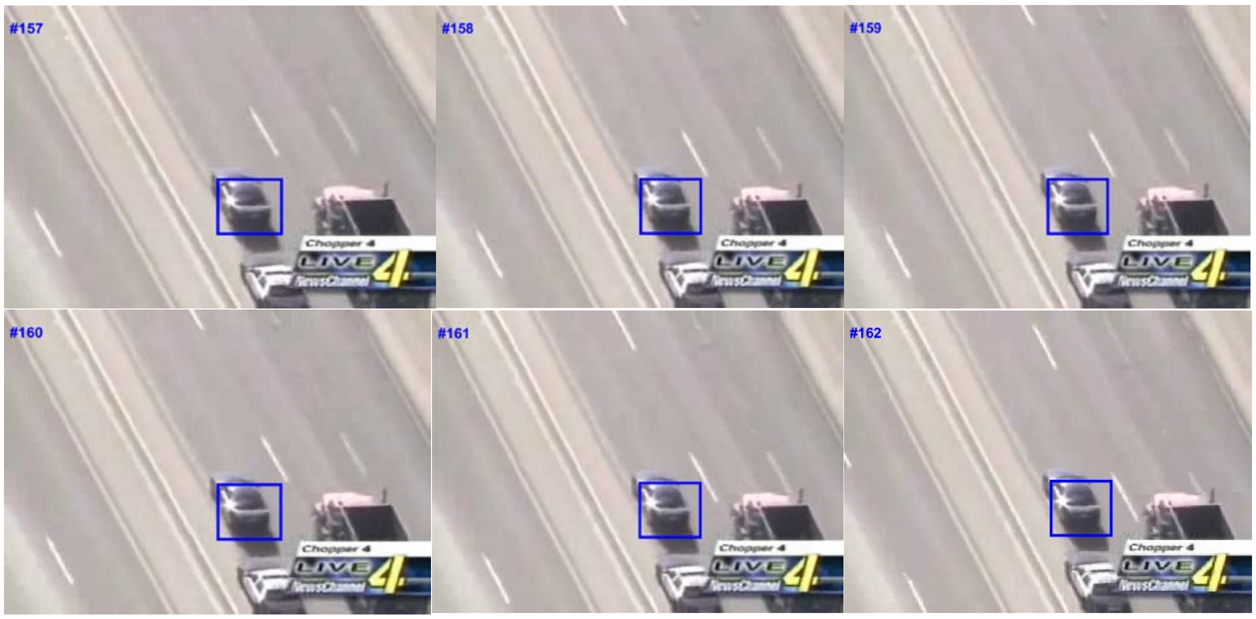


Fig 1.2 Tracking application of a car being chased from *chasing* dataset [30]

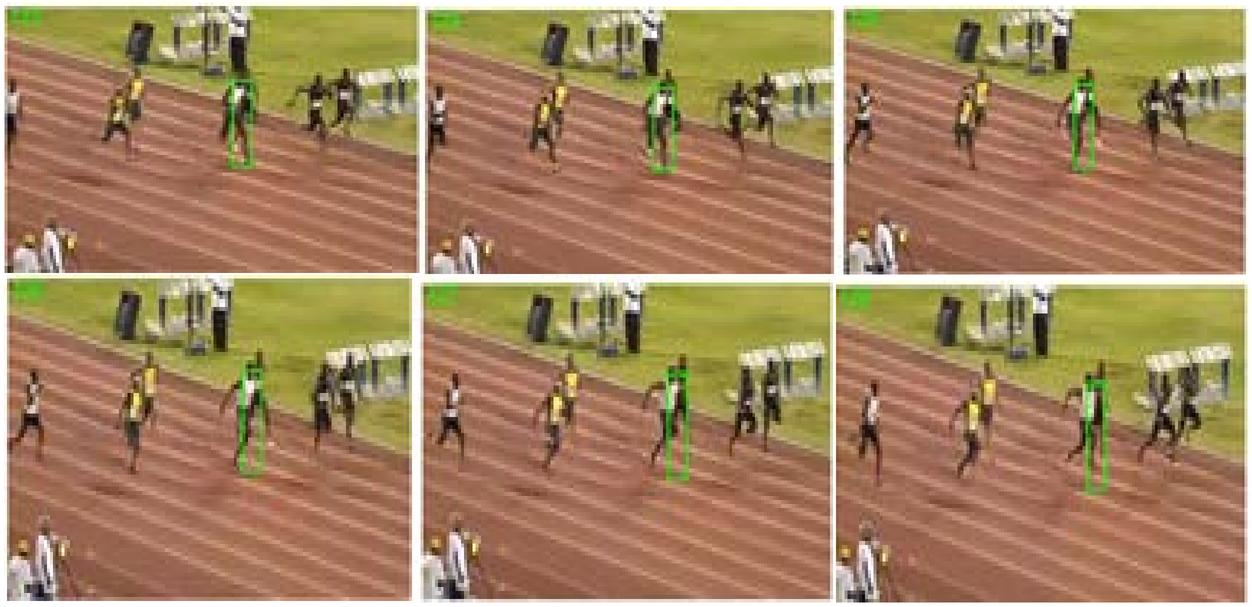


Fig 1.3 Tracking Bolt in Olympics track race from *bolt* dataset [30]

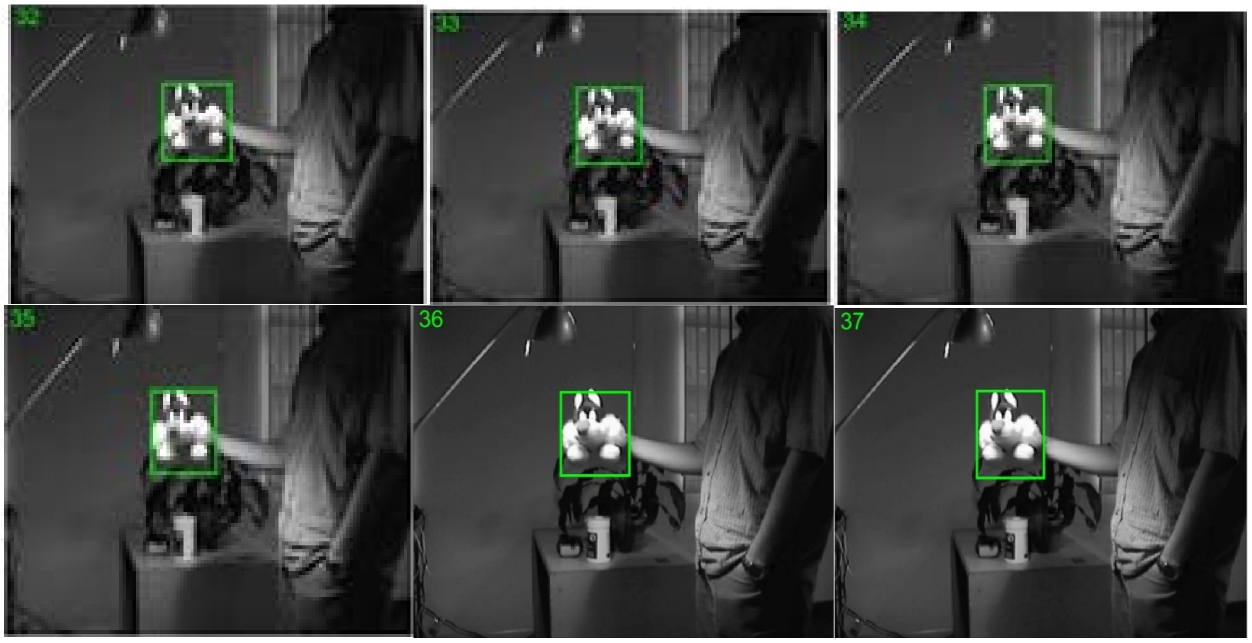


Fig 1.4 Tracking a doll called sylv when moved in illumination variation from *sylv* dataset [30]

Imagine looking at a scenery or a landscape and the first thing your eyes recognize is the characteristic features they could sense from the entire view [4]. These characteristic features, which help the human brain distinguish between a particular object and its background, are salient with resourceful information. An intelligent system which gathers the salient information in an image to segregate the object from its background is the basis of salient object detection.

On the other hand, given the initial detected position of an object in the initial frame, an intelligent system will localize the position of the moving object throughout the sequence. However, most of the previous works focused only on object detection or tracking, rather than creating an intelligent system capable of simultaneous detection and tracking in real-time. In this work, we propose a novel and intelligent vision system that can automatically detect, localize, and track the objects in high speed. Extensive experiments demonstrate that the proposed approach stands out among all the state-of-the-art detectors and trackers in terms of speed and precision.

1.1 Motivation

Unmanned aerial vehicles (UAV) came into existence for military purposes in situations, where humans flying the aircraft was very risky. In the twenty-first century, UAVs have become very common and is widely used for industrial, commercial or scientific purposes like aerial photography, agriculture, surveillance, product deliveries etc. To the best of our knowledge, till date no smart visual system has been developed yet, helping automated navigation of UAVs flawlessly.

UAVs can fly through minor difficulties with the well-established transponder-based collision avoidance systems in the absence of air-traffic or tall non-cooperative objects like tall buildings, towers, bridges, statues etc. But in presence of air-traffic with Aero planes, other flying objects like birds, other UAVs or tall structures, the navigation system of an UAV will require some visual assistance. Lack of awareness of the surrounding may lead to drastic accidents like airborne collisions with fixed or mobile objects, resulting in great loss of life and property.

Radars are an alternative of visual sensors for guiding the UAVs in their navigation. Radars are bulky with big transmitters and receivers, which are more suitable for installation in an Aeroplane. But an UAV is very small (maybe weighing 50-100 lbs.) and it has its own flying modules and processing systems installed on board. In that scenario, a small camera with visual aid and broader field of view can serve the purpose. Again if a Radar system is already present on the UAV, then the visual sensor can complement the Radar for guiding it in its auto navigation. In presence of noise, Radar data processing may lead to frequent miss detection or false alarms. Moreover, installing an image processing software on the UAV is much more cost-effective (and space effective) than installing an entire Radar processing unit. The navigation system needs to function at all time in all-weather conditions. Hence, object detection and tracking by a forward looking camera in an UAV will always boost the performance in presence or absence of a Radar system.

1.2 Goals

The object detection and tracking algorithms available in the science community generally focus on detecting objects on the ground with a down-looking camera. The goal of the proposed system is to detect both static and moving objects with a forward-looking camera in real-time, when the UAV is flying.

The biggest challenge of the tracking problem is to deal with the variation of illumination that may vary the intensity of two consecutive image frames, if looked from a slightly different angle due to the fast movement of the flying UAV. The fast motion often results in scale variation and rotation of the object. The online training to estimate the object position in the next frames becomes extremely difficult.

Again, the fast relative motion between the flying UAV and the object in front poses a serious challenge to distinguish correctly the actual reason for such variation in between image sequences. Extensive image registration with pre-processing and post-processing is required to deal with such problems.

Occlusion by clouds or tall buildings poses a threat to the success of the visual auto navigation system of the UAV. Once the object disappears from the image frame (due to getting hidden in between clouds or buildings), redetecting the object in the very next frame is very challenging.

Since the visual system is responsible for guiding the UAV in its navigation, it is very important for it to function at a very high speed – at least 5 frames per second. The online training for object estimation should not hold a lot of memory or processing unit.

1.3 Contribution

In this thesis, we integrate the techniques for salient object detection [8] and the kernelized correlation filter [3] together, and develop a long-term, error free object localization and tracking approach for autonomous navigation of flying UAVs. It generates better detection and tracking results compared to the state-of-the-art works in terms of both speed and accuracy as demonstrated in our experimentation chapter. It is evident that the proposed approach can successfully and accurately detect, as well as track, the salient object throughout the sequence, even when the appearance of the flying object suffers from deformations, scale variations, illumination variations, and in-plane/out-of-plane rotations. The main contributions of the proposed tracker are four-fold:

- The proposed approach is able to localize and generate an adaptive bounding box in real-time around the object being tracked, as the object changes its shape and size.
- Our approach, by conjoining tracking with detection and vice versa to form a closed loop system, makes it possible to handle long-term error free tracking.
- The proposed approach, by using an approximate label from the tracker in the previous frames, tracks the object in subsequent frames without requiring any computationally expensive supervised training.
- The proposed system runs automatically, without any manual initialization, and it achieves the best performance in terms of tracking accuracy and speed.

1.4 Outline

The rest of the thesis is organized in the following manner: In chapter 2, literature review of the most recent state-of-the-art tracking and detection algorithms has been presented. The research related to the distance transform based detection and correlation filter based tracking is portrayed (starting from their origin to the algorithms exhibiting best result).

In chapter 3, the distance transform based detection algorithm (used in the proposed model) is thoroughly studied. The relation between distance transform, graph theory and image processing is discussed along with their image segmentation application. Also, the advantage of using raster scanning based minimum barrier distance transform detection is fruitfully explained.

In chapter 4, the kernelized correlation filter based tracking algorithm (used in the proposed model) is thoroughly studied. The dense sampling method used to train a classifier (both linear and non-linear) is discussed in-depth. The reason for using correlation filter for tracking is examined along with the relevance of Circulant matrices, Least Squares, Kernels and image pre-processing.

Chapter 5 presents the proposed robust tracking with adaptive detection method. The proposed algorithm is tabulated, a flowchart is shown for better understanding and the detailed implementation of the algorithm with post-processing is discussed.

Chapter 6 presents the detailed experimentation, observations and results obtained with the proposed tracker, along with 6 other competing state-of-the-art trackers in 14 challenging datasets. Both quantitative and qualitative analysis is performed along with a detailed description of the comparison metrics.

In chapter 7, the advantages observed in the proposed method is briefly discussed. Some of the shortcomings observed while experimentation is mentioned. The future prospect of the proposed method is also demonstrated along with its numerous possible applications.

Appendix A presents the screenshots of all the tracked image frames in all of the 14 datasets with the proposed tracker.

Appendix B provides some of the MATLAB codes used to tabulate and plot the comparison metrics, discussed in chapter 6.

Chapter 2

Literature Review

Andriluka *et al.* [16] proposed an approach by combining a detector with the tracker. However, auto initialization is not possible in their method because the detector is needed to be trained with a huge number of data samples. Mahadevan *et al.* [17] proposed a tracker, combining a saliency map based detector for auto initialization in the first frame and hence tracking by exploiting the optical flow motion cues. This tracker is computationally very complex and could not be used in a real-time system. Nussberger *et al.* [18] used multiple cameras in an aircraft to measure the altitude and further used a tracker to sense and avoid collisions while flying. Their tracking method is found to be inefficient without the use of GPS data (which can have delays).

2.1 Object Detection

A saliency map used in salient object detection is a foreground mask, which can be visualized as a probability map of the salient pixels expressed in terms of intensity and relative to the entire image [5], [6]. Wei *et al.* [7] proposed connectivity prior and background prior and stated that the image background (generally, homogeneous) occupies a major portion of the image. As a result, this boundary can be connected very easily - connectivity prior. It is also assumed that objects are generally not present on the image boundaries, thus the image boundaries can mostly be assumed to be backgrounds - background prior. Zhang *et al.* [8] used minimum barrier distance [9], [10] to compute the saliency map using pixel connectivity (connectivity prior) with image boundary (background prior) via raster scanning the entire image.

2.1.1 Saliency Map based Object Detection

Previous research on saliency map-based object detection can be broadly classified into top-down and bottom-up methods. In top-down methods [5], [6], [19], all the possible objects in an image are localized and henceforth detection is executed on the reduced search space. But these methods are mostly task-driven and accompanied by supervised learning, making them un-realistic for real-time fast object detection. On the other hand, bottom-up methods [7], [8], [20]–[22] use the low level features (like the color, contrast, shape, texture, gradient and Spatio-temporal features) from an image to compare the feature contrast of the salient region with the background contrast.

The bottom-up salient object detection methods do not have prior-knowledge of the location of the object or the number of objects present in an image, and thus may fail in complex images. Whereas the top-down methods require training before performing detection. Our approach uses the approximate location of the object from the previous tracking results and then performs the re-detection on a much smaller search region. A similar approach was taken by Sui *et al*. [23]. Thus, our method does not require any supervised training, making it computationally efficient. Moreover, the reduced search region adds to the qualitative efficiency in performing the detection.

A geodesic saliency map was used in [7], [24] for object detection by utilizing the contrast information in the image and calculating the distance of each pixel from the background seeds to segment a region in the image. A supervised regression-based segmentation approach was developed in [21]. The binary classifier in this method was only adept at detecting single objects, but failed in the absence of any object or presence of multiple objects in an image. Some other salient object detectors [25], [26] create a ranked list of innumerable proposal windows in an image, instead of using a sliding window. This improves the recall rate, but fails to generate object localization to match the ground truth. Moreover, multiple windows were proposed for a single object leading to failure of the detector. A raster scanning method was used in [8] to generate a minimum barrier saliency map and it outperformed the geodesic saliency map. However, this method uses the entire image for creating the saliency maps, whereas our detector is adapted to look only in the most recent area where the object was located in the previous frame. Information about all the saliency-based detectors is beyond the scope of this thesis, but interested readers may refer to [27] for a comprehensive review.

2.2 Object Tracking

Classical tracking approaches are broadly classified as generative and discriminative models. In generative trackers [5], [6], [11], [12], the targets are represented as a set of basis vectors in a subspace and the trackers look for regions similar to the previously tracked targets. On the other hand, discriminative trackers [3], [13], [14] solve the problem as a binary classification in differentiating the tracked targets and the background. Ng *et al*. [15] has mathematically shown that discriminative asymptotic error is lower than generative asymptotic error, with large number of training samples.

2.2.1 Correlation Filter based Object Tracking

Correlation filters are adept at object localization, but previous work required supervised training, making it inappropriate for real-time online tracking. The minimum output sum of squared error (MOSSE) filter, proposed by Bolme *et al.* [28], efficiently trained the correlation filter on gray-scale images, making it computationally efficient and leading to real-time applications. Subsequently, an enormous amount of research has been dedicated to correlation filter-based tracking. Henriques *et al*. [29] improved the MOSSE filter with the CSK method to introduce a kernel-based correlation filter on grayscale images and thus achieved high speed in tracking with the benchmark datasets [30]. Similarly, Sui *et al*. improved on loss function to achieve consistent peak sensitivity of the tracking filter in [31].

Color features of the object were used along with a correlation filter in [32] for the first time. In this method, the correlation filter was trained by mapping multi-channel features onto a Gaussian kernel. Henriques *et al*. [3] improved the KCF tracker by integrating Gaussian and polynomial kernels along with multi-channel HOG features and achieved high speed and accuracy than most of the other state-of-the-art discriminative and generative trackers. But a major drawback of this method is that it is unable to deal with the scale variations of the target because of the fixed template size. Li *et al.* [2] handled the scale variations by accounting for the templates adaptive for tracking in each frame and also used HOG features for naming the colors in the images in SAMF tracker. Danelljan *et al.* [33] further integrated multi-scale correlation filter using HOG features in the DSST method to present another solution for adapting to the changing size and appearance of the object. But all these trackers are susceptible to occlusion and require a fixed scale and rotation of the object throughout the sequence, resulting in drifting in long-term tracking.

To deal with occlusion, Liu *et al.* [34] proposed a part-based tracking algorithm using a correlation filter. In the presence of occlusions, some part of the object may be visible and the part-based tracking exploits this feature to handle partial occlusion. However, this approach fails if the object becomes invisible between certain consecutive frames. Ma *et al.* [35] estimated the translation and scale change of the objects while tracking using the correlation between temporal contexts. This method also proposed a re-detection scheme by training a fern classifier to deal with tracking failures. However, this approach has taken down the speed of the tracker.

Some other detectors [20], [22], [36] and trackers [37], [38] use deep learning to generate more accurate results. But these methods need large-scale training datasets, which restricts their usage in real-time applications.

Chapter 3

MBD Detection

Detection of an object in a frame is analogous to the segmentation of an object from the background in a digital image. Segmentation is only possible after identification of the background and foreground pixels in an image with the basic assumption [7] that the appearance contrast between the foreground and background can be easily differentiated. Apart from this, in an image, the uniqueness, rarity, region uniformity and spatial compactness of the pixels can also help in identifying the background and the foreground pixels. All these characteristics of an image can be mathematically represented in terms of a distance function. Now, distance between any two pixels is needed to be computed with respect to the other pixels, forming a path. This path joins the two end pixels and a minimum cost function is associated, which directly depends on the intensity at each pixel. The variation of pixel intensity in an image helps in computing the distance function and thus also computes the minimum cost function for segmentation in an image.

3.1 Distance Transform

The distance transform [44] has been widely used for effectively analyzing the geometric and morphological characteristics of an object in an image [7, 8, 9, 10]. This distance transform can be represented in terms of a distance function, which is defined as follows by Rosenfeld *et al.* [44]:

Let **P** be the set of all integers (i,j). The function *f* from **P** x **P** into some non-negative integers is termed as:

a. *Positive definite,* if *f*(x,y) = 0, if and only if x = y.

b. *Symmetric,* if $f(x,y) = f(y,x)$, for all x,y in **P**.

c. *Triangular,* if $f(x,z) \leq f(x,y) + f(y,z)$, for all x, y, z in **P**.

If f satisfies all the three conditions (a-c), then f can be termed as a distance function. In some of the object detection and segmentation algorithms, the distance transform used is often termed as a *pseudo-metric,* which is mathematically identical to a distance function. Thus, a function f that satisfies conditions (a-c) is also known as a *pseudo-metric*. The function f is a metric function [10] if it satisfies all the above conditions (a-c) along with another condition:

d. *Positive,* $f(x,y) > 0$ for all $x \neq y$ and all x,y in **P**

Generally, distance function can be represented as and computed in two ways – either with Euclidean distance or with some variant of Dijkstra's algorithm. In the first approach, the Euclidean distance between a point and the background pixels is usually computed. In the latter approaches, discrete sets of image data with Dijkstra's algorithm or its variations are used for calculating the distance function. Image segmentation with the distance function of this type use seeds competition for all $x \in \mathbf{P}$. The two basic image segmentation methods [10] – Relative Fuzzy Connectedness (RFC) and Watershed (WS) belongs to this category. In these methods, image – its pixels, intensities and distance functions are represented mathematically in terms of graph theory.

An image intensity function $f : \mathbf{P} \rightarrow \mathbb{R}^l$, where **P** is its domain and its elements $\mathbf{x} \in \mathbf{P}$ are called space elements or '*spels*' [10] in short. Now, the image intensity at the pixel **x** is represented by the value of the function $f(x)$ for all $x \in \mathbf{P}$. The image intensity at *spel* **x** is an *l*-dimensional vector, representing attributes like color of an image. For segmentation, adjacent pixels in an image are taken into account and thus an *adjacency relation* is associated with the *spels* to generate the distance function by considering the *adjacent spel* pairs. The adjacency structure in an image domain **P** can be named as a '*scene*' [10]. In all the distance function evaluation, generally a rectangular *scene* is considered, such that $\mathbf{P} = \prod_{i=1}^{n}\{1,\ldots,a_n\}$, n = 2, 3, with 4-adjacency in 2D images.

	Top Adjacent Pixel	
Left Adjacent Pixel	**Chosen Pixel**	**Right Adjacent Pixel**
	Bottom Adjacent Pixel	

Fig 3.1: 4-adjacency in a 2D image. Yellow boxes represent adjacent pixels, while white boxes do not

3.1.1 Image Processing and Graph Theory Equivalence

In graph theory, vertex-edge pair represents a graph as G = {V,E}, where, **G** is a connected graph consisting of finite set of vertices **V** and finite set of edges **E**. Each edge **e** ∈ **E** connects any unordered pair {v1, v2} ∈ **V**. In image processing, for distance function $f : \mathbf{P} \rightarrow \mathbb{R}^{l}$, a *spel* defined previously is considered equivalent to the vertices of graph theory and an *adjacency relation* of an image *scene* represent the edges.

Again, a *path*, in a connected graph G = {V,E}, can be represented with a set of vertices π = $\langle \pi(0), \pi(1), \ldots \ldots, \pi(n) \rangle$, for all {π(i), π(i+1)} ∈ **V** and i ∈ {1,2,3,…n}. Now, a path connecting two vertices *a* and *b* can be mathematically written as π = $\langle \pi(0), \pi(1), \ldots \ldots, \pi(n) \rangle$, such that, π(0) = a and π(n) = b. There can be multiple paths joining the vertices *a* and *b* in a connected graph **G** and we denote them as $\prod_{\boldsymbol{a,b}}$.

But for image processing application like segmentation, we need only the minimum path connecting two pixels. For that purpose, we assign a cost function represented by a number λ(π) ≥ 0, which may be defined as the *length* of the path **π**. For example, let $f_{\boldsymbol{\lambda}}$: PxP⟶ [0, ∞] be a function used to calculate the minimum path between two pixels *a* and *b* in image **G**, then f_λ can be defined as:

$$f_\lambda = \min \{\lambda(\pi) : \pi \text{ is a path in G from } a \text{ to } b \}$$

This function $f_\lambda$is not a distance function until and unless it satisfies the following criteria: For every path $\pi = \langle\pi(0), \pi(1), \dots\dots., \pi(n)\rangle$,

1. $\lambda(\pi) = \lambda(\langle\pi(n), \pi(n-1), \dots\dots., \pi(0)\rangle)$
2. $\lambda(\pi) \leq \lambda(\langle\pi(0), \dots, \pi(i)\rangle) + \lambda(\langle\pi(i), \dots, \pi(n)\rangle)$ for all $0 \leq i \leq n$

When f_λ obeys 1 and 2, it is symmetric, positive definite and satisfies the triangular inequality, making it a pseudo-metric distance function, that can be used as a distance transform for image segmentation [9]. The weight function λ can either be a vertex weight or an edge weight and the saliency map generated from such weight functions are either vertex weight map or an edge weight map respectively. Geodesic distance map is an edge weight map, whereas minimum barrier transform map is a vertex weight map.

3.1.2 Geodesic Distance Transform

Geodesic distance transform is derived from the geodesic distance function. The cost function associated is defined as an edge weight map function $\lambda: E \rightarrow (0, \infty)$ for minimum distance from pixel *a* to *b* as λ(a,b). The path associated with λ is $\pi = \langle\pi(0), \pi(1), \dots\dots., \pi(n)\rangle$. The *geodesic distance function* (path length function) f_λ is defined as:

$$f_\lambda = \sum (\langle\pi(0), \pi(1), \dots\dots., \pi(n)\rangle) = \sum_{i=1}^{n} \lambda(\{\pi(i+1), \pi(i)\}) \qquad (3.1)$$

Wei *et al.* [7] used geodesic distance transform for object detection. In this method, image is represented as an undirected weighted graph G = {V,E}, with finite set of vertices **V** equivalent to image patches (pixels) and finite set of edges **E.** The vertices are either image patches $\{I_i\}$ or a virtual background node {B}. The edges are either internally connecting all I_i's or boundary edges connecting the background node {B} with the image boundary patches {I_i: I_i represents only image boundary}. The *geodesic saliency distance function* f_λ is computed in [7] as:

$$f_\lambda = \min_{I_1=I, I_2, \ldots, I_n = B} \sum_{i=1}^{n-1} weight(I_i, I_{i+1}) \tag{3.2}$$

such that, $(I_i, I_{i+1}, B) \in E$ and $weight$ (I_i, I_{i+1}) is the cost function λ(π) defined previously.

This distance function f_λ takes into account 3 priors:

1. Contrast prior - contrast variation between background and object
2. Backgroundness cue – An assumption inspired from photo cropping, considers that the boundary patches in an image generally do not contain the object pixel.
3. Connectivity cue – Another assumption from human vision perspective, considers all background patches are homogeneous and these pixels can be easily connected.

The Backgroundness cue assumption also leads to the failure of this method, if the image has some pixels representing object in the boundary of the map. Then the distance function calculated will be miserably wrong with bad detection results.

The connectivity cue assumption causes the *small-weight-accumulation* problem [7], where, the saliency map of the image gets whitened in the center because of high saliency value. This is due to the fact that, though human vision can consider the image of entire background as homogeneous, but in reality there is intensity variation of the background pixels and small saliency distance functions are generated in the background causing the *small-weight-accumulation* problem.

The cost function parameter *weight*(I_i, I_{i+1}) or λ(π) computes a distance function independent of image intensity when λ(π) is low. Again, when λ(π) tends to be high, the noise in object detection from the image becomes vibrant.

3.1.3 Minimum Barrier Distance Transform

Minimum barrier distance transform is derived from the minimum barrier distance function. The cost function associated is defined as an vertex weight map function $\lambda: E \rightarrow [0, \infty)$ for barrier distance from pixel *a* to *b* as $\lambda(a,b)$. The path associated with λ is $\pi = \langle \pi(0), \pi(1), \ldots\ldots, \pi(n) \rangle$. The *minimum barrier distance function* (path length function) $f_{\boldsymbol{\lambda}}$ is defined as:

$$f_{\boldsymbol{\lambda}} = \min_{\pi(i) \in I_{a,b}} (\lambda^{+}(\pi) - \lambda^{-}(\pi))$$
$$= \min_{\pi(i) \in I_{a,b}} (\max_{\boldsymbol{i=0,1,\ldots n}} \{\lambda(\pi(i))\} - \min_{\boldsymbol{i=0,1,\ldots n}} \{\lambda(\pi(i))\}) \qquad (3.3)$$

Here, $I_{a,b}$ represents all the possible paths from *a* to *b*. The minimum barrier distance map $f_{\boldsymbol{\lambda}}$ is a pseudo-metric [10] and the path length function is also known as *minimum barrier strength* [9]. The minimum barrier distance function $f_{\boldsymbol{\lambda}}$ is real valued and bounded, domain is a subset of Euclidean space $\mathbb{R}^{n}$, which in this image processing application is the 2D image pixel intensities. The advantage of minimum barrier distance transform over the other distance transforms is that the length of the path or distance function remains constant until and unless it finds a stronger barrier with intensity difference higher than the previously chosen seed. In other distance transforms, the length of path or distance function increases as the path connecting the seed and target grows.

The minimum barrier distance transform cannot be computed using the Dijkstra's algorithm like the other distance transforms and thus a new algorithm is proposed in [9]. The order of complexity to compute minimum barrier distance saliency map is O(*mn*log*n*), where *m* is the number of distinct intensities in all the pixels and *n* is the total number of pixels in the image. The optimized algorithm in [9] takes on average around half a second to compute the minimum barrier distance transform in a 300x200 image.

The biggest advantage of MBD transform is that it does not require any prior information. For example, other Euclidean based distance function require the approximate intensity distribution in the image to compute the distance map. Thus, no training is required for computing the MBD transform, making it suitable for real-time application.

The MBD transform is robust to noise, blur and smoothing, and provides better result than all other distance function as proved in [10]. Thus MBD transform becomes an obvious choice for image segmentation and object detection. Different algorithms are proposed to compute MBD transform – exact MBD [9], approximate MBD [10], raster scanned MBD [8]. The working speed of exact MBD transform is very low, whereas the performance of approximate MBD transform deteriorates with increase in noise or blur. The raster scanned MBD algorithm computes the MBD transform at an average speed of 80 frames per second as stated in [8], which is notably higher than the other versions and makes it suitable for object detection in a real time scenario like implementation of a sense and avoid system for an unmanned aerial vehicle.

3.1.4 Fast MBD Transform

The fast MBD transform is an iterative approximation of the exact MBD transform and has a linear complexity because the results can be obtained in very few iterations. But this approximation computes the distance function, comparable in quality with that of the exact MBD transform.

Fast MBD transform requires raster scanning and also inverse raster scanning of each pixel in the image to generate the distance function. We consider 4-*adjacency* in a 2D image. During raster scanning, the top and left adjacent pixels are considered, whereas the right and bottom adjacent pixels are considered in inverse raster scanning of the image.

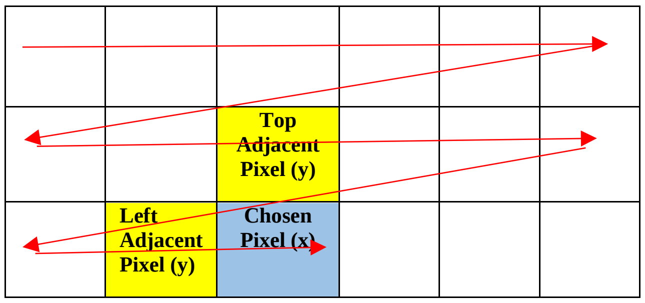


Fig 3.2: Raster Scanning [8]

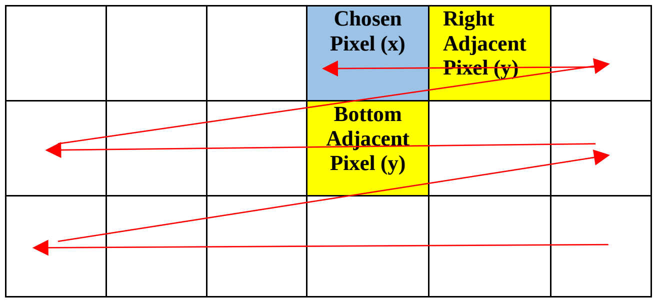


Fig 3.3: Inverse Raster Scanning [8]

The path of the raster and inverse raster scans consist of two parts –

1. Path from background seed to the adjacent pixel - π_y
2. Path from adjacent pixel to the chosen pixel – $\pi_{\{y,x\}}$

Let the entire path cost function be represented as $\lambda(\pi) = \lambda(\pi_y) + \lambda(\pi_{\{y,x\}})$, then the distance function for a single scan is given by:

$$f_\lambda = \max_{i=0,1,\ldots n} \{\lambda(\pi(i))\} - \min_{i=0,1,\ldots n} \{\lambda(\pi(i))\}$$

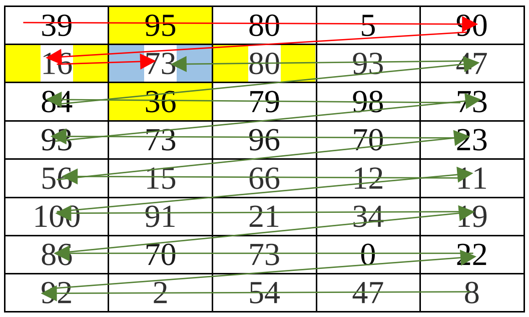
$$= \max\{A(y), I(x)\} - \min\{B(y), I(x)\} \qquad (3.4)$$

where, A(y) and B(y) are the highest and lowest pixel values on the entire path π from background seed to chosen pixel in a single scan via the adjacent pixel. Now, let f_I be the final MBD map.

f_I is modified in each scan by choosing the minimum between previous f_λ and current f_I as:

$$f_I = \min\ (\text{current } f_I, \text{previous } f_\lambda) \tag{3.5}$$

A(y) and B(y) are modified after each raster and inverse raster scan, as the path changes every time and this in turn modifies f_λ and f_I. There are *n* iterations and each iteration stop when f_λ becomes equal to f_I. So in each iteration, there is raster scan and inverse raster scan for each pixel of the image. After each iteration, the error between exact MBD and Fast MBD decreases. It has been shown in [8] that after 3 iterations (2 forward and 1 backward) generally, error becomes negligible.



39	95	80	5	90
16	73	80	93	47
84	36	79	98	73
93	73	96	70	23
56	15	66	12	11
100	91	21	34	19
86	70	73	0	22
92	2	54	47	8

Fig 3.4: Original Image pixel intensity distribution

0	0	0	0	0
0	∞	∞	∞	0
0	∞	∞	∞	0
0	∞	∞	∞	0
0	∞	∞	∞	0
0	∞	∞	∞	0
0	∞	∞	∞	0
0	0	0	0	0

Fig 3.5: Initial f_I, the MBD map

39	95	80	5	90
16	73	80	93	47
84	36	79	98	73
93	73	96	70	23
56	15	66	12	11
100	91	21	34	19
86	70	73	0	22
92	2	54	47	8

Fig 3.6: Initial Highest pixel intensity of the path A(y)

39	95	80	5	90
16	73	80	93	47
84	36	79	98	73
93	73	96	70	23
56	15	66	12	11
100	91	21	34	19
86	70	73	0	22
92	2	54	47	8

Fig 3.7: Initial Lowest pixel intensity of the path B(y)

For example, say fig 3.4 represent the pixel intensities of a 2D image. The algorithm is started by initiating f_I as:

$$f_I = \begin{cases} 0, & \text{for all the seed pixels} \\ \infty, & \text{for all other pixels in the image} \end{cases}$$

A(y) and B(y) are initialized as a matrix equal to the pixel intensities of the image.

Now suppose, first chosen pixel be {2,2}, that has an intensity of 73 in fig 3.4. So raster scan starts from {1,1} goes till {1,5} in first column, then comes to {2,1} and finally to {2,2} position. Similarly, inverse raster scan starts from {5,8} and the path is scanned in a reverse direction. In the process, A(y) and B(y) are modified resulting in computation of f_λ. This f_λ is assigned to f_I, after the first scan. But for the raster and inverse raster scan of the next pixel, f_λ is modified again. Now we compare the current f_λ and the previous f_I and choose the minimum of these two as the current f_I – which represents the present Fast MBD map. In this way, the same process is repeated for all the pixels in the image, completing the first iteration. The second iteration starts with the

MBD map already computed in the first iteration. As stated before, with 3 passes or iterations we get the desired Fast MBD map.

3.2 Salient Object Detection using Fast MBD Transform

The basic assumption considered for auto navigation of an unmanned aerial vehicle is a simple image setup – where the majority of the field of view should be homogeneous background and there should be no camouflage between the object and the background. So the main tasks needed to be executed for object detection are:

- The most salient object in the field of view must be detected first – the system should be smart enough to know which pixels in the image belongs to object and which ones belong to the background.
- Once the object is detected in an image, it should be separated from the background – either by segmentation or by generating a bounding box around the object. In this project, a bounding box is generated after salient object detection in each frame.

A good salient object detector must fulfill the following criteria for real time applications:

- The detector must be able to differentiate between all the background pixels and object pixels. All the object patches in an image should be detected and none of the patches of the background should be considered as object patch or vice versa.
- Object detection is only a small part of a bigger application. So the image segmentation process should be computationally efficient. Linear order of complexity is desired.
- The original image information should not be lost while computing the saliency map. The algorithm should never deteriorate the image resolution.

- The storage and memory requirement for the detection process should be low, making it feasible for a real time application.

Most of the salient object detection algorithms provide proposals about the location of object in the image – they do not provide the actual position in the image. The biggest advantage of fast MBD transform salient object detector is that it actually locates the object in the scene and never highlights just the probability of its appearance. Fast MBD detector also do not require any supervised training for generating the saliency map, making it useful for real-time applications like auto navigation of an unmanned aerial vehicle.

3.2.1 Saliency Map

Saliency of an image can be described as the unique property of the pixels in an image. There are certain unique properties that is exhibited by certain pixels in an image. Saliency map is an algorithm that looks for these pixels with their unique properties and mark them on the image to make it distinguishable by the computer [45]. Generally, the saliency of an image can be measured by the following unique properties of the pixels in an image:

- Edge cue - The density of the edges within a group of pixels is computed using Canny or Sobel edge detectors and compared with edge density at other pixels of the image to compute the saliency map.
- Color cue – The color distribution within a group of pixels is computed using histogram and statistical distributions (chi-squared generally) and compared with the color distribution at other pixels of the image to compute the saliency map.

- Superpixel cue – The color or texture features of similar pixels (cue) in an image are used to preserve the object boundaries. All the superpixel cues are compared to compute the saliency map.
- Spectral cue – The intensity of a group of pixels are computed in the Fourier domain and different spectral cues are obtained by computing the FFT of the group of pixels of different sizes. These different spectral cues are compared to generate the saliency map.

As mentioned in [45], the saliency map in an image can be computed in 3 ways generally:

- Saliency sum - The intensity difference between one pixel (say *x*) and all the other pixels in an image is calculated, followed by the sum of the absolute value of all the differences. This quantity determines the saliency of that particular pixel *x* and in the same manner the saliency of the other pixels is calculated to generate the final saliency map. Mathematically, if f_x be the intensity of pixel *x* and f_I be the intensity of any pixel *I* in the same image, the saliency map of the image is given by:

  Saliency map of image $\boldsymbol{P} = \sum_{I=1}^{n-1} | f_x - f_I |$, where, n is the total number of pixels in the image. If *m* number of pixels have same intensity, $\boldsymbol{P} = m * \sum_{I=1}^{n-m+1} | f_x - f_I |$ and order of complexity is O(n).
- Current and previous frame difference sum – The intensity difference between one pixel (say *x*) in the current frame and all the other pixels in the previous frame of a video sequence is calculated, followed by the sum of the absolute value of all the differences. This quantity determines the saliency of that particular pixel *x* in the current frame and in the same manner the saliency of the other pixels is calculated to generate the final saliency map for the current frame. Mathematically, if f_x be the intensity of pixel *x* in the current

frame and f_I be the intensity of any pixel I in the previous frame of the same video sequence, the saliency map of the current frame is given by:

Saliency map of the current frame $\boldsymbol{P}$ = $\sum_{I=1}^{n-1} | f_x - f_I |$, where, n is the total number of pixels in the image. If *m* number of pixels have same intensity, $\boldsymbol{P}$ = $m * \sum_{I=1}^{n-m+1} | f_x - f_I |$ and order of complexity is O(n).

- Coordinate saliency difference – The saliency maps of the current frame and the previous frame in a video sequence are calculated using the first way of saliency map computation and then the difference between the saliency map at each pixel of current frame and that of the previous frame is calculated to generate the final saliency map of the current frame. Mathematically, if f_x be the intensity of pixel *x* and f_I be the intensity of any pixel ***I*** in the current frame of a video sequence, the saliency map of the current frame is given by:

  $\boldsymbol{P}$ = $\sum_{I=1}^{n-1} | f_x - f_I |$ and let $\boldsymbol{Q}$ be the saliency map of the previous frame calculated in the same manner as ***P***, then the final saliency map of the current frame = $\boldsymbol{P - Q}$

3.2.2 Fast MBD Saliency Map

Object detection with generation of saliency map from the fast MBD transform described in section 3.1.4 is by far the best object detection algorithm (known to me) in terms of computational complexity and efficiency of the output. At first, the 3 color channels of an image are separated and each is converted to a grayscale version. For each of the 3 channels, the fast MBD transform of each pixel in the image is computed using equations (3.4) and (3.5) as mentioned in section 3.1.4. The background pixels are chosen as the seeds and assigned an intensity of zero to start with and subsequently the fast MBD transform is calculated for each pixel three times corresponding to each channel. After getting the saliency map from the 3 channels, they are combined together by pixel-wise adding the intensities received in the 3 saliency maps. The final saliency map is normalized to bring the intensities between zero and one. The fast MBD transform runs 3 iterations for generating each of the saliency maps as more iterations do not improve the detection results any further, but unnecessarily increases the computational complexity.

The following detection results are obtained by using the fast MBD salient object detection algorithm from [8] with a frame from *airplane_016* dataset in MATLAB:

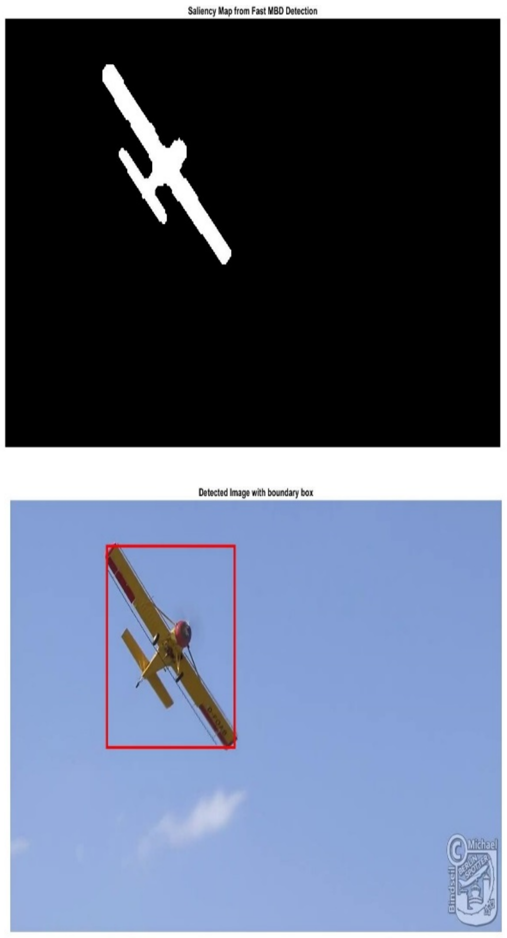

Saliency Map from Fast MBD Detection
Detected Image with boundary box

Fig 3.8 (a) Saliency Map generated by Fast MBD detector with *airplane_016* dataset [30] (b) Bounding box around original image frame from *airplane_016* dataset [30] after detection

Chapter 4

KCF Tracking

The negative samples in the classification task is as important as the positive samples and discriminative classifiers like KCF [3] take into account both positive and negative samples for classification. In the tracking problem, positive samples are collected from the foreground patches and negative samples from the background patches and a classifier uses these labels to train the system (basic idea behind tracking-by-detection methods). The complexity of an algorithm increases as the number of samples used for classification also increases. To cope with this problem, most of the online trackers try to avoid large number of negative samples or use similar negative samples (leading to redundancy), making the process inefficient. KCF gives the option of using innumerable samples (both positive and negative) by implementing classification in a correlation framework (Fourier domain) for both linear regression and non-linear regression (using kernels) models to train the tracker. This has increased both efficiency and accuracy of the tracking in real-time. The mathematical models from classical signal processing and ridge regression used by KCF are described in this chapter.

4.1 Dense Sampling

The core idea behind dense sampling is that all the samples are used for classification, instead of few samples chosen randomly. After choosing the base sample vectors, cyclic operations are performed to regenerate more samples – which can represent all the other possible samples. The squares of error of the correlation output between the generated samples and the test input (next frame) is minimized to train the model.

4.1.1 Regularized Least Squares

The training of a classifier to predict positive and negative labels in an image can be obtained by solving the Regularized Least Squares (RLS). RLS is a Ridge Regression problem, which can be solved by a single system of linear equations unlike the convex quadratic optimization required to solve a Support Vector Machine (SVM) as shown in [41]. Let **X** and **Y** be 2 sets of random variables such that there are *n* sets of statistically independent and identically distributed training samples **T** = $\{(x_1, y_1), (x_2, y_2), \ldots\ldots, (x_n, y_n)\}$, that follows the probability distribution of **X**x**Y**. The joint probability distribution of (x,y) is given by p(x,y) = p(y|x).p(x), where p(y|x) is the probability of occurrence of y given x.

We define a binary classifier **B(x)** to distinguish positive and negative labels from the training sample set **T**. Now, **B(x)** is trained with RLS as:

$$\text{RLS} = \min_{\mathbf{B}} \sum_{i=1}^{n} \boldsymbol{G}\,(y_i, \mathbf{B}(x_i)) \; + \lambda\, ||\mathbf{B}||^2 \tag{4.1}$$

where, $\boldsymbol{G}((y_i,\mathbf{B}(x_i))$ is a loss function, **B** is the regularization trade off and λ is a regularization parameter used to control the regularization and makes sure that there is no overfitting. The loss function chosen in RLS is $\boldsymbol{G}(y,\mathbf{B(x)}) = (y - \mathbf{B(x)})^2$. Now, diagonalization technique is used in [41] to express **B** in a closed form as:

$$\mathbf{B} = (\mathbf{X^T}X + \lambda\mathbf{I})^{-1}\, \mathbf{X^T}y \tag{4.2}$$

where, **I** is an identity matrix and $\mathbf{X}^T$ is the transpose of matrix **X**. In the complex plane, $\mathbf{X^T}$ becomes $\mathbf{X^H}$ and equation (4.2) can be written as:

$$\mathbf{B} = (\mathbf{X^H X} + \lambda\mathbf{I})^{-1}\, \mathbf{X^H}y \tag{4.3}$$

where, $\mathbf{X^H}$ is the Hermitian transpose of **X** i.e. $(\mathbf{X}^*)^{\mathbf{T}}$ where, $\mathbf{X}^*$ is the complex conjugate of **X**.

The RLS trains the classifier by computing **B** from the input samples **X**, regularization parameter λ and the output test data samples y.

4.1.2 Circulant Matrix

The base samples can be represented as a vector and shifted cyclically to form a matrix with all the shifted vectors. Henrique *et al.* [29] assumes that the dense samples (all the positive and negative samples) can be approximated from the positive samples along with a very small background (containing the negative samples), just by shifting it cyclically. This assumption stands true for most of the cases, making the tracking model superfast.

Let **x** = (x_1,x_2,……,x_n) be a vector consisting of *n* input samples. The Circulant matrix obtained from **x** is given by:

$$\mathbf{M(x)} = \begin{bmatrix} x_1 & x_2 & x_3 & \cdots & x_n \\ x_n & x_1 & x_2 & \ldots & x_{n-1} \\ x_{n-1} & x_n & x_1 & \ldots & x_{n-2} \\ \vdots & . & . & \ddots & \vdots \\ x_2 & x_3 & x_4 & \cdots & x_1 \end{bmatrix} \tag{4.4}$$

M(x) is the same matrix as **X** in equation (4.3). Each of the rows are created from the previous row by shifting one element at the end to the front. It can be generated very easily with the help of a permutation matrix **Δ** as:

$$\mathbf{\Delta} = \begin{bmatrix} 0 & 0 & 0 & \cdots & 1 \\ 1 & 0 & 0 & \ldots & 0 \\ 0 & 1 & 0 & \ldots & 0 \\ \vdots & . & . & \ddots & \vdots \\ 0 & 0 & 0 & \cdots 1 & 0 \end{bmatrix} \tag{4.5}$$

M(x) can be obtained from $\mathbf{\Delta.x^T}$ as **Δ** shifts each element of vector **x** by one space in the next row. In general, the *i*th row can be obtained from $\mathbf{\Delta^{\mathit{i}}.x^T}$. Also negative *i* will shift **x** in reverse direction. Thus **M(x)** has rows created from **x** by shifting half of it in the positive direction and the other half in the negative direction. Element of the Circulant matrix can be represented as:

$\mathbf{X}_{ij} = \mathbf{x}_{(j-i)\mathrm{mod}\,n}$, where *n* is the size of vector **x**, *i* and *j* are the row and column position in the matrix and *mod* computes the remainder when divided by *n*.

We train the classifier with input sample vector **x** and test label vector **y**. The fascinating thing about Circulant matrix is that **M(x).y** actually computes the convolution between vectors **x** and **y**. So now we can train the classifier in the Fourier domain and compute the convolution as:

$$\mathbf{M(x).y} = \boldsymbol{F}^{-1}\{\boldsymbol{F}^{*}(\mathbf{x}) \odot \boldsymbol{F}(\mathbf{y})\} \tag{4.6}$$

where, $\boldsymbol{F}$ and $\boldsymbol{F}^{-1}$ represent the Fourier transform and its inverse, * represents the complex conjugate and $\odot$ represents element-wise multiplication.

The intrinsic property of Circulant matrices is that their sums, products and inverses are also Circulant – leading to the fact that we do not have to store all of them in the memory and thus increases the computation speed of the tracker. The convolution between two vectors can be easily computed in Fourier domain with just element-wise multiplication and then taking an inverse as shown in equation (4.6) – resulting in orders of computational complexity reduction. The matrix inverse operation is computationally expensive and calculated efficiently in the Fourier domain using Eigen decomposition, that we will discuss in the next section.

4.1.3 Connection with RLS

The Circulant matrix **M(x)** or **X** can be evaluated from the training vector **x** in the Fourier domain using a Discrete Fourier Transform matrix **D** (constant and square) and the process is deterministic as all the elements of the training sample vector **x** has real values and not based on probability. Let $\underline{\mathbf{x}}$ be the DFT of vector **x**, then by Eigen decomposition of **X**, we can represent it as:

$$\mathbf{X} = \mathbf{D}\,\boldsymbol{\Lambda}\,\mathbf{D}^{\mathbf{H}} \tag{4.7}$$

where, $\mathbf{\Lambda}$ = *diagonal*($\underline{\mathbf{x}}$) is the diagonal matrix created from $\boldsymbol{\lambda}$ during Eigen decomposition as shown in equation (4.8):

$$\mathbf{X}\boldsymbol{\lambda} = p\boldsymbol{\lambda} \tag{4.8}$$

The matrix $\mathbf{\Lambda}$ consist of the elements of the Eigen vector $\boldsymbol{\lambda}$ in its diagonal and p is the Eigenvalue in the Eigenvalue decomposition of $\mathbf{X}$ in (4.8). This is sometimes known as diagonalization in classical signal processing. $\mathbf{D}$ is independent of $\mathbf{x}$ and becomes handy in computing DFT of any vector $\mathbf{d}$ as $\boldsymbol{F}(\mathbf{d}) = (\sqrt{n}).\ \mathbf{D.d}$, n being the size of vector $\mathbf{d}$ and $\boldsymbol{F}(\mathbf{d})$ is the Fourier transform of $\mathbf{d}$.

We can compute $\mathbf{X^H X}$ from equation (4.7) as:

$$\mathbf{X^H X} = (\mathbf{D\ \Lambda\ D^H})^{\mathbf{H}}\,(\mathbf{D\ \Lambda\ D^H}) = \mathbf{D\ \Lambda^*\ D^H\ D\ \Lambda\ D^H} \tag{4.9}$$

Now, $\mathbf{D^H D} = \mathbf{I}$, where $\mathbf{I}$ is an identity matrix, reducing equation (4.9) to:

$$\begin{aligned}\mathbf{X^H X} &= \mathbf{D\ \Lambda^*\ \Lambda\ D^H}\\ &= \mathbf{D}\ \{\mathit{diagonal}(\underline{\mathbf{x}})^*\}\{\mathit{diagonal}(\underline{\mathbf{x}})\}\ \mathbf{D^H}\\ &= \mathbf{D}\ \{\mathit{diagonal}(\underline{\mathbf{x}}^* \odot \underline{\mathbf{x}})\}\ \mathbf{D}\end{aligned} \tag{4.10}$$

where, $\mathbf{\Lambda^*\ \Lambda}$ has diagonal element-wise multiplication leading to equation (4.10). Again, $(\underline{\mathbf{x}}^* \odot \underline{\mathbf{x}})$ is actually the autocorrelation of a vector $\mathbf{x}$ (here image signal) in the Fourier domain, which is equivalent to power spectrum of the signal in classical signal processing.

The RLS binary classifier $\mathbf{B}$ in equation (4.3) can be represented in the frequency domain using the Eigen decomposed Circulant matrices in equation (4.10) as:

$$\begin{aligned}\underline{\mathbf{B}} &= [\mathit{diagonal}\{(\underline{\mathbf{x}}^*)\,/(\underline{\mathbf{x}}^* \odot \underline{\mathbf{x}} + \lambda)\}]\ \underline{\mathbf{y}}\\ &= \mathit{diagonal}\{(\underline{\mathbf{x}}^* \odot \underline{\mathbf{y}})\,/(\underline{\mathbf{x}}^* \odot \underline{\mathbf{x}} + \lambda)\}\end{aligned} \tag{4.11}$$

where, $\underline{\mathbf{B}}$ is the binary classifier in frequency domain, $\underline{\mathbf{x}}$ is the training vector in frequency domain, $\underline{\mathbf{y}}$ is the testing vector sample in the frequency domain and λ is the regularization parameter defined in equation (4.3). Division is performed element-wise iteratively to obtain $\underline{\mathbf{B}}$.

This $\underline{\mathbf{B}}$ is the regularized correlation filter in the frequency domain defined in [28]. The correlation filter **B** in the spatial domain can be easily obtained using an inverse Fourier transform. The computational complexity of taking DFT is linear $O(n\log n)$ and all the divisions and multiplications are element-wise with a complexity of $O(n)$. This is a great advantage over the standard method of computing RLS with a complexity of $O(n^3)$ as it involves matrix product and inversion.

4.2 Non-Linear Regression

The objective of non-linear regression is to compute a regression function for classification, unlike the binary classification (where only 2 outputs are expected). This regression function is represented as a kernel or non-linear classifier. Generally, in tracking-by-detection methods, some classifiers use the power spectrum of the samples. It is difficult to kernelize the spatial model based on non-linear classifiers that use the power spectrum of the samples, as the regression function grows in complexity with the increase of the number of samples. The kernel matrix representation in [40] and the use of regularized least squares with kernels in [29] gave a solution for using non-linear regression models in the tracking problem without affecting the complexity, even though the number of samples increase. This is done by using the kernel matrix.

4.2.1 Kernel Matrix

The *Kernel trick* in [40] maps the input **x** to a higher dimensional non-linear feature space **H(x)**. Measuring the similarity between input samples is not as easy as finding the *similarity* *measure* between output labels. A kernel k (function) is defined in [40] to measure the similarity between inputs x_1 and x_2 with a mapping in the real plane as:

$$k\colon \mathbf{X} \text{ x } \mathbf{X} \to \mathbb{R}$$

$$(x_1,x_2) \to k(x_1,x_2) \qquad (4.12)$$

The simplest form of kernel function is an *inner product* operation defined between inputs x_1 and x_2 as:

$$\langle x_1, x_2\rangle = \sum_{i,j=1}^{n}[x_i]\ [x_j], \qquad (4.13)$$

where, i,j are the i-th and j-th elements in the input vector $\mathbf{x}$. Now, if the input samples are normalized, then in the feature space $\mathbf{H(x)}$, the inner product is equivalent to the cosine of the angles between the vectors x_1 and x_2. Thus, the inner product can be computed in terms of lengths and angles in the mapped vector feature space. Equation (4.12) changes to the following in the mapped feature space, allowing geometric and linear algebraic operations for training the classifiers:

$$k(x_1,x_2) := \langle x_1, x_2\rangle = \langle \mathbf{H}(x_1), \mathbf{H}(x_2)\rangle \qquad (4.14)$$

A *distance function* $\boldsymbol{f}(\mathbf{y})$ segregates labels $\mathbf{y}$ in the feature space. The distance between a distance function and the testing sample($\boldsymbol{y}$) determines the classification (that is to be assigned) corresponding to the testing sample. Now, if this distance is equal in magnitude for two different samples, then it becomes difficult to choose the better kernel. So a weight α_i is assigned to the i-th kernel that reflects the importance of the i-th kernel function for classifying a particular test sample. In *constrained optimization* problem, this weight α_i is made equivalent to a *Lagrange's multiplier* $\alpha_i>0$ and the distance function can be derived from a *Lagrangian* like:

$$\boldsymbol{L}(\mathbf{B},\text{c},\alpha) = \frac{1}{2}\,||\mathbf{B}||^2 - \sum_{i=1}^{n}\alpha_i(y_i(\langle x_i, \mathbf{B}\rangle\ + \text{c}) - 1) \qquad (4.15)$$

where, **B** is the RLS binary classifier (defined in section 4.1.1) and is also a primal variable in the Lagrangian. **c** is another primal variable constant in the set of linear equations. Now for getting an optimized kernel distance function, the Lagrangian in (4.15) needs to be maximized with respect to the dual variable α and minimized with respect to primal variables **B** and **c**. The *dual optimization problem* obtained by differentiating equation (4.15) with respect to **B**, c and α individually leads to a RLS binary classifier in terms of dual variable α (free of primal variables **B** and **c**) and the final result is expressed in the higher dimension feature space **H(x)** as:

$$\mathbf{B} = \sum_i \alpha_i \boldsymbol{H}(x_i) \tag{4.16}$$

where, x_i is the *i*-th testing sample to be labeled.

The distance function $\boldsymbol{f}(\boldsymbol{y})$, when represented in terms of inner product kernel function k in the dual space with respect to RLS binary classifier **B** looks like:

$$\boldsymbol{f}(\boldsymbol{y}) = \mathbf{B}^{\mathbf{T}} \boldsymbol{y} = \sum_{i=1}^{n} \alpha_i \, k(\boldsymbol{y}, x_i) \tag{4.17}$$

This distance function $\boldsymbol{f}(\boldsymbol{y})$ is the new kernelized classifier. A kernel matrix **K** (nxn) stores all the inner products between all the sample pairs (mentioned in [3]) and its (*i*,*j*)-th element is expressed as:

$$\mathbf{K}_{ij} = k(x_i, x_j) \tag{4.18}$$

In non-linear regression, the complexity of $\boldsymbol{f}(\boldsymbol{y})$ increases with increase in number of samples, which can be tackled using the *kernel trick* and the *Circulant matrices* as discussed in the next section.

4.2.2 Kernel Regularized Least Squares

The relation between the RLS classifier and the Kernel matrix is explained in the previous section. We can use the kernel matrix **K**, testing sample **y** and the regularization parameter λ to evaluate the dual space variable or the *kernel weight* **α** from equations (4.3), (4.17) and (4.18) as:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{4.19}$$

where, **I** is an identity matrix of same dimension as the kernel matrix **K**.

For relaxing the complexity, the kernel matrix would be preferred to be Circulant in case of non linear regression, so that we can evaluate it easily in the Fourier domain as we did for the linear RLS classifier. Henrique *et al*. [3] states that a kernel matrix is Circulant if the kernel function satisfies:

$$k(x_1,x_2) = k(\boldsymbol{\Delta}\, x_1, \boldsymbol{\Delta}\, x_2) \tag{4.20}$$

where, **Δ** is the Permutation matrix defined in section 4.1.2 to evaluate Circulant matrix **M(x)** for input sample vector **x**. The data at each position of the kernel matrix should have equal importance for a matrix to satisfy (4.20) and become Circulant.

The kernel matrix in (4.19) is evaluated in the Fourier domain using a correlation operation and is known as the *kernel correlation*. The kernel correlation between two vectors $\mathbf{x_1}$ and $\mathbf{x_2}$ (of same length) is represented as:

$$k_i^{\mathbf{x_1 x_2}} = k(\mathbf{x_2}, \boldsymbol{\Delta}^{i-1}\, \mathbf{x_1}) \tag{4.21}$$

where, k is evaluating the inner products between the sample pairs $\mathbf{x_2}$ and $\boldsymbol{\Delta}^{i-1}\mathbf{x_1}$ as shown in equation (4.18) and stores it in the vector $k_i^{\mathbf{x_1 x_2}}$. **Δ** is the permutation matrix and i is the i-th position in the vector $k_i^{\mathbf{x_1 x_2}}$.

In the higher dimensional feature space **H(x)**, kernel correlation is computed using inner product as:

$$k_i^{\mathbf{x_1 x_2}} = \mathbf{H}^{\mathbf{T}}(x_2)\, \mathbf{H}(\mathbf{\Delta}^{i\text{-}1} x_1) \tag{4.22}$$

Similarly, autocorrelation of the input sample vector **x** can be evaluated and stored in a kernel vector like (4.21) as:

$$k_i^{\mathbf{xx}} = k(\mathbf{x}, \mathbf{\Delta}^{i\text{-}1}\, \mathbf{x}) \tag{4.23}$$

Now in the Fourier domain, the kernel autocorrelation vector can be represented as $\underline{k}^{\mathbf{xx}}$. Thus, the kernel weight vector or the dual space variable $\boldsymbol{\alpha}$ can be evaluated from input sample kernel autocorrelation by simplifying equation (4.19) and taking an inverse Fourier transform as:

$$\boldsymbol{\alpha} = \mathbf{F^{-1}}\, (\underline{\mathbf{y}}\, (\underline{k}^{\mathbf{xx}} + \lambda)^{-1}) \tag{4.24}$$

Equation (4.24) represents KRLS weight vector used to train the classifier, where divisions are done element-wise. Conventional kernels require *n*x*n* matrix computations [$O(n^4)$ as require sliding window over entire image], when compared to only *n*x1 vector computations in Fourier domain, resulting in closed form solution and complexity relaxation to $O(n^2 \log n)$.

4.2.3 KRLS Filter

The KRLS weight vector defined in the previous section (equation 4.24) can be used to train the classifier. Once the classifier is trained, we want to detect the image pixels as foreground or background. Now a single testing image pixel can be represented as *y*. But we are interested in testing multiple pixels in an image and consider that all such pixels (*y*'s) are combined in a single test vector **y**. The kernelized classifier ***f***(***y***) defined in equation (4.17) classifies all the elements in testing vector **y** to determine them as foreground or background.

This is where we again exploit the benefit of Circulant matrix. Let the RLS binary classifier **B** in equation (4.16) be comprised of the kernel matrix **K** defined in equation (4.18) and let us represent the new *classifier kernel matrix* as $\mathbf{B^y}$. The cyclic shifts between base input training sample vector **x** and the base output testing sample vector **y** generates all the possible training and testing sample vectors - used to develop $\mathbf{B}^{\mathbf{y}}$. The (*i,j*)-th element of the matrix $\mathbf{B}^{\mathbf{y}}$ is computed like equation (4.21) as:

$$k_{i,j}^{\mathbf{xy}} = k(\boldsymbol{\Delta}^{j-1}\mathbf{y}, \boldsymbol{\Delta}^{i-1}\mathbf{x}) \tag{4.25}$$

where, $k_{i,j}^{\mathbf{xy}}$ is the kernel correlation between the base input training sample vector **x** and the base output testing sample vector **y**. **Δ** is the permutation matrix defined in section 4.1.2. Equation (4.21) is different from equation (4.25) with a $\boldsymbol{\Delta}^{j-1}$-term as vectors **x** and **y** may have different dimensions.

$k^{\mathbf{xy}}$ is the *kernel correlation vector* comprising of all the elements $k_{i,j}^{\mathbf{xy}}$. The Circulant matrix $\mathbf{M}(k^{\mathbf{xy}})$ (defined in section 4.1.2) of $k^{\mathbf{xy}}$ is used to evaluate the classifier kernel matrix $\mathbf{B}^{\mathbf{y}}$ as:

$$\mathbf{B}^{\mathbf{y}} = \mathbf{M}(k^{\mathbf{xy}}) \tag{4.26}$$

The kernel classifier $\boldsymbol{f}(\boldsymbol{y})$ of equation (4.17) can be termed as the *KRLS classifier* and defined as:

$$\boldsymbol{f}(\boldsymbol{y}) = (\mathbf{B}^{\mathbf{y}})^{\mathbf{T}}\, \boldsymbol{\alpha} \tag{4.27}$$

Using the cyclic shift knowledge of equation (4.26), we can evaluate $\boldsymbol{f}(\boldsymbol{y})$ vector in the Fourier domain and get it back in the spatial domain with an inverse Fourier transform as:

$$\boldsymbol{f}(\boldsymbol{y}) = \mathbf{F}^{-1}(\, \underline{k}^{\mathbf{xy}} \odot \underline{\boldsymbol{\alpha}}\,) \tag{4.28}$$

where, $\mathbf{F}^{-1}$ is the inverse Fourier transform, $\underline{k}^{\mathbf{xy}}$ and $\underline{\boldsymbol{\alpha}}$ are the Fourier transforms of $k^{\mathbf{xy}}$ and $\boldsymbol{\alpha}$ and $\odot$ evaluates the convolution between $\underline{k}^{\mathbf{xy}}$ and $\underline{\boldsymbol{\alpha}}$ in the Fourier domain. $\boldsymbol{f}(\boldsymbol{y})$ vector stores all the classified (detected) image patches from the base test sample vector **y**. $\boldsymbol{f}(\boldsymbol{y})$ is obtained by filtering operations on kernels in the Fourier domain to enhance the speed of the tracker and hence is also known as the *KRLS filter*.

4.3 Kernel Correlation Association

Computation of kernels for non-linear regression is also very rigorous and complex as the kernel correlation ($k^{\mathbf{x}\mathbf{y}}$) computation requires evaluation of 2 vectors (training vector **x** and test vector **y**) for all their relative shifts (to generate the Circulant matrices making training and detection easier for the classifier). This computational burden can be removed if we can express the kernels also as Circulant matrices. Any kernel that follows equation (4.20) could be represented as a Circulant matrix. The inner product or radial basis function kernels follow equation (4.20) and has been briefly described in the following subsections.

4.3.1 Inner Product Kernel

Let $k(\mathbf{x}_1, \mathbf{x}_2)$ represents a kernel function (similarity measure) between two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ defined in section 4.2.1. Then, vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ form an *inner product* kernel function $IP(\mathbf{x}_1^{\mathbf{T}}\mathbf{x}_2) = k(\mathbf{x}_1, \mathbf{x}_2)$, for some function $IP(\mathrm{x})$, such that it follows equation (4.21) as:

$$k_i^{\mathbf{x}_1\mathbf{x}_2} = k(\mathbf{x}_2, \mathbf{\Delta}^{i-1}\,\mathbf{x}_1) = IP(\mathbf{x}_2^{\mathbf{T}}\,\mathbf{\Delta}^{i-1}\mathbf{x}_1) \tag{4.29}$$

where, $k_i^{\mathbf{x}_1\mathbf{x}_2}$ is the kernel correlation and represent the *i*–th element of the inner product kernel k (here $IP(\mathrm{x})$), $\mathbf{\Delta}$ is the permutation matrix defined in section 4.1.2.

Now, considering the inner product kernel function $IP(\mathrm{x})$ to operate element-wise on the input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, we can express the kernel correlation vector $k^{\mathbf{x}_1\mathbf{x}_2}$ from equation (4.26) as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = IP(\mathbf{M}(\mathbf{x}_1)\,\mathbf{x}_2) \tag{4.30}$$

where, $\mathbf{M}(\mathbf{x}_1)$ represents the circular matrix of vector $\mathbf{x}_1$ defined in section 4.1.2.

Thus, we can use the kernel trick of section 4.2.2 and evaluate $k^{\mathbf{x}_1\mathbf{x}_2}$ in the Fourier domain as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = IP\,(\mathbf{F}^{-1}\,(\mathbf{M}(\mathbf{x}_1)\,\mathbf{x}_2)\;) = IP\,(\mathbf{F}^{-1}\;(\underline{\mathbf{x}}_1^{*} \odot \underline{\mathbf{x}}_2)) \tag{4.31}$$

where, $\mathbf{F}^{-1}$ represents the Inverse Fourier Transform, $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ represent vectors $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ in the Fourier domain and ($\underline{\mathbf{x}}_1^{*} \odot \underline{\mathbf{x}}_2$) is the autocorrelation between vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Equation (4.31) can be derived using the diagonalization technique discussed in equation (4.6) in section 4.1.3.

Most commonly used inner product kernel is the *polynomial kernel* $k(\mathbf{x}_1, \mathbf{x}_2) = (\mathbf{x}_1^{\mathbf{T}}\mathbf{x}_2 + \mathrm{c})^{\mathrm{m}}$, where c and m are constants, such that the kernel vector $k^{\mathbf{x}_1\mathbf{x}_2}$ is evaluated in the Fourier domain as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = (\mathbf{F}^{-1}\,(\underline{\mathbf{x}}_1^{*} \odot \underline{\mathbf{x}}_2)\; + \;\mathrm{c})^{\mathrm{m}} \tag{4.32}$$

The complexity for evaluating equation (4.32) is $O(n\log n)$.

4.3.2 Radial Basis Function Kernel

Let $k(\mathbf{x}_1, \mathbf{x}_2)$ represents a kernel function (similarity measure) between two vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ defined in section 4.2.1. Then, vectors $\mathbf{x}_1$ and $\mathbf{x}_2$ form an *radial basis* kernel function $RB(||\mathbf{x}_1 - \mathbf{x}_2||)^2 = k(\mathbf{x}_1, \mathbf{x}_2)$, for some function $RB(\mathrm{x})$, such that it follows equation (4.21) as:

$$k_i^{\mathbf{x}_1\mathbf{x}_2} = k(\mathbf{x}_2, \mathbf{\Delta}^{i-1}\,\mathbf{x}_1) = RB(||\mathbf{x}_2 - \mathbf{\Delta}^{i-1}\mathbf{x}_1||^2) = RB(||\mathbf{x}_1||^2 + ||\mathbf{x}_2||^2 - 2\,\mathbf{x}_2^{\mathbf{T}}\mathbf{\Delta}^{i-1}\mathbf{x}_1) \tag{4.33}$$

where, $k_i^{\mathbf{x}_1\mathbf{x}_2}$ is the kernel correlation and represent the i–th element of the radial basis kernel k (here $RB(\mathrm{x})$), $\mathbf{\Delta}$ is the permutation matrix defined in section 4.1.2, $||\mathbf{x}||^2$ is the l-2 norm of $\mathbf{x}$.

Now, considering the radial basis kernel function $RB(\mathrm{x})$ to operate element-wise on the input vectors $\mathbf{x}_1$ and $\mathbf{x}_2$, we can express the kernel correlation vector $k^{\mathbf{x}_1\mathbf{x}_2}$ from equation (4.26) as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = RB(\mathbf{M}(\mathbf{x}_1)\,\mathbf{x}_2) \tag{4.34}$$

where, $\mathbf{M}(\mathbf{x}_1)$ represents the circular matrix of vector $\mathbf{x}_1$ defined in section 4.1.2.

Thus, we can use the kernel trick of section 4.2.2 and evaluate $k^{\mathbf{x}_1\mathbf{x}_2}$ in the Fourier domain as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = RB\,(\,\mathbf{F}^{-1}\,(\mathbf{M(x}_1\mathbf{)\;x}_2)\;) = RB\,(\|\mathbf{x}_1\|^2 + \|\mathbf{x}_2\|^2 - 2\mathbf{F}^{-1}\;(\underline{\mathbf{x}}_1^* \odot \underline{\mathbf{x}}_2)) \tag{4.35}$$

where, $\mathbf{F}^{-1}$ represents the Inverse Fourier Transform, $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ represent vectors $\underline{\mathbf{x}}_1$ and $\underline{\mathbf{x}}_2$ in the Fourier domain and $(\underline{\mathbf{x}}_1^* \odot \underline{\mathbf{x}}_2)$ is the autocorrelation between vectors $\mathbf{x}_1$ and $\mathbf{x}_2$. Equation (4.35) can be derived using the diagonalization technique discussed in equation (4.6) in section 4.1.3.

Most commonly used radial basis kernel is the *Gaussian kernel* $k(\mathbf{x}_1, \mathbf{x}_2) = \exp(-\frac{1}{\sigma^2}\|\mathbf{x}_1 - \mathbf{x}_2\|^2)$, where σ^2 is the constant variance, such that the kernel correlation vector $k^{\mathbf{x}_1\mathbf{x}_2}$ is evaluated in the Fourier domain as:

$$k^{\mathbf{x}_1\mathbf{x}_2} = \exp(-\frac{1}{\sigma^2}(\|\mathbf{x}_1\|^2 + \|\mathbf{x}_2\|^2 - 2\mathbf{F}^{-1}\,(\underline{\mathbf{x}}_1^* \odot \underline{\mathbf{x}}_2))) \tag{4.36}$$

The complexity for evaluating equation (4.36) is $O(n\log n)$.

4.4 Raw Pixel Preprocessing

Correlation filtering involves element-wise multiplication between the image and the correlation filter in the Fourier domain. The left edge of the image touches the right edge and the top edge touches the bottom edge while convolving with a filter in the Fourier domain. Images are non-periodic with discontinuity between the opposite edges. On the other hand, Fourier transform is periodic. The image boundaries touching each other affect the correlation output (generates noise in the Fourier domain) and subsequently tracking performance deteriorates. To tackle with such a problem, preprocessing the images with cosine window was suggested in [28].

Let x_{ij} represent the (i,j)-th raw pixel intensity in an image matrix, then the pixel intensity after cosine window preprocessing can be expressed as:

$$x_{ij} = (x_{ij}^{raw} - 0.5)\sin(\pi i/n)\sin(\pi j/n), \qquad \forall (i,j) = 0,1,2,\ldots\ldots,n\text{-}1 \qquad (4.37)$$

Cosine windowing assign weights to transform the pixel intensities on the boundaries to zero and the center pixels to highest magnitudes. Thus approximating the opposite edges of an image, such that the image sequence may appear periodic and produce less noise in the Fourier domain.

4.5 Multiple Channel Inputs

The Fourier domain presents a great advantage for multiple channel signals (here image) as the kernel functions (inner product and radial basis kernels considered) from each channel can be evaluated in the Fourier domain and simply added together to get the multiple channel kernel function.

Let $\mathbf{x} = [\mathbf{x}^1,\mathbf{x}^2,\ldots\ldots,\mathbf{x}^n]$ be input vectors from n channels. Considering the linear property of Fourier transform, a linear kernel $k_n^{\mathbf{x_1x_2}}$ from n channels in the dual space can be expressed similar to equation (4.6) as:

$$k_n^{\mathbf{x_1x_2}} = \mathbf{F}^{-1}\left(\textstyle\sum_{i=1}^{n} \left(\underline{\mathbf{x}}_1^{i*} \odot \underline{\mathbf{x}}_2^{\,i}\right)\right) \qquad (4.38)$$

Equation (4.38) represents Dual Correlation Filter (DCF) mentioned in [3]. If input vector $\mathbf{x}$ (here $\mathbf{x_1}$) consist of a single base sample, $k_n^{\mathbf{x_1x_2}}$ can be evaluated from n multiple channels, maintaining the diagonalization and Eigen decomposition described earlier. In case of multiple base samples constituting the input vector $\mathbf{x}$, the kernel function $k^{\mathbf{x_1x_2}}$ can only be computed for a single channel to maintain the Eigen decomposition in computing the kernel, making it equivalent to the MOSSE filter in [28].

Let $\mathbf{x} = [\mathbf{x}^1,\mathbf{x}^2,\ldots\ldots,\mathbf{x}^n]$ be input vectors from n channels. Considering the linear property of Fourier transform, a Gaussian kernel $k_n^{\mathbf{x_1x_2}}$ from n channels can be expressed similar to equation (4.36) as:

$$k_n^{\mathbf{x_1 x_2}} = \exp\left(- \frac{1}{\sigma^2} \left(\|\mathbf{x_1}\|^2 + \|\mathbf{x_2}\|^2 - 2\, \mathbf{F^{-1}} \left(\sum_{i=1}^{n} \left(\underline{\mathbf{x}}_{\mathbf{1}}^{\boldsymbol{i}*} \odot \underline{\mathbf{x}}_{\mathbf{2}}^{\boldsymbol{i}}\right)\right)\right) \qquad (4.39)$$

The *n* channels considered in equation (4.39) can be expressed in terms of Histogram of Gradient (HOG with 31 gradient orientation bins) descriptors.

Chapter 5

The Proposed Tracker

In this chapter, the working and implementation of our robust tracker is described. Fig 5.1 is a pictorial representation of the proposed tracker. Fig 5.1 (a) is a bunch of consecutive input image frames from a video. Fig 5.1 (b) shows the auto-initialization of the first frame – which is the input for KRLS training. The trained KRLS classifies the images into backgrounds and foregrounds. Fig 5.1 (c) shows the low confidence values (correlation peak) detection and redection with generation of saliency map in the reduced search region followed by generating thresholded binary image with post processing and finally a bounding box is drawn around the redetected object. The last image with the redetected bounding box is again fed back for KRLS training. This redetection scheme actually improves the tracking performance by many folds.

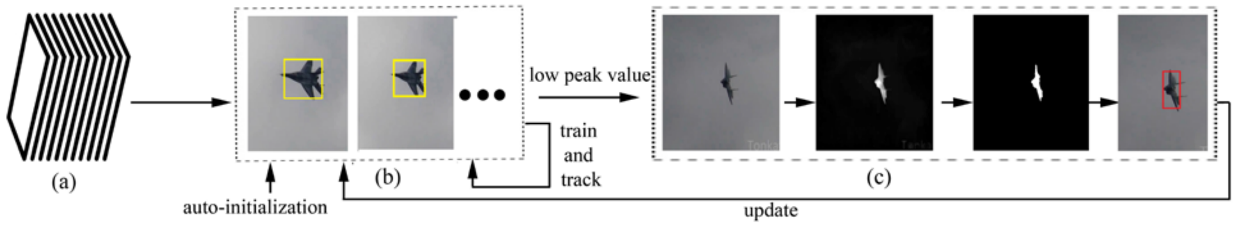


Fig 5.1: Illustration of our approach: (a) input frames, (b) auto initialization and KRLS training, (c) redetection,(from left to right) saliency map generation, binary image thresholding with post processing and bounding box drawn around redetected object

In the following sections, we will discuss the algorithm and its implementation in depth.

5.1 Algorithm Formulation

The pseudocode for the tracking algorithm is presented in **Algorithm 1**. Input images from a video sequence is converted into consecutive frames numbered sequentially and stored in a folder.

Another output folder is created, where the image sequence with output containing a rectangular bounding box around object in each frame is stored. In the tracking algorithm, the Gaussian kernel variance, regularization parameter and chosen correlation response threshold are also needed to be provided as an input along with the input image sequence. The algorithm is written with 4 basic functions.

The first function `main` calls the detection function for the first frame. It also calls the correlation response function for calculating the correlation response for each image.

The second function `saliency_map` is the detection function that creates a binary saliency map image and also performs post-processing to improve the binary classification.

The third function `correlation_response` is the KRLS filter, which is training and detecting the object in each frame. Again the correlation response is checked for each frame in this function and if it goes below a certain threshold, the `saliency_map` is called again for redetection. After redetection, the KRLS filter again starts evaluating the correlation response considering the redetected binary saliency map as the first frame.

The fourth function `boundary_box` takes in the binary saliency map image as an input and sends back the output image $\mathbf{O_i}$ with a boundary box around it.

5.1 **Algorithm 1**: Proposed Tracking Algorithm

Input: Image frames $\mathbf{I_1}, \mathbf{I_2}, \ldots, \mathbf{I_n}$ from the video sequence, where **n** is total number of frames, Gaussian kernel with variance **var**, regularization parameter λ, correlation filter response threshold γ.

Output: Bounding box around target object in each of output image frames $\mathbf{O_1}, \mathbf{O_2}, \ldots, \mathbf{O_n}$ of the video sequence.

```
function $O_i$ = main($I_i$, var, $\lambda$, $\gamma$)
  $I_i$ = rgb2gray($I_i$);  // Converted to grayscale image
  $s\_m_1$ = saliency_map($I_1$);  // saliency map of 1st frame
  $O_1$ = $s\_m_1$;
  $O_i$ = correlation_response($s\_m_1$, $I_i$, var, $\lambda$, $\gamma$);
  return;
end

function s_m_i = saliency_map(I_i)
  A_i = I_i;
  B_i = I_i;
  if(p,q-th pixel ∈ Background(p,q))      s_m_i = 0;
  else                                    s_m_i = ∞;
  for j = 1:3
    if(mod(j,2) == 1)
      {
      for each p,q, traverse via raster scan
        {              // (p,q) is the chosen testing pixel
        for each of upper and left neighbor of p,q-th pixel
          {
          Dist_i(Path_(x,y)(p,q)) = max{B_i(x,y),I_i(p,q)}
                                          – min{A_i(x,y),I_i(p,q)};
          if(Dist_i(Path_(x,y)(p,q)) < s_m_i)
            {                    // (x,y) is the background pixel
            s_m_i = Dist_i(Path_(x,y)(p,q));
            B_i(x,y) = max{B_i(x,y),I_i(p,q)};
            A_i(x,y) = min{A_i(x,y),I_i(p,q)};
            }
          }
        }
      }  // Dist_i(Path_(x,y)(p,q)) is the distance between testing
    else //pixel (p,q) and background seed pixel (x,y) in image I_i
      {
       for each p,q, traverse via inverse raster scan
        {
        for each of lower and rightt neighbor of p,q-th pixel
          {
          Dist_i(Path_(x,y)(p,q)) = max{B_i(x,y),I_i(p,q)}
                                          – min{A_i(x,y),I_i(p,q)};
          if(Dist_i(Path_(x,y)(p,q)) < s_m_i)
            {
            s_m_i = Dist_i(Path_(x,y)(p,q));
            B_i(x,y) = max{B_i(x,y),I_i(p,q)};
            A_i(x,y) = min{A_i(x,y),I_i(p,q)};
            }
          }
        }
      }
    // Post processing below:
    n_hist_i = normalize(hist(s_m_i));  // Normalize between 0 to 1
    m_g = mean(n_hist_i);    // m_g global mean
    if(n_hist_i < m_g)         n_hist_1_i = n_hist_i;
    else                       n_hist_2_i = n_hist_i;
    m_1 = mean(n_hist_1_i);
    m_2 = mean(n_hist_2_i);
    thres_var_(p,q)i=sum(n_hist_1_i).(m_1-m_g)^2)+sum(n_hist_2_i.(m_2-m_g)^2);
    thresᵢ = max(thres_var(p,q)ᵢ);  // Binary image thresholding
    if(s_mᵢ < thresᵢ)   s_mᵢ = 0; // Black
    else                s_mᵢ = 1; // White
    [Oᵢ, x_cordᵢ, y_cordᵢ, widthᵢ, heightᵢ] = boundary_box(s_mᵢ);
    s_mᵢ = Oᵢ;
  return;
end

function Oᵢ = correlation_response(s_mᵢ, Iᵢ, var, λ, γ)
  // s_mᵢ is the i-th base vector of saliency map image matrix s_mᵢ
  // c_rᵢ is the correlation response vector of i-th base vector
  for(i=2;i<n+1;i++)
    {
    for(k=1;k<(size(s_mᵢ₋₁)+1);k++) // size(s_mᵢ₋₁) is the
      {      // total no. of base vectors of Iᵢ for training
      aₖ = ifft2(sum(conj(fft2(s_mᵢ₋₁)) .*fft2(s_mᵢ₋₁), 3));
       bₖ = 2*(s_mᵢ₋₁(:)'*s_mᵢ₋₁(:) – aₖ);// s_mᵢ₋₁ is a base vector
      cₖ = exp(-1 / var * abs(bₖ) / numel(bₖ));
      dₖ = fft2(Iᵢ) ./ (fft2(cₖ) + λ);   // KRLS Training
      eₖ = ifft2(sum(conj(fft2(Iᵢ)) .*fft2(s_mᵢ₋₁), 3));
      fₖ = s_mᵢ₋₁(:)'*s_mᵢ₋₁(:) + Iᵢ (:)'* Iᵢ (:) – 2*eₖ;
      gₖ = exp((-1 / var )* abs(fₖ) / numel(fₖ));
      c_rᵢ,ₖ = real(ifft2(dₖ .* fft2(gₖ))); // KRLS Detection
      }
    c_rᵢ = max(c_rᵢ,ₖ);// Finds the maximum correlation response
                 // among all the input base vectors for image Iᵢ
    if(s_mᵢ < thresᵢ)   s_mᵢ = 0;     // Black
    else                s_mᵢ = c_rᵢ;   // White
    [Oᵢ, x_cordᵢ, y_cordᵢ,  widthᵢ, heightᵢ] = boundary_box(s_mᵢ);
    if( c_rᵢ < γ && c_rᵢ ≥ (γ-0.1) )
      Sᵢ = (1.5* widthᵢ) * (1.5* heightᵢ);
      s_mᵢ = saliency_map(Sᵢ);
    else if( c_rᵢ < (γ-0.1) && c_rᵢ ≥ (γ-0.2) )
      Sᵢ = (2* widthᵢ) * (2* heightᵢ);
      s_mᵢ = saliency_map(Sᵢ);
    else if( c_rᵢ ≤ (γ-0.1) )
      Sᵢ = Iᵢ;
      s_mᵢ = saliency_map(Sᵢ);
    else
      s_mᵢ = s_mᵢ₋₁;
    }
  return;
end

function [Oᵢ,x_cordᵢ,y_cordᵢ, widthᵢ,heightᵢ]=boundary_box(Objᵢ)
              // Drawing bounding box in output images Oᵢ
  Oᵢ = regionprops(Objᵢ, 'BoundingBox', 'Centroid');
  x_cordᵢ = floor(Oᵢ.BoundingBox(2));
  y_cordᵢ = floor(Oᵢ.BoundingBox(1));
     // Coordinate of top leftmost corner of bounding box
  widthᵢ = Oᵢ.BoundingBox(3); // width of bounding box
  heightᵢ = Oᵢ.BoundingBox(4);// height of bounding box
  open image(Oi);
  rectangle('Position',[y_cordi, x_cordi, widthi, heighti]);
end
```

5.2 Auto-initialization in First Frame

The auto initialization in the first frame is performed by using the Fast Minimum Barrier Distance detector (discussed in chapter 3) to generate saliency map and eventually the binary saliency map is given threshold (both local and global) using Otsu's method (discussed in next subsection) for achieving better accuracy with the well-known image processing thresholding technique. After getting the binary saliency map in the first frame, a rectangular bounding box is constructed and drawn around the object to visually differentiate it from the background.

5.2.1 Saliency Map

The input images are converted to gray-scale images from RGB (for each channel and the outputs are added at the end), as the computations with grayscale images is much easier and saves a lot of resource. All the saliency maps generated are binary, but the output frames with bounding box (obtained after tracking) are displayed and stored as colored images – serving the purpose of both auto navigation and doing so in real-time smoothly. Saliency map is generated by computing the minimum distance (f_λ) between each of the pixels and the background pixels on an image. Let the coordinate of a chosen pixel be (*p*,*q*) and a particular background pixel be (*x*,*y*). Now the adjacent pixels (described in section 3.1.1, chapter 3) of (*p*,*q*) i.e. (*p*-1,*q*), (*p*+1,*q*), (*p*,*q*-1) and (*p*,*q*+1) are taken into account to evaluate the distance by considering path $\pi_{(x,y)}^{(p,q)} = <\pi(0),$ $\pi(1),\ldots\ldots,\pi(n)>$, such that, $\pi(0) = (x,y)$ and $\pi(n) = (p,q)$. A distance cost function or weight function or length of path λ(*π*) (described in section 3.1.1, chapter 3), associated with each pixel is used to generate the saliency map **s_m** for an image **I** as follows:

$$f_\lambda(p,q) = \min_{\lambda \in \pi_{(x,y)}^{(p,q)}} \lambda(\pi)$$

where, $\pi_{(x,y)}^{(p,q)}$ is the set of all possible paths joining background pixels (*x*,*y*) and (*p*,*q*). The minimum barrier distance function (described in equation (3.3), chapter 3) is used as the distance cost function:

$$f_\lambda = \min_{\pi(\mathrm{i}) \in \mathrm{I}_{(x,y),(p,q)}} (\lambda^{+}(\pi) - \lambda^{-}(\pi))$$

$$= \min_{\pi(\mathrm{i}) \in \mathrm{I}_{(x,y),(p,q)}} \left(\max_{\boldsymbol{i=0,1,\dots n}} \{\lambda(\pi(\mathrm{i}))\} - \max_{\boldsymbol{i=0,1,\dots n}} \{\lambda(\pi(\mathrm{i}))\} \right) \quad (5.1)$$

Now, let **s_m** be the final MBD map, such that:

$$\mathbf{s\_m} = \begin{cases} 0, & \text{for all the background pixels } (x,y) \\ \infty, & \text{for all other pixels in the image } (p,q) \end{cases}$$

Let A(*x*,*y*) and B(*x*,*y*) be the highest and lowest pixel values on the entire path *π* from background (*x*,*y*) to chosen pixel (*p*,*q*) in a single scan via the adjacent pixels. A(y) and B(y) are initialized as a matrix equal to the pixel intensities of the image **I**.

Next each pixel (*p*,*q*) is traversed via raster scan and inverse raster scan (described in section 3.1.4, chapter 3). During raster scan (*p*,*q*-1) and (*p*-1,*q*) adjacent pixels are updated, whereas in inverse raster scan, (*p*,*q*+1) and (*p*+1,*q*) adjacent pixels are updated using equations (3.4 and 3.5) from chapter 3 as:

$$f_\lambda = \max_{\boldsymbol{i=0,1,\dots n}} \{\lambda(\pi(\mathrm{i}))\} - \min_{\boldsymbol{i=0,1,\dots n}} \{\lambda(\pi(\mathrm{i}))\}$$

$$= \max\{A(y), I(p,q)\} - \min\{B(y), I(p,q)\} \quad (5.2)$$

s_m is modified in each scan by choosing the minimum between previous f_λ and current **s_m** as:

$$\mathbf{s\_m} = \min(\text{current } \mathbf{s\_m}, \text{ previous } f_\lambda) \quad (5.3)$$

A(x,y) and B(x,y) are modified after each raster and inverse raster scan as the path changes every time and this in turn modifies f_λ and **s_m**. The procedure is iteratively repeated 3 times with 2 raster scans and 1 inverse raster scan. The size of image **I** (the search area for fast MBD) is provided by the redetection described in latter section. The post processing of the saliency map is discussed in the next section.

5.2.2 Post Processing

The quality of binary saliency map generated by the fast MBD detector is enhanced by giving a threshold using Otsu's method. Taking into consideration the varying illumination, background, object size, noise content in each frame, a constant threshold if applied to the binary image would be highly inefficient. Thus a well-known global threshold based on inter-class variance maximization [42] is considered to adaptively choose a threshold in each frame depending on the above mentioned varying parameters.

Let **H** be the *histogram* of *saliency map* **s_m** that is made of pixels with intensity levels $\mathbf{L} \in [0, l\text{-}1]$ and n_i be the number of pixels with intensity i, where $\mathrm{i} \in \mathbf{L}$, then,

$$\mathbf{H} = \sum_{i=0}^{l-1} n_i \tag{5.4}$$

Let $\mathbf{H_n}$ be the *normalized histogram* of **s_m**, such that for every threshold value t, $t \in \mathbf{L}$, we divide the normalized histogram into two groups – $\mathbf{C_1} \in \mathbf{H_n}^{\boldsymbol{i}}$, $i \in [0,t]$ and $\mathbf{C_2} \in \mathbf{H_n}^{\boldsymbol{i}}$, $i \in [t+1,l\text{-}1]$, represented as:

$$\mathbf{P_1} = P(\mathbf{C_1}) = \sum_{i=0}^{t} \mathbf{H}_{\boldsymbol{n}}^{\boldsymbol{i}} \tag{5.5}$$

$$\mathbf{P_2} = P(\mathbf{C_2}) = \sum_{i=t+1}^{l-1} \mathbf{H}_{\boldsymbol{n}}^{\boldsymbol{i}} = 1 - \mathbf{P_1} \tag{5.6}$$

Let the mean intensity of all the pixels in group1 ($\mathbf{C_1}$) is represented as m_1 and defined as:

$$m_1 = \sum_{i=0}^{t} i.P(\mathrm{i}/\mathbf{C}_1) = \sum_{i=0}^{t} \frac{P(\boldsymbol{C}_1/\mathrm{i}).\mathrm{P}(i)}{P(\boldsymbol{C}_1)} = \frac{1}{\mathbf{P_1}} \sum_{i=0}^{t} i.\mathbf{H}_{\boldsymbol{n}}^{\boldsymbol{i}} \tag{5.7}$$

where, $P(i) = \mathbf{H}_{\boldsymbol{n}}^{\boldsymbol{i}}$ and $P(\mathbf{C_1}/i) = 1$. Similarly if m_2 be the mean intensity of all the pixels in group2 ($\mathbf{C_2}$), then it can be defined as:

$$m_2 = \frac{1}{\mathbf{P_1}} \sum_{i=t+1}^{l-1} i.\mathbf{H}_{\boldsymbol{n}}^{\boldsymbol{i}} \tag{5.8}$$

Let m_g be the *global intensity mean* and m_t be the mean intensity up to t levels. Next, the *inter- class variance* is derived in [42] as:

$$\sigma_b^2 = \mathbf{P_1}.(m_1 - m_g)^2 + \mathbf{P_2}.(m_2 - m_g)^2 = \frac{m_g.\mathbf{P_1} - m_t}{\mathbf{P_1}.(1 - \mathbf{P_1})} \tag{5.9}$$

For each $t \in \mathbf{L}$, we evaluate the inter-class variance as σ_b^2 (t) and the optimal threshold t_{opt} for the saliency map **s_m** is given by:

$$\sigma_b^2\ (t_{opt}) = \max_{0<t<l-1} \ \sigma_b^2(t) \tag{5.10}$$

The result obtained by thresholding the saliency map **s_m** with σ_b^2 (t_{opt}) produces a binary image (like the one in fig 5.1(c) third image from left to right), such that the background and the object can be distinguished very easily.

Once we get the binary image from Otsu's method after thresholding, we are interested in assigning a rectangular bounding box around the object – so that while tracking in auto navigation, we never make the object out of sight. Inbuilt MATLAB functions *Regionprop* and *Rectangle* are used to make and draw a rectangular bounding box around the object with a chosen color – **thus auto initializing the object in the first frame**. We also store the ***height*** and ***width*** of this bounding box to be used for determining the search area while redetecting.

5.3 KRLS Training

The auto initialized binary saliency map of the first frame is fed as an input to the KRLS filter for training and henceforth the trained classifier detects object in the subsequent frames. The training on the previously tracked images and the detection in the following images runs in parallel.

From the saliency map, an input sample vector **x** is taken and a Circulant matrix **M(x)** is generated from it as described in section 4.1.2 of chapter 4. We have performed non-linear regression to train the classifier and thus a Gaussian kernel function $k(\mathbf{x}, \mathbf{y})$ (similarity function for the input samples **x** and testing labels **y**) as defined in section 4.3.2 of chapter 4 is used:

$k(\mathbf{x}, \mathbf{y}) = \exp(-\frac{1}{\sigma^2} \|\mathbf{x} - y\|^2)$, where σ^2 is the constant variance. All the elements $k(\mathbf{x}, \mathbf{y})$ are stored in the kernel matrix **K** as shown in equation (4.18) in chapter 4. Now, a kernel weight **α** is defined in equation (4.19), which represents the importance of each kernel associated with the distance function measurement as:

$$\boldsymbol{\alpha} = (\mathbf{K} + \lambda\mathbf{I})^{-1}\mathbf{y} \tag{5.11}$$

In section 4.2.2 of chapter 4, the kernel $k(\mathbf{x}, \mathbf{y})$ is shown to be Circulant, if it satisfies equation (4.20) and transformed to higher order feature space for making the computations in Fourier domain. The Gaussian kernel used, satisfy the criteria mentioned in section 4.2.2 of chapter 4. Thus, the Gaussian kernel weight is evaluated with equation (4.24) as:

$$\boldsymbol{\alpha} = \mathbf{F}^{-1} (\underline{\mathbf{y}} (\underline{k}^{\mathbf{xy}} + \lambda)^{-1}) \tag{5.12}$$

where, $\underline{k}^{\mathbf{xy}}$ is the kernel correlation vector in the Fourier domain and is defined in equation (4.36) as:

$$k^{\mathbf{xy}} = \exp(-\frac{1}{\sigma^2}(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\mathbf{F}^{-1} (\underline{\mathbf{x}}^* \odot \underline{\mathbf{y}})) \tag{5.13}$$

The kernel weight **α**, computed with equation (5.12) is the primary KRLS training parameter that will be used by the KRLS filter in the next section to calculate the correlation response of the testing labels **y** with respect to the input base sample **x**. The discontinuity of the consecutive image frames creates noise while computing correlation in the Fourier domain and is removed by pre-processing with a cosine window as described in section 4.4 of chapter 4. Also the linearity property of Fourier Transform is exploited to take inputs from multiple channels (we used HOG features with 31 channels/bins) by simply adding them in the frequency domain to get the output as mentioned in section 4.5 of chapter 4. Equation (4.39) is used to compute the kernel correlation $k_n^{\mathbf{xy}}$ from *n* channels for a Gaussian kernel function (σ^2 variance) as follows:

$$k_n^{\mathbf{xy}} = \exp\left(-\frac{1}{\sigma^2}\left(\|\mathbf{x}\|^2 + \|\mathbf{y}\|^2 - 2\,\mathbf{F}^{-1}\left(\sum_{i=1}^{n}\left(\underline{\mathbf{x}}^{i*} \odot \underline{\mathbf{y}}^{i}\right)\right)\right. \tag{5.14}$$

5.4 KRLS Detection

In the previous section, we computed the kernel correlation function $k^{\mathbf{xy}}$ and the Gaussian kernel weight $\boldsymbol{\alpha}$ for training the KRLS filter. For detecting an object in the subsequent frame with the trained filter, we use the KRLS classifier $\mathbf{O}^{\mathbf{k}}_{\mathbf{i}}(\boldsymbol{y})$ defined in equation (4.28) of chapter 4 as follows:

$$\mathbf{O}^{\mathbf{k}}_{\mathbf{i}}(\boldsymbol{y}) = \mathbf{F}^{-1}(\underline{k}^{\mathbf{xy}} \odot \underline{\boldsymbol{\alpha}}) \tag{5.15}$$

where, $\odot$ evaluates the convolution between $\underline{k}^{\mathbf{xy}}$ and $\underline{\boldsymbol{\alpha}}$ in the Fourier domain (element-wise operation). Equation (5.15) represents the *correlation response* of a testing sample **y** with the KRLS filter trained on the previously detected frames. Now, the object in the *i*-th testing frame is given by the maximum correlation response as follows:

$$\mathbf{O}_{\mathbf{i}}(\boldsymbol{y}) = \max_{\boldsymbol{k}} \boldsymbol{O}_{\mathbf{i}}^{\boldsymbol{k}}(\mathbf{y}) \tag{5.16}$$

where, $\boldsymbol{k}$ represents one of the input base vectors from the Circulant matrix $\mathbf{M(x)}$ and the maximum of all such correlation response is evaluated for each of the *i*-th frames and stored in $\mathbf{O}_{\mathbf{i}}(\boldsymbol{y})$. Finally, a bounding box is wrapped around the detected object with the maximum correlation response to store in memory and display.

5.5 Online Update - Redetection

Evaluating the saliency map in each image frame of a video sequence or evaluating the saliency map over the entire image is computationally very expensive. Thus an adaptive redetection method is proposed which do not take down the speed of our fast kernelized correlation filter tracker. After all, we made the regression model Circulant and kernelized for maintaining a high tracking speed. The redetection of the object in our proposed tracker is directly related to the correlation response of the KRLS filter. This is because it has been seen experimentally that as the tracker starts failing to lose the object, its correlation response starts going down. The search area for the redetection of object using fast MBD detector is made dependent on the height and width of the bounding box of the last successful tracking result.

In the KRLS detector section, we saw in equation (5.15) that the correlation response is computed for each of the base input vectors and the maximum of those correlation responses (equation 5.16) is considered as the one corresponding to the object. Now let $\boldsymbol{\gamma}$ be the correlation filter response threshold that we will check for redetection, $\boldsymbol{S}$ be the adaptive search area for redetection whenever correlation filter response goes below the threshold $\boldsymbol{\gamma}$ and ***height*** and ***width*** be the dimension of the last successful bounding box such that:

Case 1: If $0.4 \leq \boldsymbol{\gamma_1} < 0.5$, $\boldsymbol{height_1} = 1.5^* \boldsymbol{height}$, $\boldsymbol{width_1} = 1.5^* \boldsymbol{width}$ and

$$\boldsymbol{S_1} = \boldsymbol{height_1}^* \boldsymbol{width_1} = 2.25^*\boldsymbol{S}$$

Case 2: If $0.3 \leq \boldsymbol{\gamma_2} < 0.4$, $\boldsymbol{height_2} = 2^* \boldsymbol{height}$, $\boldsymbol{width_2} = 2^* \boldsymbol{width}$ and

$$\boldsymbol{S_2} = \boldsymbol{height_2}^* \boldsymbol{width_2} = 4^*\boldsymbol{S}$$

Case 3: If $\boldsymbol{\gamma_3} < 0.3$, $\boldsymbol{S_3} = \boldsymbol{S}$ i.e. searched over entire image

where, the subscripts 1, 2 and 3 for each variable correspond to the respective cases. This adaptive search area $\boldsymbol{S}$ is again fed as input to the fast MBD detector for redetection and the saliency map generation with post processing is provided as the initialization for the KRLS filter. This redetection is performed iteratively whenever the object is partially or absolutely outside the bounding box.

5.6 Implementation

All the input image frames (**I**$_i$) are stored in a single folder and named sequentially. Fast MBD detection is performed on the first frame and the saliency map is passed on to the KRLS filter. The regularization parameter λ mentioned in equation (5.12) is set to 10^{-4} as recommended in [3]. The correlation response threshold parameter $\boldsymbol{\gamma}$ mentioned in tracking algorithm and the equations (5.17 - 5.19) is set to 0.5 after testing rigorously with various datasets. During detection and redetection, we perform raster scanning and inverse raster scanning 3 times in total (2 raster and 1 inverse raster scan) for each frame as stated in [8]. All the output image frames (**O**$_i$) with rectangular bounding box around the object is stored in a folder and named sequentially, similar to the input image frames.

Chapter 6

Experimentation and Results

The proposed tracking algorithm and its results were presented in IEEE 28th International Conference on Tools with Artificial Intelligence at San Jose, CA in 2016. This chapter discusses the results submitted to the conference as a paper [47]. A kernelized correlation filter based tracker with adaptive redetection (boosting the performance) is implemented in this work. Our method is also compared with 6 other *state-of-the-art* trackers, published in recent time and exhibit promising results. All the seven trackers (ours and the six others) are rigorously tested with 14 challenging datasets, where the object is subjected to variations of illumination, shape, size, in-plane and out-plane rotations throughout the sequences.

In this chapter, we will discuss about the implementation platform, comparison metrics, rigorous quantitative and qualitative comparisons with the other six trackers to show the robustness of proposed tracker in outperforming all of them – thus could be implemented for automated navigation of unmanned aerial vehicle in real-time.

6.1 System Information

The proposed algorithm is implemented in C++ and OpenCV v.3.2.14 on a PC with an Intel (R) Xeon (R) W3520 2.67 GHz CPU and 6 Gigabytes of RAM. All the other competing tracker codes are obtained from the respective authors' websites and implemented on the same machine for consistency. Some of the competing trackers were implemented in MATLAB, some were implemented in C++ and some had *mex* files generated from C++ code to be implemented on

MATLAB. All the parameters of the competing trackers were set to the values recommended by the respective authors.

6.2 Datasets and Other Trackers

Our problem statement to guide an unmanned aerial vehicle with their navigation do not allow us to test the result on the available benchmark datasets [30] as those videos represent other kind of object tracking problems like face tracking, on-stage dancer tracking, doll tracking etc. A navigation system requires sky in the background and flying objects to be tracked. Thus, extensive search was done to find videos fulfilling such criteria. *Aircraft* (620 frames), *big_2* (382 frames), *airplane_001* (200 frames), *airplane_004* (200 frames), *airplane_005* (200 frames), *airplane_006* (200 frames), *airplane_007* (200 frames), *airplane_011* (300 frames), *airplane_012* (300 frames), *airplane_013* (300 frames), *airplane_015* (300 frames), *airplane_016* (300 frames) from [43], *youtube_dataset_2* (475 frames) and *youtube_dataset_3* (301 frames) were used to evaluate our algorithm on UAV-like flying objects and test it with other 6 trackers.

The six other *state-of-the-art* trackers used for comparison with the proposed tracker are *CT* [12], *STC* [46], *CN* [32], *DSST* [33], *SAMF* [2] and *KCF* [3].

6.3 Comparison Metric

We are basically interested in two performance parameters to decide if a tracker is better than the other. Those two parameters are:

1. Execution speed to make the tracker work in real-time.
2. Robust tracking result with bounding box around the object in all frames without failure.

The first performance parameter can be verified very easily during runtime, just by comparing the execution speeds of all the trackers in the respective codes. But comparing the second performance parameter to decide on a better tracker is much more complex.

All trackers are very sensitive to initialization (i.e. the bounding box around the object in the first frame). Many trackers behave abnormally if the size or position of the bounding box is slightly varied. This is because the bounding box determines the positive and negative samples for discriminative tracking and a slight variation may change the prediction in the subsequent frames by a huge margin. Thus, *One Pass Evaluation* (*OPE*) and *Temporal Robustness Evaluation* (*TRE*) experiments are generally performed for robust evaluation of any tracker [30]. In case of *OPE*, the trackers are ran from first frame till the last frame and compared with the *ground truth* (*GT*) to provide some average estimations about tracking or missing the target, for each of the sequences.

On the other hand, in *TRE*, unlike *OPE*, the average estimations about tracking or missing the target is computed on a segment of the sequence instead of the entire sequence of frames. Finally, an average of all such *TRE*s in each segment of the frame sequence is computed to get the estimations. This is done to avoid the misinterpretation of tracker performance due to wrong initialization in the first frame.

6.4 Quantitative Evaluation

The comparison of the tracking performance with the *ground truth* (*GT*) by *OPE* and *TRE* is mathematically expressed in terms of *Center Location Error* (*CLE*), *Precision Rate* (*PR*) and *Success Rate* (*SR*) (also defined in [30]).

CLE is defined as the average Euclidean distance between the centermost pixel coordinate of the bounding box given by *ground truth* and the same by the tracked object. Lower *CLE*, suggests better performance of a tracker. But if the tracker fails in some frame to track the object, then this distance could be any random number in the Euclidean plane. Thus, *PR* and *SR* were proposed. *PR* is computed as the percentage of frames in which *CLE* is lower than a given threshold. In our experiment, we kept this threshold to be 20 pixels as suggested in [30].

On the other hand, *SR* evaluates the overlapping of the bounding box. Tracked results are considered to be successful if $\frac{a_t \cap a_g}{a_t \cup a_g} > \theta$, where $\theta \in [0, 1]$, a_t and a_g denotes the areas of the bounding boxes of tracker's output and *ground truth* respectively. *SR* is defined as the percentage of frames, where the overlap rates are greater than the threshold θ. Generally, θ is set to 0.5 (in [30] and also in our experiment), which reflects a 50% overlap ratio threshold.

6.4.1 Ground Truth Annotation

The trackers can be compared with the computation metrics discussed earlier, only if all the frames in all the 14 datasets are manually labelled. We have manually noted down the pixel coordinates of a rectangular bounding box around the object in all these frames for comparison. Basically, in each frame the *x* and *y* coordinates of the top-leftmost corner of the rectangular bounding box is stored along with the *height* and *width* of the bounding box. Thus the *ground truth* consists of 4,278 coordinates and dimensions of 4,278 manually labelled bounding boxes in all the frames of 14 datasets.

6.4.2 OPE, TRE Computation

OPE is computed for each of the seven trackers on all the 14 datasets ran from first frame till the last frame and compared with the *ground truths* to obtain the *CLE, , PR* and *SR.*

In *TRE* computation, the comparison with the *ground truth* to obtain the *CLE, PR* and *SR* is carried out in several segments of the sequence. For example, first segment may start from 10th frame and continue till last frame. The next segment then starts from 20th frame and continues till the last frame. Likewise, the last segment runs from 190th frame till 200th frame if the total number of frames in the sequence is 200. Finally, the average *CLE, PR* and *SR* are evaluated from all the segments. This process is repeated for each of the seven trackers on all the 14 datasets.

6.4.3 Comparison with other *State-of-the-Art* Trackers

The results for the quantitative evaluation between our approach and the other six competing trackers *CT* [12], *STC* [46], *CN* [32], *DSST* [33], *SAMF* [2] and *KCF* [3] has been tabulated in Table I. For *OPE* evaluation, 4,278 frames from 14 datasets were thoroughly examined for each of the seven trackers (total 29,946 frames). For *TRE* evaluation, each of the 14 datasets is divided into 20 segments and thus each of the tracker is examined with around 85,560 frames (total of 5,98,920 frames for seven trackers).

TABLE I
QUANTITATIVE ANALYSIS OF PROPOSED AND SIX COMPETING TRACKERS ON 14 DATASETS. THE **BEST** AND THE <u>SECOND BEST</u> RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND <u>UNDERLINE-POINT-STYLES</u> RESPECTIVELY.

	OURS	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Average Precision Rate (OP*E)*	**0.83**	0.15	<u>0.49</u>	0.44	0.46	0.48	0.44
Average Success Rate (*OPE)*	**0.62**	0.19	0.41	0.38	<u>0.41</u>	<u>0.41</u>	0.39
Average Precision Rate (*TRE)*	**0.78**	0.29	<u>0.54</u>	0.45	<u>0.54</u>	0.49	0.48
Average Success Rate (*TRE)*	**0.56**	0.31	0.41	0.39	<u>0.46</u>	0.43	0.42
CLE (in pixels)	**9.02**	232.72	<u>31.97</u>	84.85	54.92	45.71	91.88
Average Speed (fps)	**108.94**	22.11	32.87	30.04	7.09	5.53	<u>65.28</u>

It is evident from Table I that our proposed tracker generated best average *OPE* and *TRE* results, along with highest average speed and lowest average *CLE* on the 14 datasets, when compared with the six competing *state-of-the-art* trackers. The detailed evaluation of *OPE* and *TRE* of all the seven trackers on the 14 challenging datasets is also tabulated and the *PR* and *SR* of the respective *OPE* and *TRE* are presented in Table II, Table III, Table IV and Table V respectively. It can be observed in both Precision Rate and Success Rate tables, that the average performance on almost all the datasets is better compared to the six other trackers.

6.2 TABLE II
PRECISION RATE OF *OPE* OF THE PROPOSED AND SIX COMPETING TRACKERS ON 14 DATASETS. THE **BEST** AND THE <u>SECOND BEST</u> RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND <u>UNDERLINE-POINT-STYLES</u> RESPECTIVELY.

Tracker / Datasets	*OURS*	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Aircraft	**0.85**	0.18	<u>0.60</u>	0.16	0.17	<u>0.60</u>	0.51
airplane_001	**0.92**	0.02	0.38	0.20	0.26	<u>0.43</u>	0.12
airplane_004	**0.79**	0.25	0.42	0.44	<u>0.50</u>	0.21	0.37
airplane_005	**0.81**	0.09	0.36	0.33	0.32	<u>0.39</u>	0.27
airplane_006	**0.92**	0.22	0.65	0.54	0.54	<u>0.75</u>	0.53
airplane_007	**0.76**	0.08	0.46	<u>0.61</u>	0.36	0.27	0.37
airplane_011	**0.90**	0.27	0.31	<u>0.43</u>	0.28	0.31	0.25
airplane_012	0.74	0.02	0.83	0.15	<u>0.88</u>	**0.99**	0.81
airplane_013	**0.89**	0.02	0.26	<u>0.32</u>	<u>0.32</u>	0.19	0.12
airplane_015	**0.82**	0.35	0.49	0.73	0.58	0.57	<u>0.79</u>
airplane_016	**0.83**	0.18	0.65	<u>0.76</u>	0.73	0.52	0.45
big_2	0.89	0.31	0.85	0.82	<u>0.91</u>	**0.94**	0.85
youtube_dataset_2	**0.81**	0.04	<u>0.66</u>	0.65	0.56	0.54	0.46
youtube_dataset_3	**0.69**	0.06	0.12	0.07	0.07	0.05	<u>0.26</u>

6.3 TABLE III
SUCCESS RATE OF *OPE* OF THE PROPOSED AND SIX COMPETING TRACKERS ON 14 DATASETS. THE **BEST** AND THE <u>SECOND BEST</u> RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND <u>UNDERLINE-POINT-STYLES</u> RESPECTIVELY.

Tracker / Datasets	*OURS*	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Aircraft	**0.52**	0.15	0.34	0.18	0.13	<u>0.50</u>	0.42
airplane_001	**0.78**	0.02	0.32	0.17	0.27	<u>0.46</u>	0.12
airplane_004	**0.54**	0.49	0.47	<u>0.53</u>	0.42	0.43	0.44
airplane_005	**0.57**	0.15	<u>0.39</u>	0.20	0.26	0.28	0.23
airplane_006	**0.54**	0.19	<u>0.49</u>	0.43	0.45	0.46	0.43
airplane_007	<u>0.53</u>	0.13	0.47	0.46	**0.55**	0.34	0.49
airplane_011	**0.77**	0.21	0.33	<u>0.34</u>	0.29	0.31	0.20
airplane_012	0.47	0.08	0.48	0.31	<u>0.59</u>	**1.00**	0.70
airplane_013	**0.70**	0.04	<u>0.27</u>	0.19	0.30	0.11	0.06
airplane_015	**0.71**	0.45	0.49	<u>0.65</u>	0.59	0.38	0.54
airplane_016	**0.73**	0.22	<u>0.62</u>	0.58	0.56	0.66	0.58
big_2	0.61	0.30	0.58	0.58	**0.65**	0.56	<u>0.63</u>
youtube_dataset_2	**0.56**	0.05	0.25	<u>0.45</u>	0.40	0.27	0.43
youtube_dataset_3	**0.62**	0.10	0.16	0.24	0.22	0.06	<u>0.29</u>

We can find in Table II that 11 out of 14 best results in favor of our tracker, demonstrating Precision Rate, whereas 12 out of 14 best results favor our tracker in Table III, demonstrating Success Rate for *OPE*.

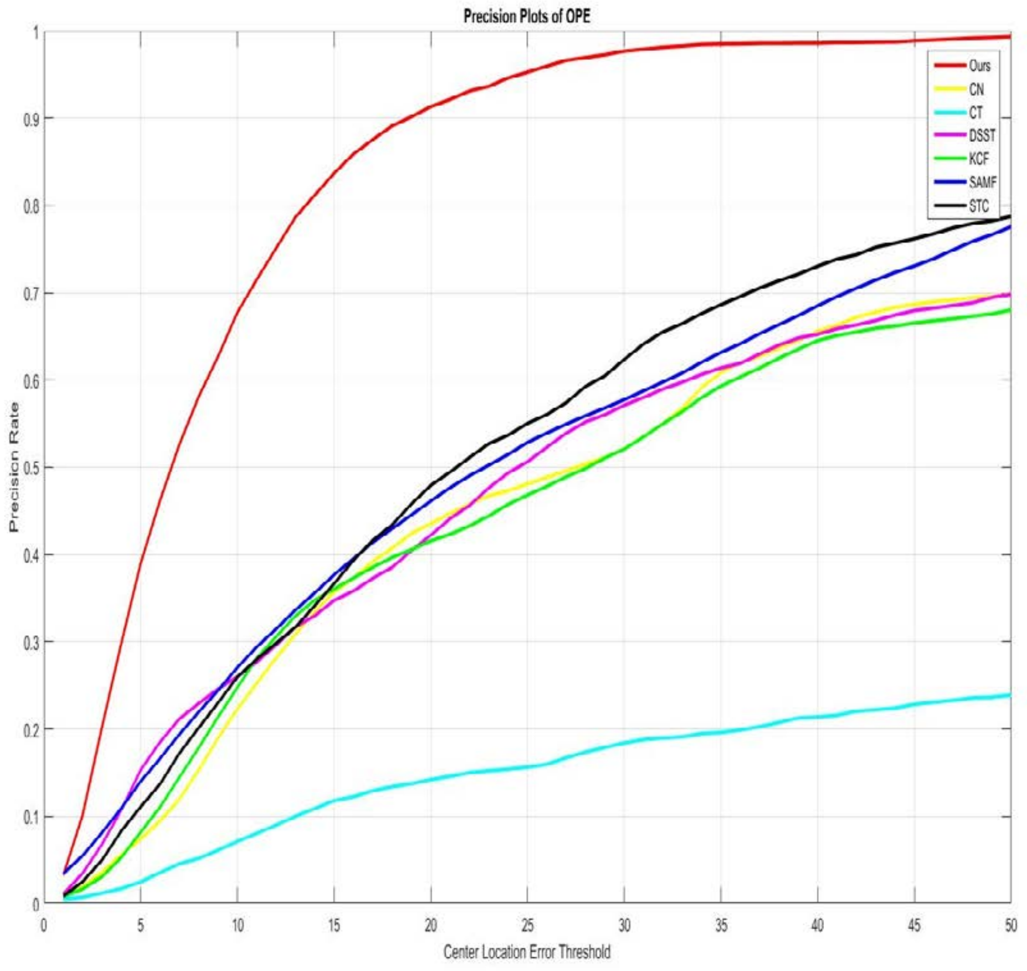


Fig 6.1: OPE Precision Plot demonstrating average precision rate of 7 competing trackers over 14 challenging datasets

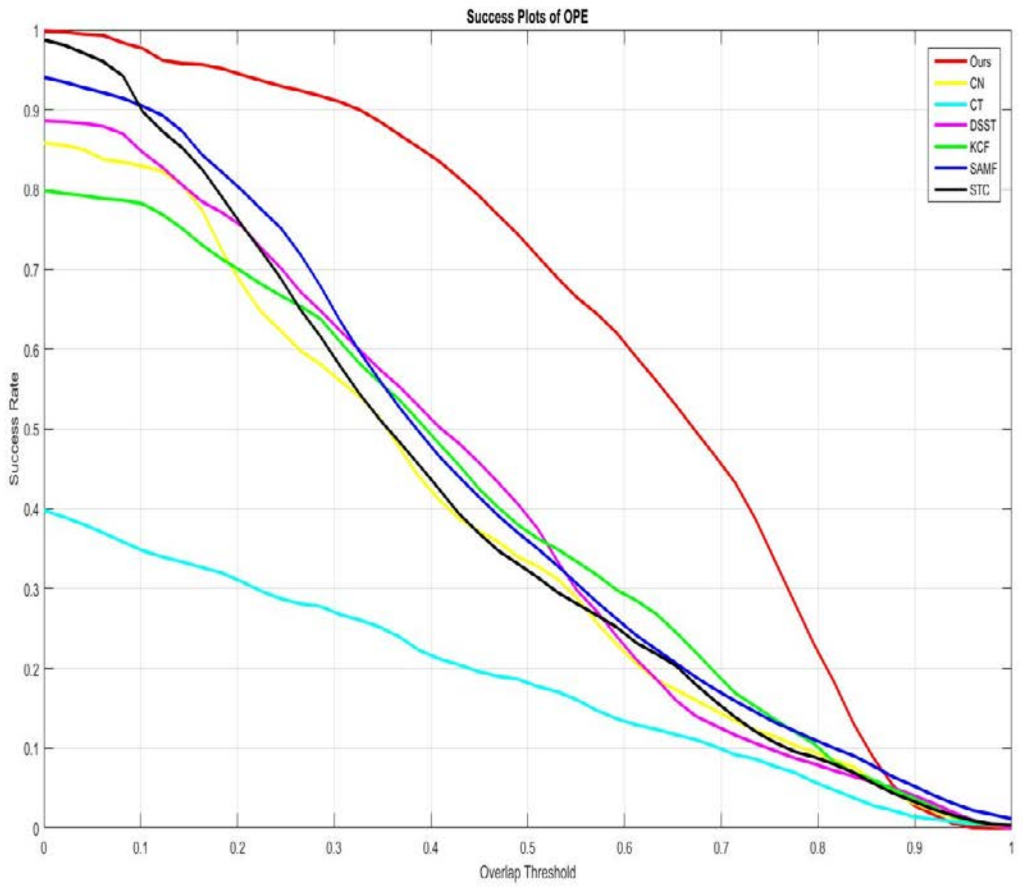


Fig 6.2: OPE Success Plot demonstrating average success rate of 7 competing trackers over 14 challenging datasets

The precision plot in fig 6.1 is demonstrating the Precision Rate of *OPE* in the y-axis and the CLE threshold in the x-axis. The success plot in fig 6.2 is demonstrating the Success Rate of *OPE* in the y-axis and the Overlap threshold in the x-axis. Both the plots are computed for all the seven competing trackers and a 50 point mean is evaluated to represent them correctly according to the Success Rate and Precision Rate definitions, defined in the earlier section. By definition, the plot with maximum area under the curve has best Success Rate in fig 6.2. We can observe that the *red* curve representing our proposed method has the maximum area under the curve and thus shows the best performance.

6.4 TABLE IV
AVERAGE PRECISION RATE OF *TRE* OF THE PROPOSED AND SIX COMPETING TRACKERS ON 14 DATASETS. THE **BEST** AND THE SECOND BEST RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND UNDERLINE-POINT-STYLES RESPECTIVELY.

Tracker / Datasets	*OURS*	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Aircraft	**0.85**	0.09	0.59	0.03	0.08	0.31	0.07
airplane_001	**0.92**	0.22	0.30	0.44	0.49	0.46	0.45
airplane_004	**0.78**	0.35	0.50	0.61	0.61	0.63	0.46
airplane_005	0.49	0.47	**0.59**	0.37	0.53	0.39	0.48
airplane_006	**0.92**	0.37	0.60	0.53	0.55	0.55	0.47
airplane_007	**0.76**	0.20	0.37	0.52	0.59	0.54	0.54
airplane_011	**0.89**	0.43	0.79	0.24	0.60	0.26	0.34
airplane_012	0.76	0.50	0.81	0.25	**0.89**	0.27	0.84
airplane_013	**0.89**	0.17	0.27	0.43	0.46	0.45	0.27
airplane_015	**0.70**	0.24	0.43	0.66	0.59	0.68	0.65
airplane_016	0.81	0.27	0.58	0.80	0.70	**0.82**	0.63
big_2	0.54	0.40	0.81	0.80	**0.85**	0.82	**0.85**
youtube_dataset_2	**0.82**	0.18	0.46	0.39	0.38	0.41	0.30
youtube_dataset_3	**0.73**	0.13	0.39	0.22	0.29	0.24	0.34

6.5 TABLE V
AVERAGE SUCCESS RATE OF *TRE* OF THE PROPOSED AND SIX COMPETING TRACKERS ON 14 DATASETS. THE **BEST** AND THE SECOND BEST RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND UNDERLINE-POINT-STYLES RESPECTIVELY.

Tracker / Datasets	*OURS*	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Aircraft	**0.52**	0.08	0.34	0.05	0.10	0.46	0.06
airplane_001	**0.78**	0.21	0.19	0.36	0.45	0.37	0.40
airplane_004	0.54	0.44	0.46	0.53	**0.56**	0.54	0.46
airplane_005	0.34	0.38	**0.49**	0.29	0.45	0.30	0.41
airplane_006	**0.52**	0.32	0.44	0.40	0.41	0.41	0.36
airplane_007	0.53	0.28	0.44	0.43	**0.57**	0.44	0.56
airplane_011	**0.77**	0.43	0.53	0.32	0.53	0.33	0.41
airplane_012	0.35	0.50	0.56	0.40	**0.78**	0.41	**0.78**
airplane_013	**0.71**	0.20	0.30	0.36	0.44	0.37	0.29
airplane_015	0.59	0.32	0.56	0.59	0.54	**0.60**	0.48
airplane_016	**0.66**	0.45	0.49	0.55	0.51	0.56	0.52
big_2	0.38	0.39	0.51	0.60	0.51	**0.61**	0.59
youtube_dataset_2	**0.56**	0.19	0.18	0.30	0.27	0.31	0.27
youtube_dataset_3	**0.61**	0.17	0.25	0.27	0.33	0.28	0.31

We can find in Table IV that 10 out of 14 best results (two second best result) in favor of our tracker, demonstrating Precision Rate of *TRE*, whereas 8 (two second best result) out of 14 best results favor our tracker in Table V, demonstrating Success Rate of *TRE*.

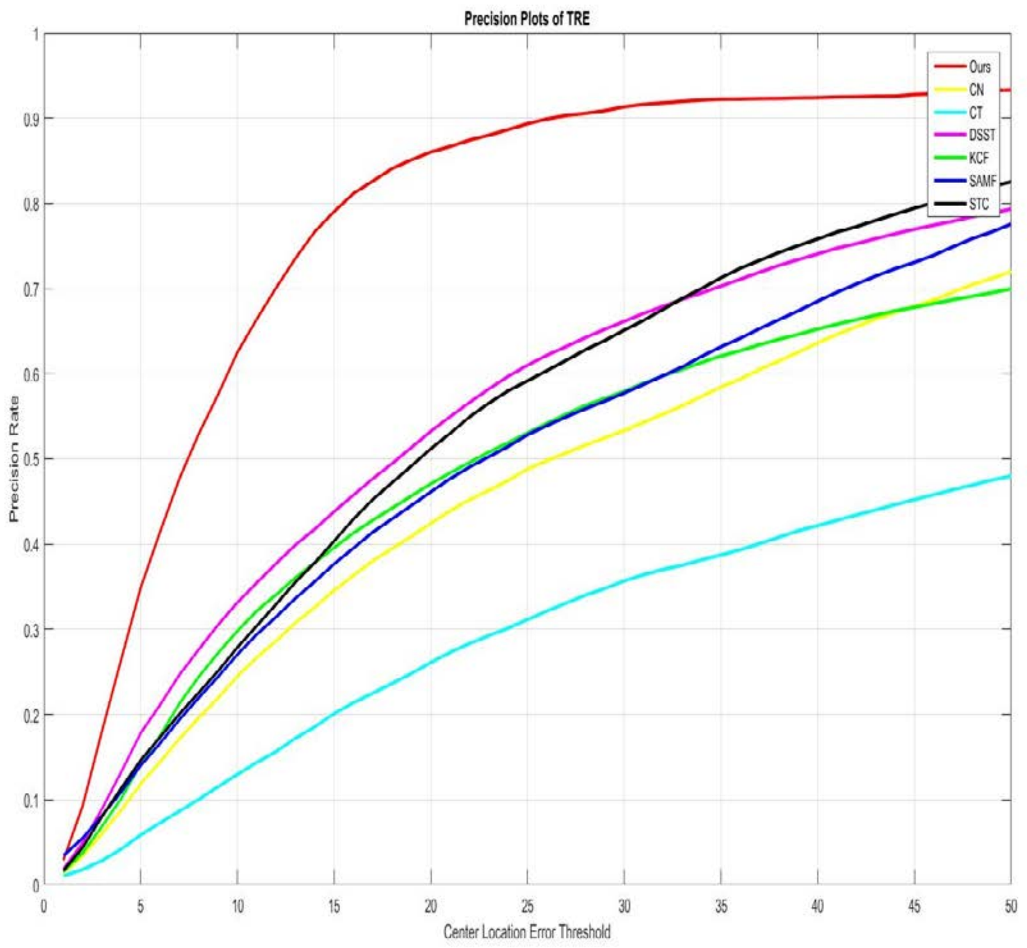


Fig 6.3: TRE Average Precision Plot demonstrating average precision rate of 7 competing trackers over 14 challenging datasets

Again, fig 6.3 and fig 6.4 represent the average Precision Rate and average Success Rate for *TRE*. These plots are very similar to the plots shown in figs 6.1 and 6.2. We can observe in these plots that our proposed approach has largest area (*red* curves in both the plots), whereas, *CT* has the least area. This means that our tracker has the best performance and *CT* has the worst performance in terms of Precision Rate and Success Rate for *TRE*.

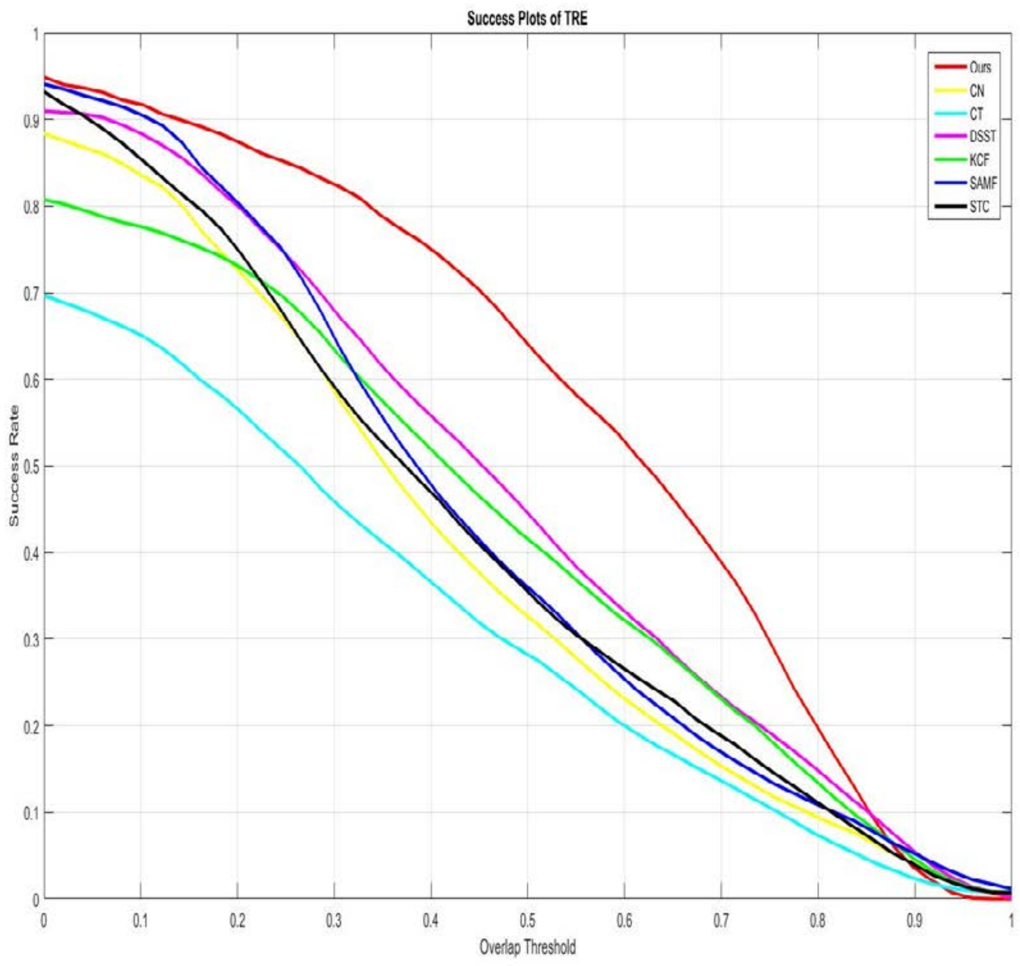


Fig 6.4: TRE Average Success Plot demonstrating average success rate of 7 competing trackers over 14 challenging datasets

6.4.4 Speed Comparison

As presented in Table VI, our algorithm achieves an average speed of **108.94 frames per second (fps)** on the 14 challenging video sequences. The second best tracker, *KCF*, has an average speed of 49.18 fps on the same video sequences. The slow speed of *KCF* is due to the fact that once it fails to track in a certain frame, then in the subsequent frames the correlation response goes down, resulting in delay of detection. Our approach re-detects the object once drifting starts (by observing the correlation response going below a threshold) and thus speed never goes down, making it 2 folds faster than *KCF*.

TABLE VI
SPEED COMPARISON OF 7 TRACKERS ON 14 DATASETS. THE **BEST** AND THE SECOND BEST RESULTS ARE HIGHLIGHTED WITH **BOLD-FACE** AND UNDERLINE-POINT-STYLES RESPECTIVELY.

Tracker Datasets	*OURS*	*CT*	*STC*	*CN*	*DSST*	*SAMF*	*KCF*
Aircraft	**105.62**	35.73	63.32	38.58	9.88	4.20	49.18
airplane_001	50.41	17.92	37.30	**89.34**	20.41	3.99	84.61
airplane_004	**99.26**	17.93	28.90	17.81	4.49	6.20	64.11
airplane_005	49.26	18.06	28.62	17.73	4.25	5.92	**60.58**
airplane_006	58.14	18.01	31.289	26.14	6.36	7.75	**90.87**
airplane_007	43.73	17.94	29.65	21.65	4.60	7.93	**104.29**
airplane_011	**140.40**	18.54	34.81	35.21	8.94	2.77	37.85
airplane_012	**246.45**	18.55	10.03	3.91	0.99	1.66	10.62
airplane_013	39.53	18.11	39.09	**72.71**	16.16	3.55	60.62
airplane_015	**134.72**	17.68	24.40	14.01	3.29	5.68	53.43
airplane_016	**131.93**	17.80	12.79	5.33	1.25	2.45	16.60
big_2	**160.82**	33.84	37.76	15.02	3.79	7.57	68.54
youtube_dataset_2	83.23	18.76	32.80	25.91	6.12	9.07	**124.93**
youtube_dataset_3	**72.72**	18.58	16.57	7.16	1.60	3.19	22.47
mean	**108.94**	22.11	32.87	30.04	7.09	5.53	65.28

Similarly, when compared to *CN*, *STC* and *CT*, our approach stands out by more than 3 times faster than their average speeds. This high speed tracking of our approach makes it suitable for real-time application like UAV auto navigation.

6.5 Qualitative Evaluation

The image sequences may have to deal with different attributes (difficulties as mentioned in [30]), which needs to be tackled by each tracker to show promising result on the second performance parameter discussed earlier. These attributes are illumination variation, scale variation, occlusion, deformation, motion blur, fast motion, in-plane rotation, out-of-plane rotation, out-of-view, background clutters and low resolution [30]. Rigorous testing has been performed on all the seven trackers with the available 14 datasets and results of some of these problems are discussed in this section.

6.5.1 Snapshots of Tracking Result

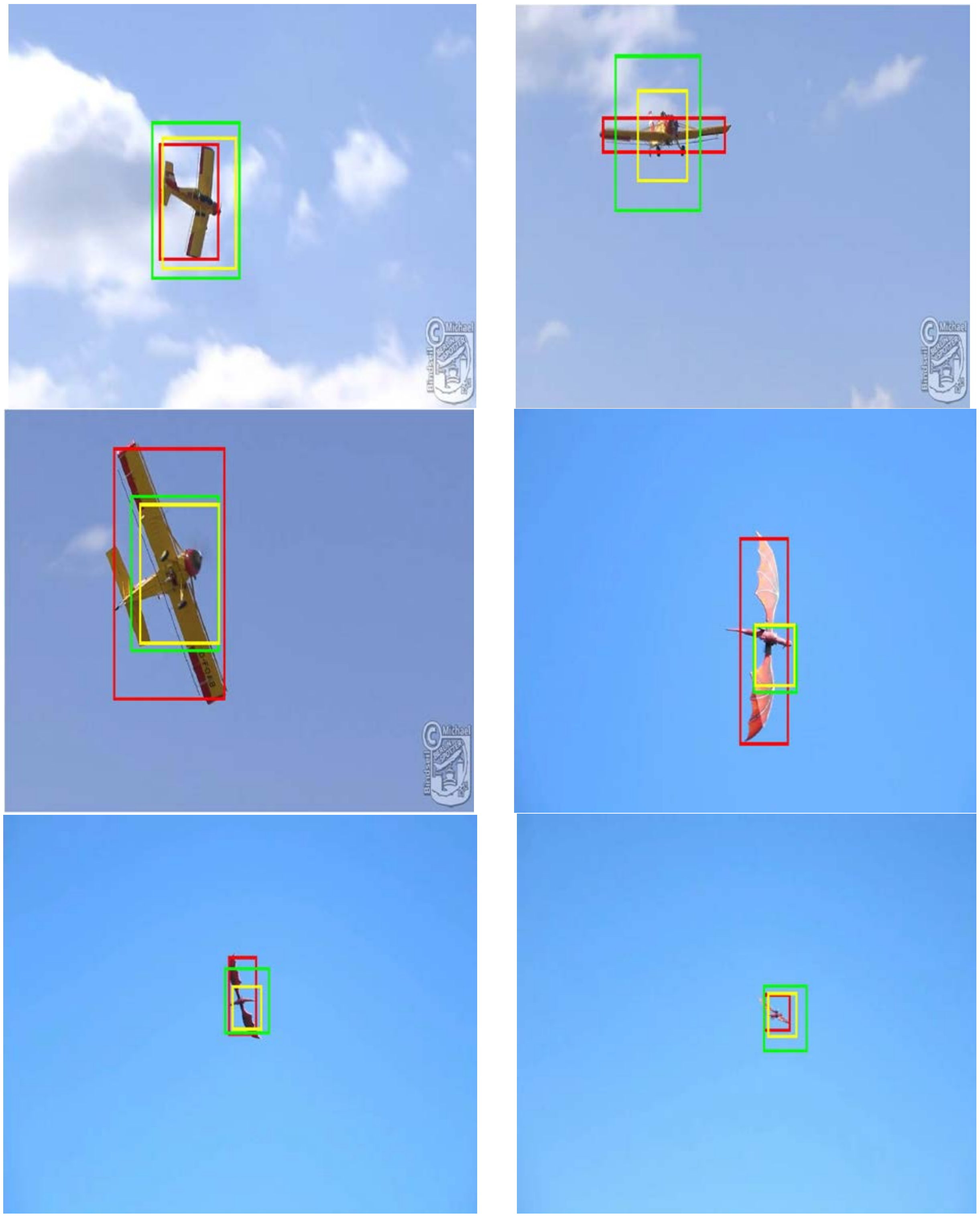


Fig 6.5: Tracking result with bounding box around the objects for shape, size and illumination variation. Color code: *red* – Proposed method, *yellow* – SAMF, *green* – KCF

Fig 6.5 presents some of the screenshots, demonstrating the proposed tracker in action. All the frames from the 14 datasets with tracking result from our proposed tracker is demonstrated in detail in Appendix A for the interested readers to further inspect.

6.5.2 Scale Variation and Partial/Total Occlusion

It has been observed experimentally that, in the initial frames, all the trackers produce acceptable outputs. This is because the object does not vary its shape or size to a great extent in the initial frames.

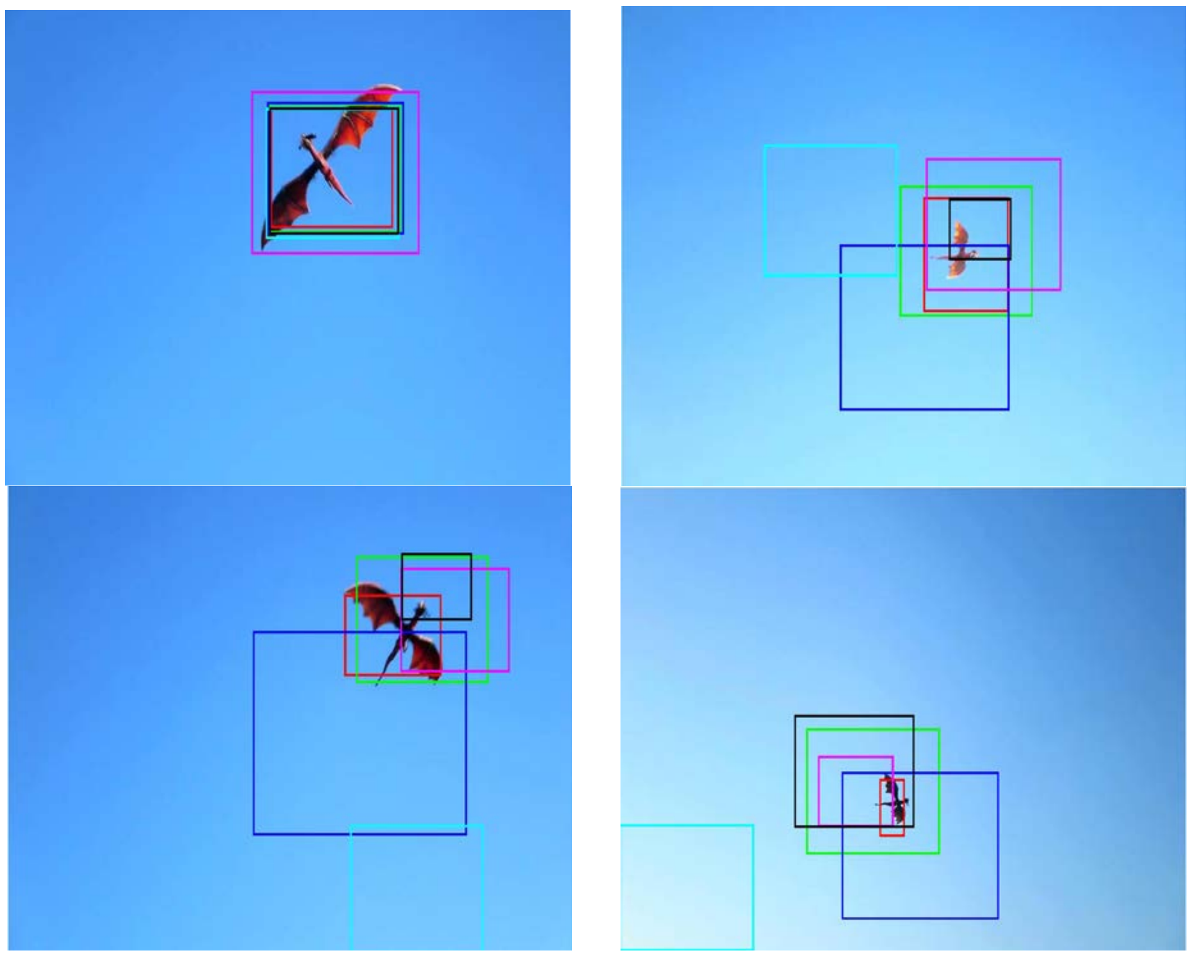


Fig 6.6: Tracking result with bounding box around the objects in *youtube_dataset_3* dataset for all the 7 trackers exhibiting scale variation. Color code same as Precision and Success Plots

The chosen frames from *youtube_dataset_3* in Fig 6.6 shows that the object size is varying in different frames. All the seven trackers assign bounding box around these frames. Some of the trackers have fixed bounding box and some are changing with the size of object. But none of the six competing trackers can assign bounding box of varying size, compatible with the object in all the frames. *CT*, *SAMF*, *KCF* and *STC* trackers fail miserably in assigning correct bounding boxes in Fig 6.6. The *red* bounding box, representing our algorithm, covers the entire object in all the 4 frames of Fig 6.6 with least background pixels inside the bounding box. It proves the robustness and adaptive nature of our tracking algorithm with respect to scale variation of the object.

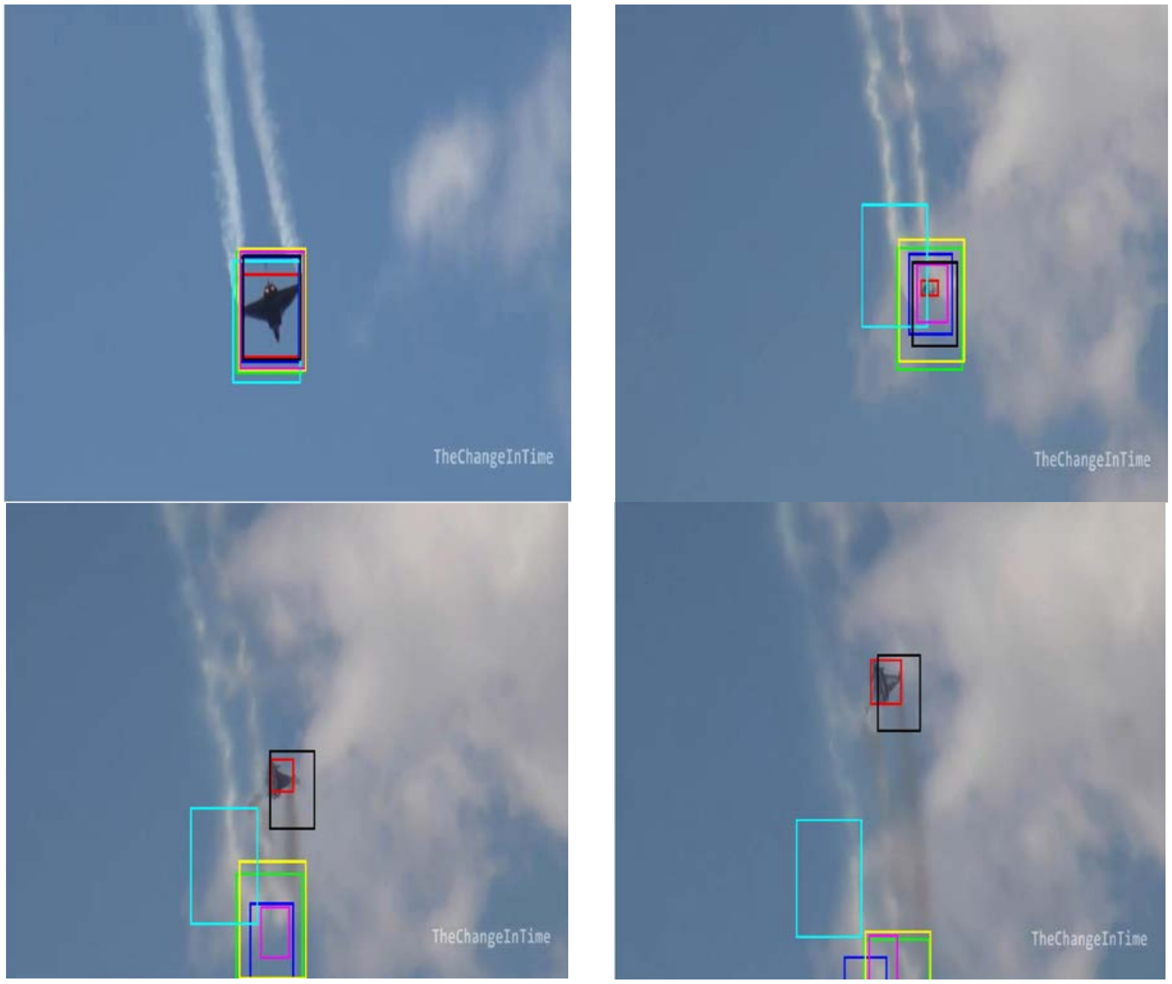


Fig 6.7: Tracking result with bounding box around the objects in *airplane_005* dataset for all the 7 trackers exhibiting partial occlusion. Color code same as Precision and Success Plots

Again, occlusion is a great problem in object tracking, where the object may become partially or totally invisible in a certain frame and reappear in the subsequent frames. Most of the trackers fail to assign correct bounding box around the reappearing object. In Fig 6.7, we have taken consecutive snapshots of an Aeroplane flying in between the clouds from *aeroplane_005* dataset. The Aeroplane hides in the cloud and reappears again causing occlusion. Only our tracker (*red* bounding box) and *STC* shows promising results to occlusion in these frames. Although the size of *STC* bounding box is less accurate than our method and the speed of *STC* is very less when compared to our method. Hence, our algorithm shows robustness to partial or full occlusion compared to other trackers.

6.5.3 Fast Motion and Illumination Variation

The distance between objects in two consecutive frames increase rapidly if the object is moving very fast. Some trackers fail to track in such cases, as the search area of tracking is very small and consist of the pixels corresponding to the nearby pixels of the object in the previous frame. Also, the camera sensor may not capture the images very clearly, when the object is moving very fast, leading to motion blurs. In case of auto navigation application of UAV, we will be expecting flying objects to track and thus, fast moving object is inevitable. Fig 6.8 presents a fast flying Aeroplane in consecutive frames from *airplane_001* dataset. We can see that almost all the other six competing trackers miss the target, whereas a *red* bounding box (representing our algorithm), successfully tracks the object in all the frames with scale variation. Hence, we can conclude that our algorithm is not vulnerable to fast motion.

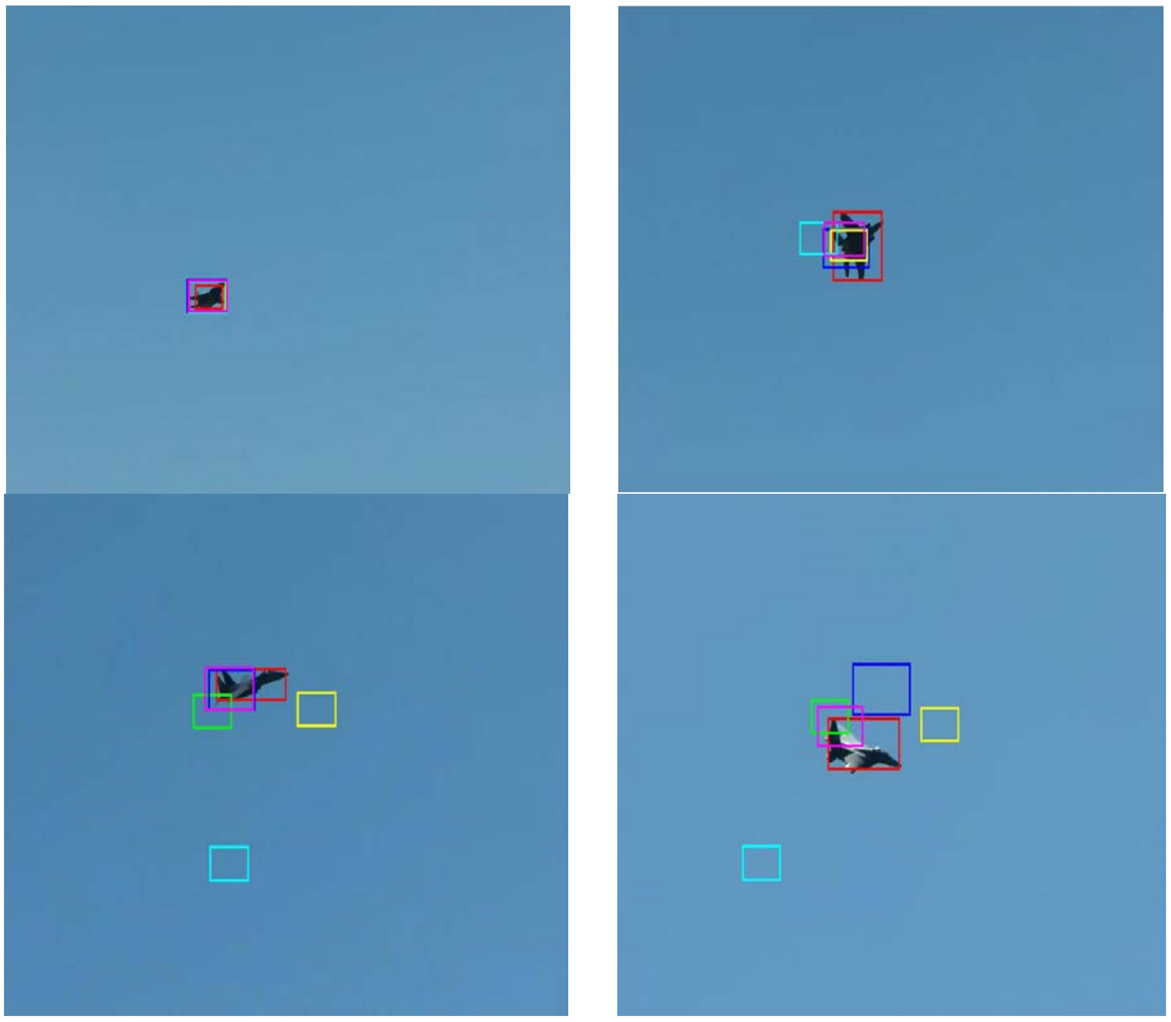


Fig 6.8: Tracking result with bounding box around the objects in *airplane_001* dataset for all the 7 trackers exhibiting fast motion. Color code same as Precision and Success Plots

Illumination variation is very common in photography. The light reflecting from the object may not be consistent in all the frames. The illumination variation is very sudden, but the tracker is trained over a certain number of frames in discriminative online tracking. Thus, the training may not consider this sudden change and fail miserably in the upcoming frames while tracking. Fig 6.8 presents one such scenario, where light starts reflecting from a flying Aeroplane in dataset *aeroplane_006*. *CT* fails miserably with illumination variation. All the other competing trackers successfully track the object, but the bounding box can only cover a part of the object now, for *CN*, *KCF* and *SAMF*. The bounding box of *DSST* becomes very small with illumination variation.

Our redetection scheme improves the tracker training, resulting in better performance with detection in the subsequent frames – which is quite evident in Fig 6.9 showing *red* bounding box with no object pixel missing.

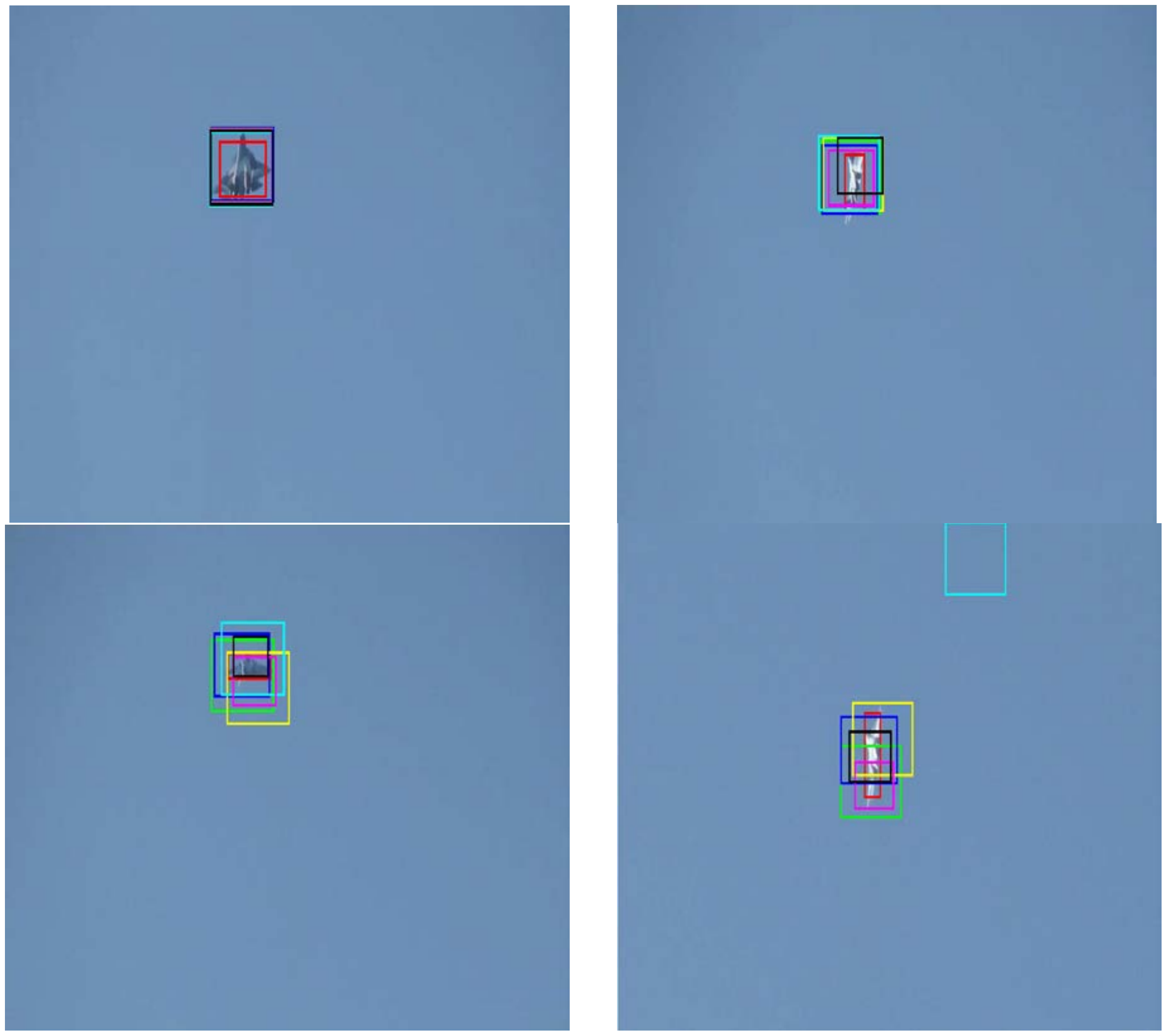


Fig 6.9: Tracking result with bounding box around the objects in *airplane_006* dataset for all the 7 trackers exhibiting illumination variation. Color code same as Precision and Success Plots

6.5.4 Rotation Dynamics Robustness

We would be expecting in-plane and out-of-plane rotations of the flying object while tracking it, along with translation. It is one of the most challenging tracking problem where majority of *state-of-the-art* trackers exhibits poor performance. Fig 6.10 shows in-plane rotation of a flying

Aeroplane from *Aircraft* dataset. We can see that *CT*, *DSST* and *CN* cannot even keep any part of the object during in-plane rotation. The bounding box of *DSST* becomes very small too. *KCF*, *STC* and *SAMF* partially tracks the object (*green* and *blue* bounding box cannot wrap around the entire object) in these frames. On the other hand, our tracker shows promising result by bounding the objects in *red* with variation of size, as well as keeping all the foreground pixels within the boundary box.

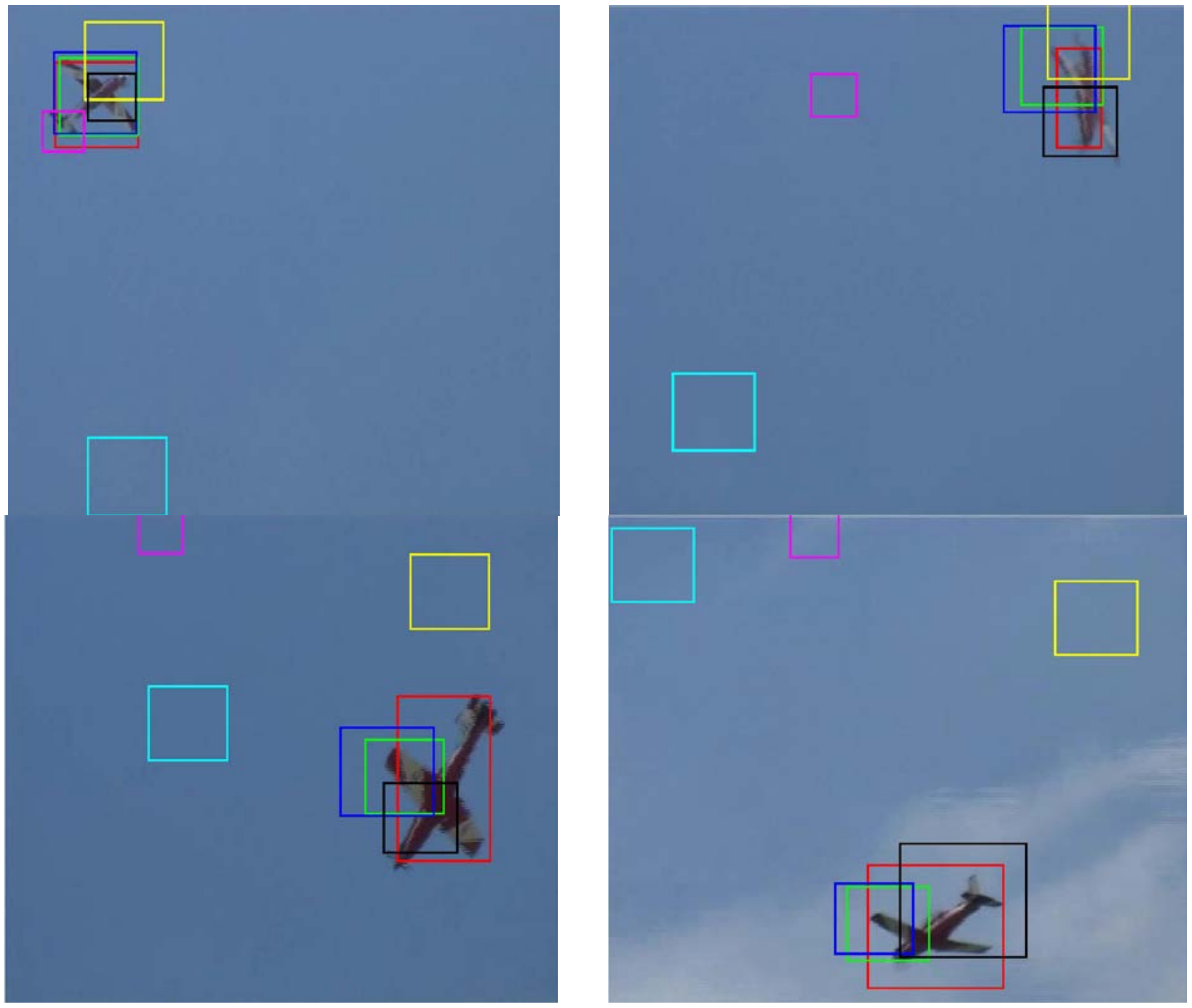


Fig 6.10: Tracking result with bounding boxes around the objects in *Aircraft* dataset for all the 7 trackers exhibiting in-plane rotation. Color code same as Precision and Success Plots

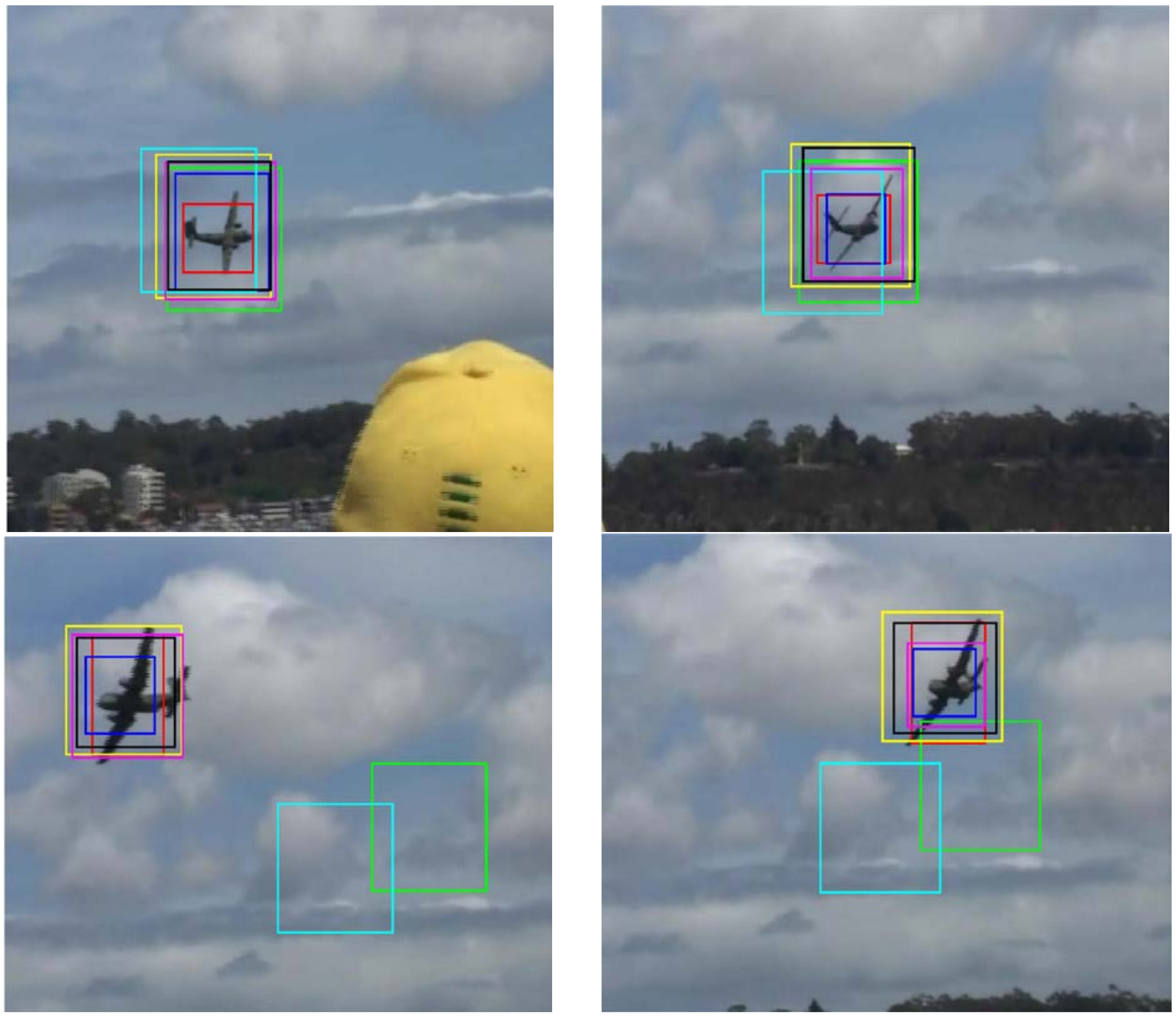


Fig 6.11: Tracking result with bounding boxes around the objects in *big_2* dataset for all the 7 trackers exhibiting out-of-plane rotation. Color code same as Precision and Success Plots

The out-of-plane rotation effect on different trackers is depicted in Fig 6.11 with *big_2* dataset. *CT* fails to tackle both in-plane and out-of-plane rotation dynamics. *KCF* gives decent performance for the in-plane rotation, but miserably fails for out-of-plane rotation. Again reduced *pink* bounding box is a problem for *DSST* tracker. *CN* and *STC* wraps the object with out-of-plane rotation, but distinctive amount of background pixels also comes in the bounding boxes and this may affect the tracker training leading to failure in the subsequent frames. It is clear that our tracker gives good result with out-of-plane rotation dynamics as well.

After extensive experimentation we found that our tracker exhibited promising result with almost all of the attributes mentioned in [30], unlike the six other competing trackers. The most distinctive results are shown in this chapter and some more results are also shown in the appendices.

Chapter 7

Conclusion and Future Work

7.1 Summary

In this work, we have defined a new problem in the computer vision field with object detection and tracking to guide an unmanned aerial vehicle (UAV) in its navigation. Extensive research has been performed to track down all the recent state-of-the-art object trackers and detectors and the most relevant ones were studied with good care. Even though the existing methods showed great results with some of the datasets (or in some favorable conditions), these approaches were inept to function as expected for the object detection and tracking from a forward-looking camera in a flying UAV. For some methods, either the speed was too low to perform in a real-time application or the tracker would behave abnormally in fast motion of the camera or in presence of illumination variation.

The correlation filter based tracker (KCF) from [3] showed good potential, as it was training the classifier in the Fourier domain with Circulant matrix, kernelized representation – resulting in high speed. After experimentation, we found that this tracker would fail miserably for scale variation or motion blur (due to fast relative motion between camera and object), out-of-plane rotation or partial/total occlusion and partially track object in presence of illumination variation. Another problem of manually labelling the bounding box in the first frame made it uncomfortable for auto navigation of UAV.

A minimum barrier distance transform based detection (MBD) was included in the tracker to provide a bounding box around the object from the first frame. Also, an adaptive redetection technique was proposed to tackle with the failures and difficulties of the KCF. The adaptive bounding box around the objects with different shape and size was a notable addition. A new post-processing method was added to the boost the quality of the saliency map generated by the MBD detector. The proposed new algorithm was implemented in OpenCV with C++ and extensively tested with relevant challenging datasets.

The most related other trackers were also extensively experimented to compare with our proposed method. After qualitative and quantitative evaluations on challenging datasets (related to UAV navigation application), we found that our proposed method stands out among all the other trackers in terms of speed. In terms of accuracy, it obtained best result for 11 out of 14 challenging datasets, second best result for most of the other datasets and a best average result. All the other methods stopped immaturely in one or more of the datasets. But the proposed method successfully assigned bounding box around objects in all the 14 challenging datasets. Hence, the proposed method was successful in demonstrating desired result and better performance than all other state-of-the-art approaches compared in this work.

7.2 Future Work

The vulnerability of the proposed method is that it fails if the object detection fails in the first frame or any of the other frames while redetecting. While experimenting, we tried to break the proposed method and came up with two situations where it fails. They are:

- In presence of more than one object in the path of the UAV, the MBD detector fails to create a correct saliency map, resulting in failure of the proposed tracker. More work needs to be done to make the detector adaptive to multiple saliency map generation.
- We also found that if the object is too big or too small, the MBD detector do not provide satisfactory result. In case of too big object, the proposed method gets confused to assign

a correct bounding box as the foreground covers more pixels than the background now, which was not likely.

The proposed tracker can also be used for other applications and should not be limited to only one application of UAV navigation. This is true because while testing the benchmark datasets from [30], it was found that the proposed method performs quite effectively with many other applications. Some of them are listed below:

1. Surveillance
2. Face detection and tracking
3. Tracking in movie graphics

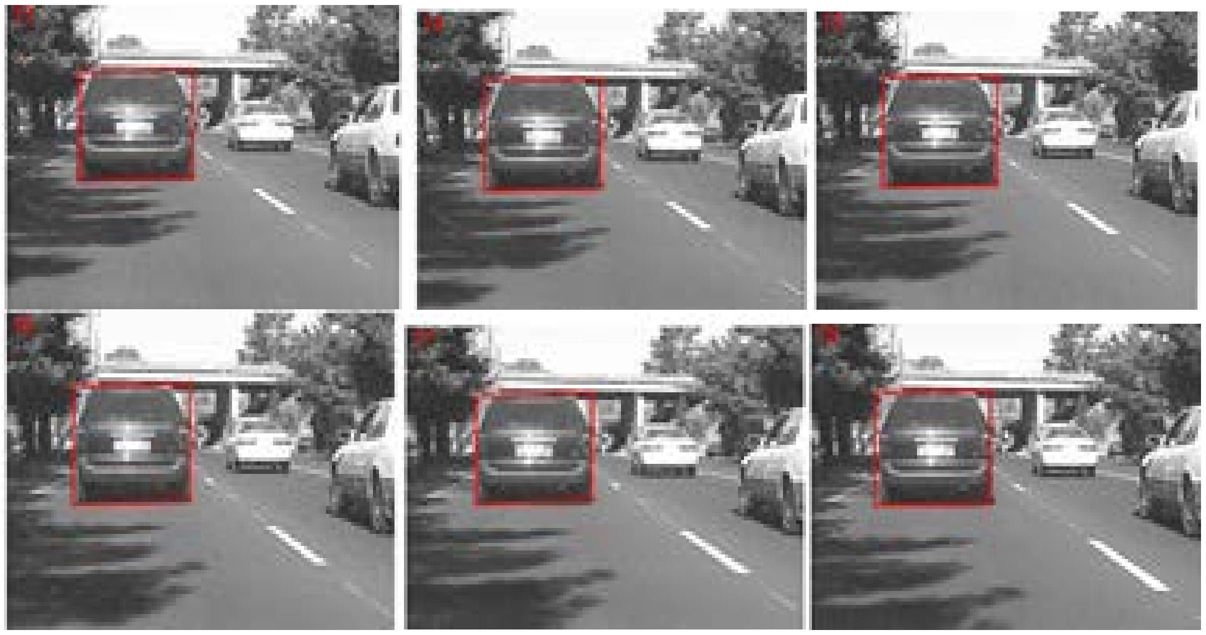


Fig 7.1: Surveillance operation possible with proposed tracker in *car4* dataset [30]

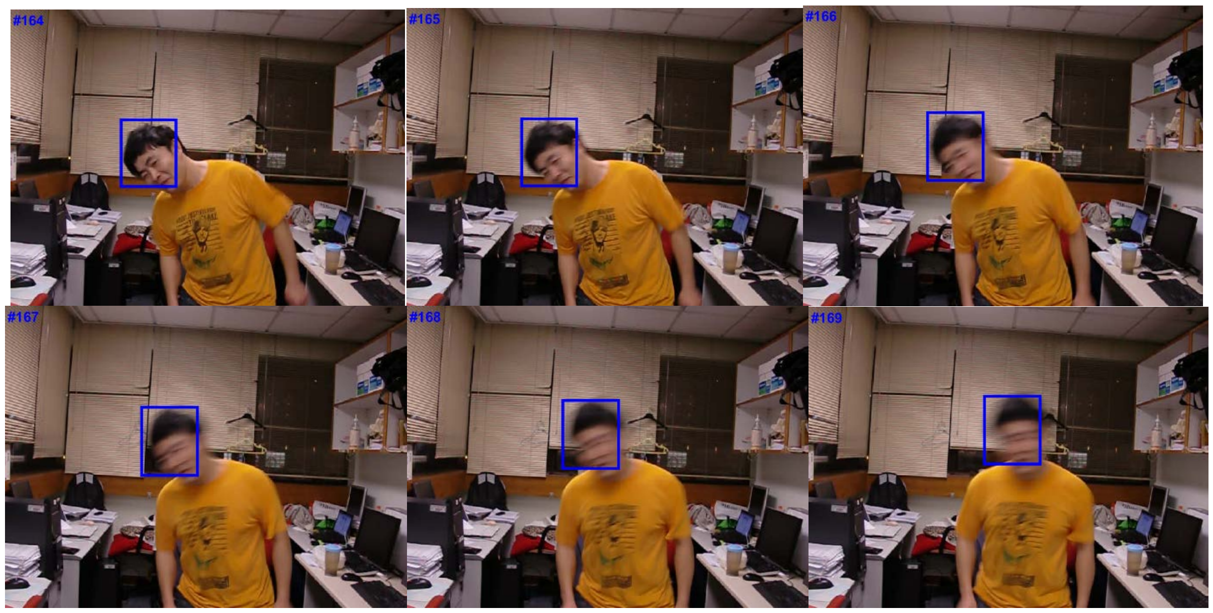


Fig 7.2: Face detection application possible with proposed tracker in *shaking2* dataset [30]

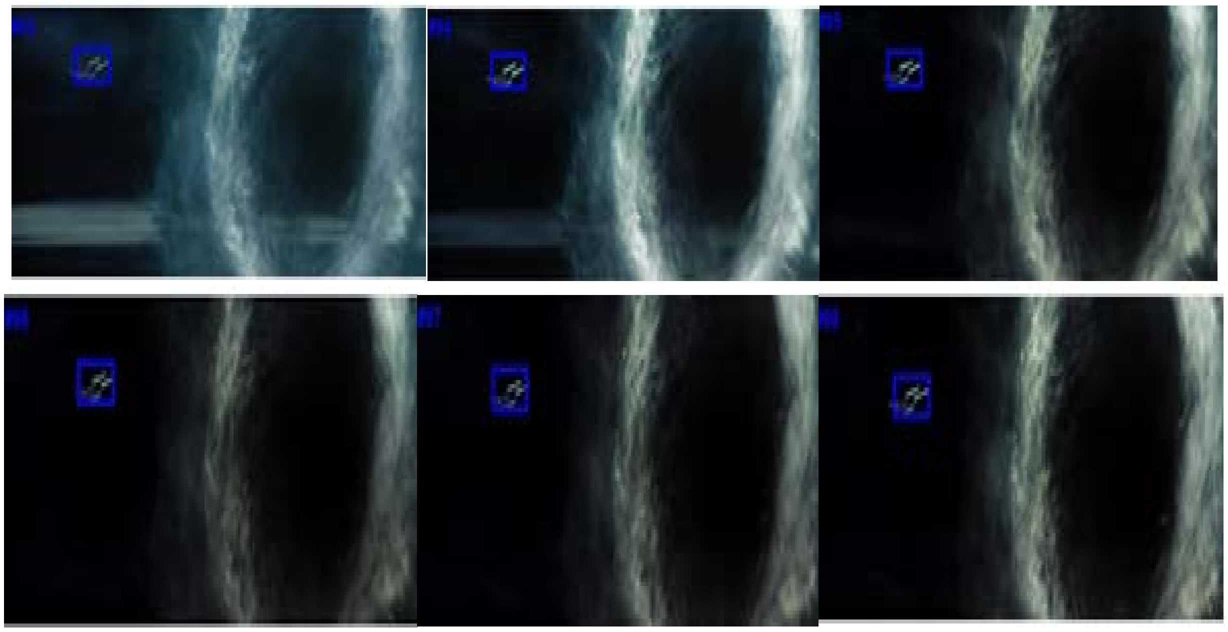


Fig 7.3: Tracking space shuttle from *startrek* dataset [30] – movie graphics tracking application

References

[1] A. Smeulders, D. Chu, R. Cucchiara, S. Calderara, A. Dehghan, and M. Shah, "Visual tracking: an experimental survey," *TPAMI*, 2013.

[2] Y. Li and J. Zhu, "A scale adaptive kernel correlation filter tracker with feature integration," in *European Conference on Computer Vision*. Springer, 2014, pp. 254–265.

[3] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, "High-speed tracking with kernelized correlation filters," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 37, no. 3, pp. 583–596, 2015.

[4] L. Itti and C. Koch, "Computational modelling of visual attention," *Nature reviews neuroscience*, vol. 2, no. 3, pp. 194–203, 2001.

[5] H. Cholakkal, D. Rajan, and J. Johnson, "Top-down saliency with locality-constrained contextual sparse coding." *BMVC*, 2015.

[6] J. Yang and M.-H. Yang, "Top-down visual saliency via joint crf and dictionary learning," in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 2296–2303.

[7] Y. Wei, F. Wen, W. Zhu, and J. Sun, "Geodesic saliency using background priors," in *European Conference on Computer Vision*. Springer, 2012, pp. 29–42.

[8] J. Zhang, S. Sclaroff, Z. Lin, X. Shen, B. Price, and R. Mech, "Minimum barrier salient object detection at 80 fps," in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 1404–1412.

[9] K. C. Ciesielski, R. Strand, F. Malmberg, and P. K. Saha, "Efficient algorithm for finding the exact minimum barrier distance," *Computer Vision and Image Understanding*, vol. 123, pp. 53–64, 2014.

[10] R. Strand, K. C. Ciesielski, F. Malmberg, and P. K. Saha, "The minimum barrier distance," *Computer Vision and Image Understanding*, vol. 117, no. 4, pp. 429–437, 2013.

[11] B. Babenko, M.-H. Yang, and S. Belongie, “Visual tracking with online multiple instance learning,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 983–990.

[12] K. Zhang, L. Zhang, and M.-H. Yang, “Real-time compressive tracking,” in *European Conference on Computer Vision*. Springer, 2012, pp. 864–877.

[13] S. Hare *et al*., "Struck: Structured Output Tracking with Kernels," in *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 10, pp. 2096-2109, Oct. 1 2016. doi: 10.1109/TPAMI.2015.2509974

[14] Z. Kalal, K. Mikolajczyk, and J. Matas, “Tracking-learning-detection,” *IEEE transactions on pattern analysis and machine intelligence*, vol. 34, no. 7, pp. 1409–1422, 2012.

[15] A. Jordan, “On discriminative vs. generative classifiers: A comparison of logistic regression and naive bayes,” *Advances in neural information processing systems*, vol. 14, p. 841, 2002.

[16] M. Andriluka, S. Roth, and B. Schiele, “People-tracking-by-detection and people-detection-by-tracking,” in *Computer Vision and Pattern Recognition, 2008. CVPR 2008. IEEE Conference on*. IEEE, 2008, pp. 1–8.

[17] V. Mahadevan and N. Vasconcelos, “Saliency-based discriminant tracking,” in *Computer Vision and Pattern Recognition, 2009. CVPR 2009. IEEE Conference on*. IEEE, 2009, pp. 1007–1013.

[18] A. Nussberger, H. Grabner, and L. Van Gool, “Robust aerial object tracking in high dynamic flight maneuvers,” *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. 2, no. 1, p. 1, 2015.

[19] A. Kocak, K. Cizmeciler, A. Erdem, and E. Erdem, “Top down saliency estimation via superpixel-based discriminative dictionaries.” in *BMVC*, 2014.

[20] S. He, R. W. Lau, W. Liu, Z. Huang, and Q. Yang, “Supercnn: A superpixelwise convolutional neural network for salient object detection,” *International Journal of Computer Vision*, vol. 115, no. 3, pp. 330–344, 2015.

[21] P. Wang, J. Wang, G. Zeng, J. Feng, H. Zha, and S. Li, “Salient object detection for searched web images via global saliency,” in *Computer Vision and Pattern Recognition (CVPR), 2012 IEEE Conference on*. IEEE, 2012, pp. 3194–3201.

[22] R. Zhao, W. Ouyang, H. Li, and X. Wang, “Saliency detection by multi-context deep learning,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 1265–1274.

[23] Y. Sui, G. Wang, Y. Tang, and L. Zhang, “Tracking completion,” *arXiv preprint arXiv:1608.08171*, 2016.

[24] W. Zhu, S. Liang, Y. Wei, and J. Sun, “Saliency optimization from robust background detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2014, pp. 2814–2821.

[25] J. Feng, Y. Wei, L. Tao, C. Zhang, and J. Sun, “Salient object detection by composition,” in 2011 *International Conference on Computer Vision*. IEEE, 2011, pp. 1028–1035.

[26] P. Siva, C. Russell, T. Xiang, and L. Agapito, “Looking beyond the image: Unsupervised learning for object saliency and detection,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 3238–3245.

[27] A. Borji, M.-M. Cheng, H. Jiang, and J. Li, “Salient object detection: A survey,” *arXiv preprint arXiv:1411.5878*, 2014.

[28] D. S. Bolme, J. R. Beveridge, B. A. Draper, and Y. M. Lui, “Visual object tracking using adaptive correlation filters,” in *Computer Vision and Pattern Recognition (CVPR), 2010 IEEE Conference on*. IEEE, 2010, pp. 2544–2550.

[29] J. F. Henriques, R. Caseiro, P. Martins, and J. Batista, “Exploiting the circulant structure of tracking-by-detection with kernels,” in *European conference on computer vision*. Springer, 2012, pp. 702–715.

[30] Y. Wu, J. Lim, and M.-H. Yang, “Online object tracking: A benchmark,” in *Proceedings of the IEEE conference on computer vision and pattern recognition*, 2013, pp. 2411–2418.

[31] Y. Sui, Z. Zhang, G. Wang, Y. Tang, and L. Zhang, “Real-time visual tracking: Promoting the robustness of correlation filter learning,” *arXiv preprint arXiv:1608.08173*, 2016.

[32] M. Danelljan, F. Shahbaz Khan, M. Felsberg, and J. Van de Weijer, “Adaptive color attributes for real-time visual tracking,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2014, pp. 1090–1097.

[33] M. Danelljan, G. H¨ager, F. Khan, and M. Felsberg, “Accurate scale estimation for robust visual tracking,” in *British Machine Vision Conference, Nottingham, September 1-5, 2014*. BMVA Press, 2014.

[34] T. Liu, G. Wang, and Q. Yang, “Real-time part-based visual tracking via adaptive correlation filters,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 4902–4912.

[35] C. Ma, X. Yang, C. Zhang, and M.-H. Yang, “Long-term correlation tracking,” in *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, 2015, pp. 5388–5396.

[36] S. Ren, K. He, R. Girshick, and J. Sun, “Faster r-cnn: Towards real-time object detection with region proposal networks,” in *Advances in neural information processing systems*, 2015, pp. 91–99.

[37] C. Ma, J.-B. Huang, X. Yang, and M.-H. Yang, “Hierarchical convolutional features for visual tracking,” in *Proceedings of the IEEE International Conference on Computer Vision*, 2015, pp. 3074–3082.

[38] H. Li, Y. Li, and F. Porikli, “Deeptrack: Learning discriminative feature representations online for robust visual tracking,” *IEEE Transactions on Image Processing*, vol. 25, no. 4, pp. 1834–1848, 2016.

[39] R. M. Gray, *Toeplitz and circulant matrices: A review*. now publishers inc, 2006.

[40] B. Scholkopf and A. J. Smola, *Learning with kernels: support vector machines, regularization, optimization, and beyond*. MIT press, 2001.

[41] R. Rifkin, G. Yeo, and T. Poggio, “Regularized least-squares classification,” *Nato Science Series Sub Series III Computer and Systems Sciences*, vol. 190, pp. 131–154, 2003.

[42] N. Otsu, “A threshold selection method from gray-level histograms,” *Automatica*, vol. 11, no. 285-296, pp. 23–27, 1975.

[43] A. Li, M. Lin, Y. Wu, M. Yang, and S. Yan, “NUS-PRO: A New Visual Tracking Challenge,” *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 38, no. 2, pp. 335–349, 2016.

[44] A. Rosenfeld, J. L. Pfaltz, Distance functions on digital pictures, *Pattern Recognition 1* (1968) 33-61.

[45] Y. Zhai, M. Shah (2006), Visual attention detection in video sequences using spatiotemporal cues, *MM '06 Proceedings of the 14th ACM international conference on Multimedia*. Pages 815-824

[46] Zhang, Kaihua, "Fast visual tracking via dense spatio-temporal context learning." in *European Conference on Computer Vision*. Springer International Publishing, 2014.

[47] Bharati, S., Nandi, S., Wu, Y., Sui, Y., Wang, G. (2016), “Fast and Robust Object Tracking with Adaptive Detection”, *The 28th IEEE International Conference on Tools with Artificial Intelligence (ICTAI)*, 2016.

Appendices

Appendix A

The screenshots showing bounding box around objects in some of the frames of 14 challenging datasets are presented below:

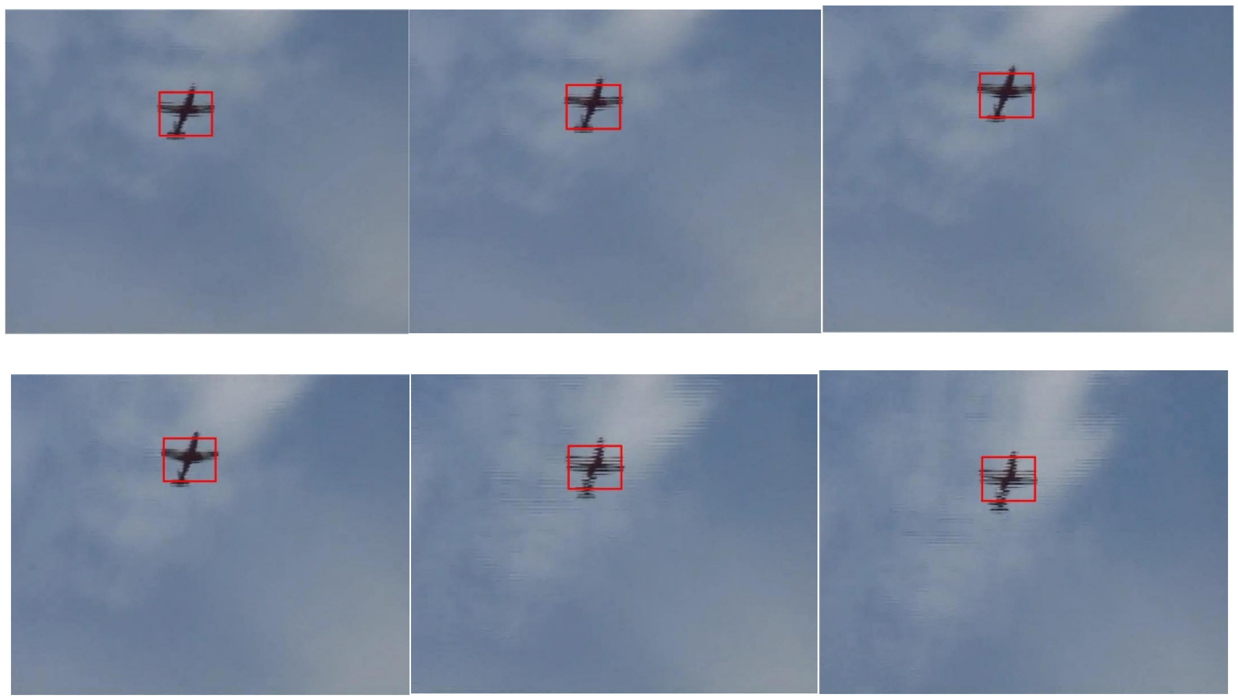


Fig A.1: Snapshots from *Aircraft* dataset showing tracking with the proposed tracker

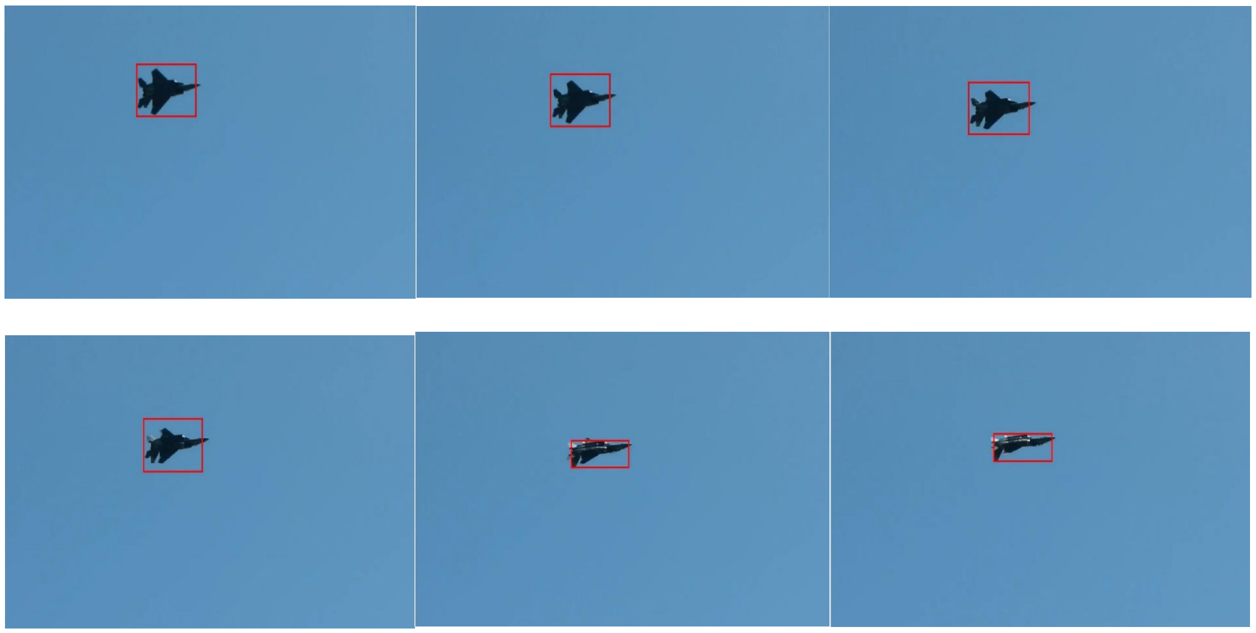


Fig A.2: Snapshots from *airplane_001* dataset showing tracking with the proposed tracker

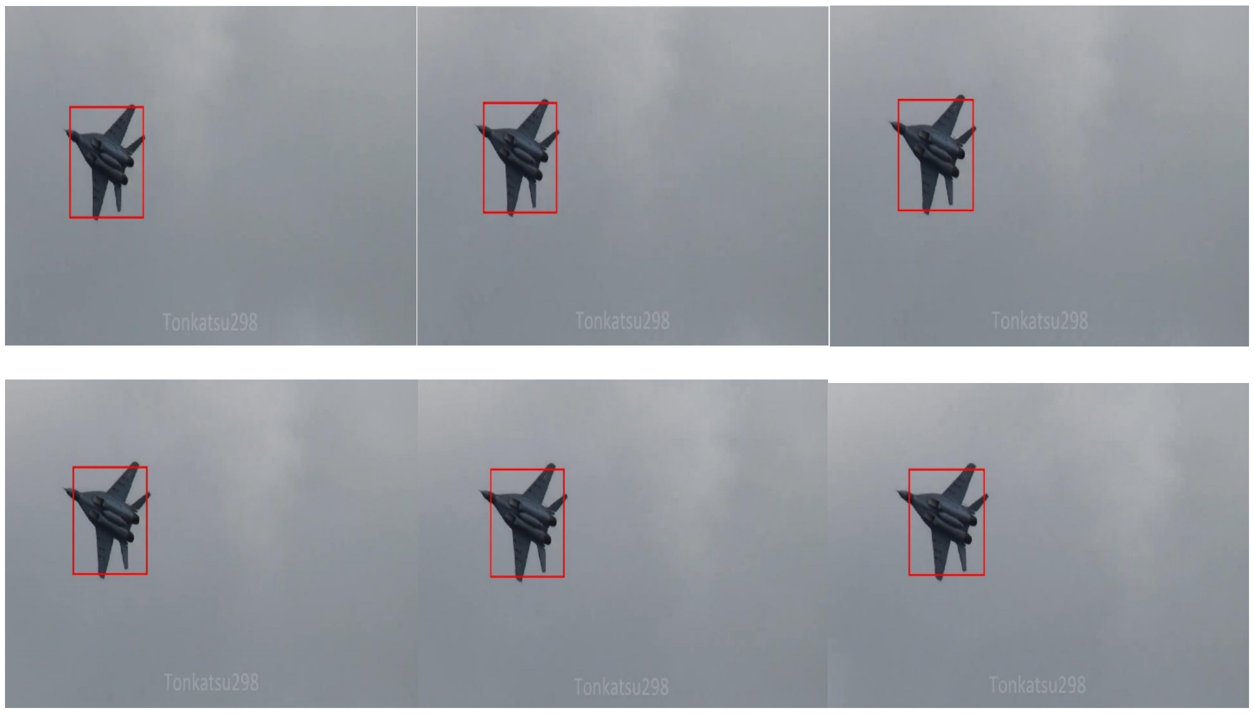


Fig A.3: Snapshots from *airplane_004* dataset showing tracking with the proposed tracker

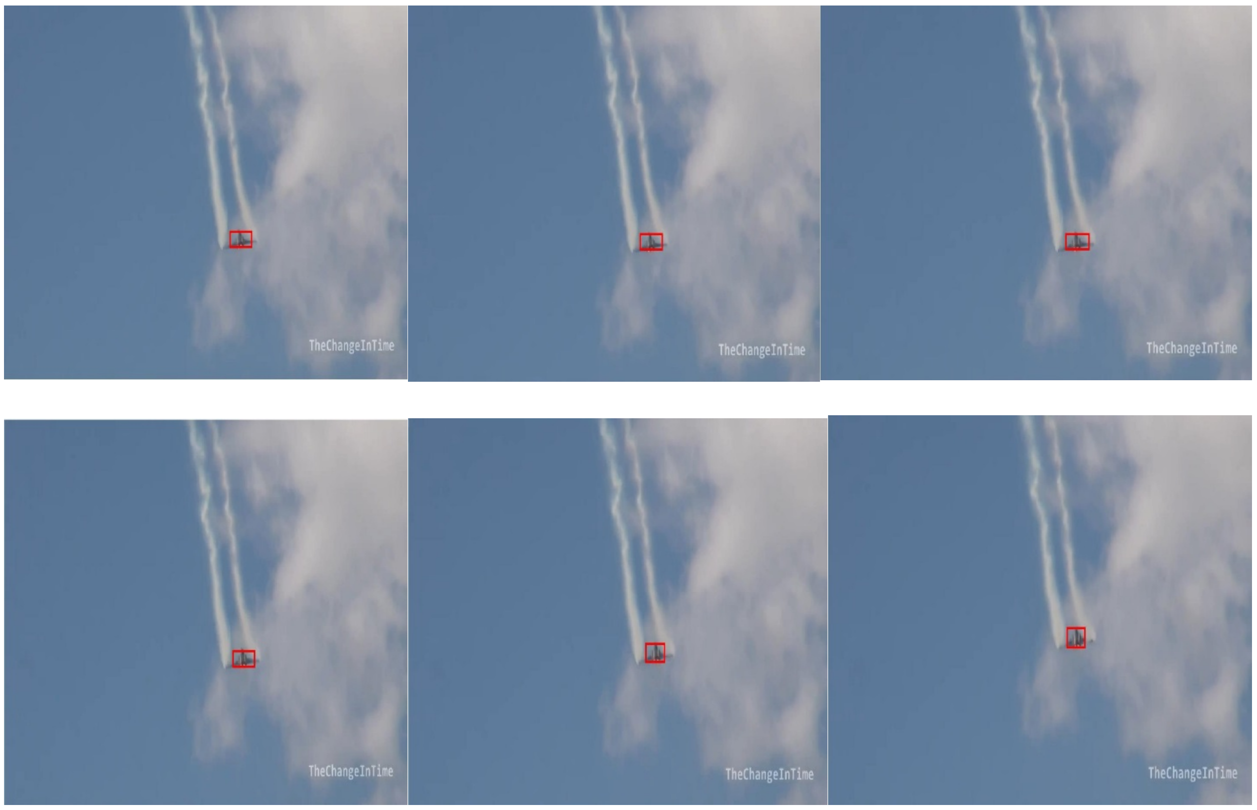


Fig A.4: Snapshots from *airplane_005* dataset showing tracking with the proposed tracker

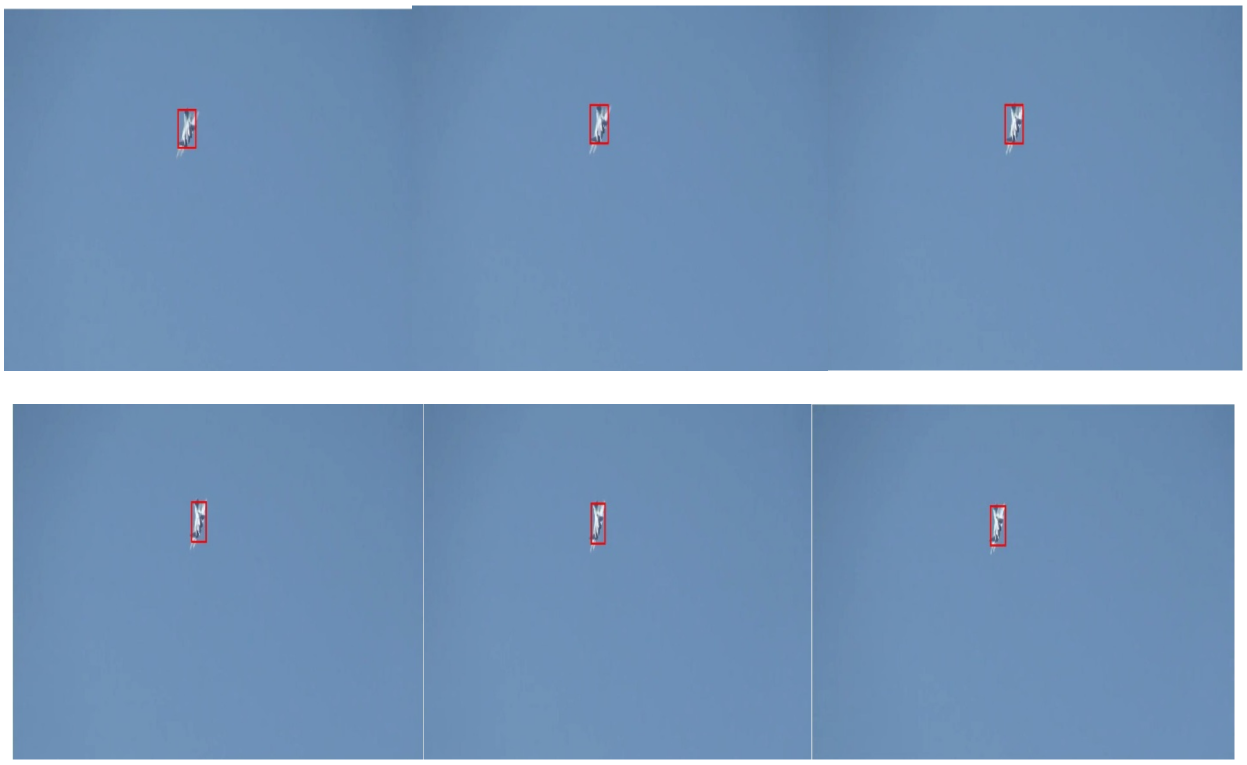


Fig A.5: Snapshots from *airplane_006* dataset showing tracking with the proposed tracker

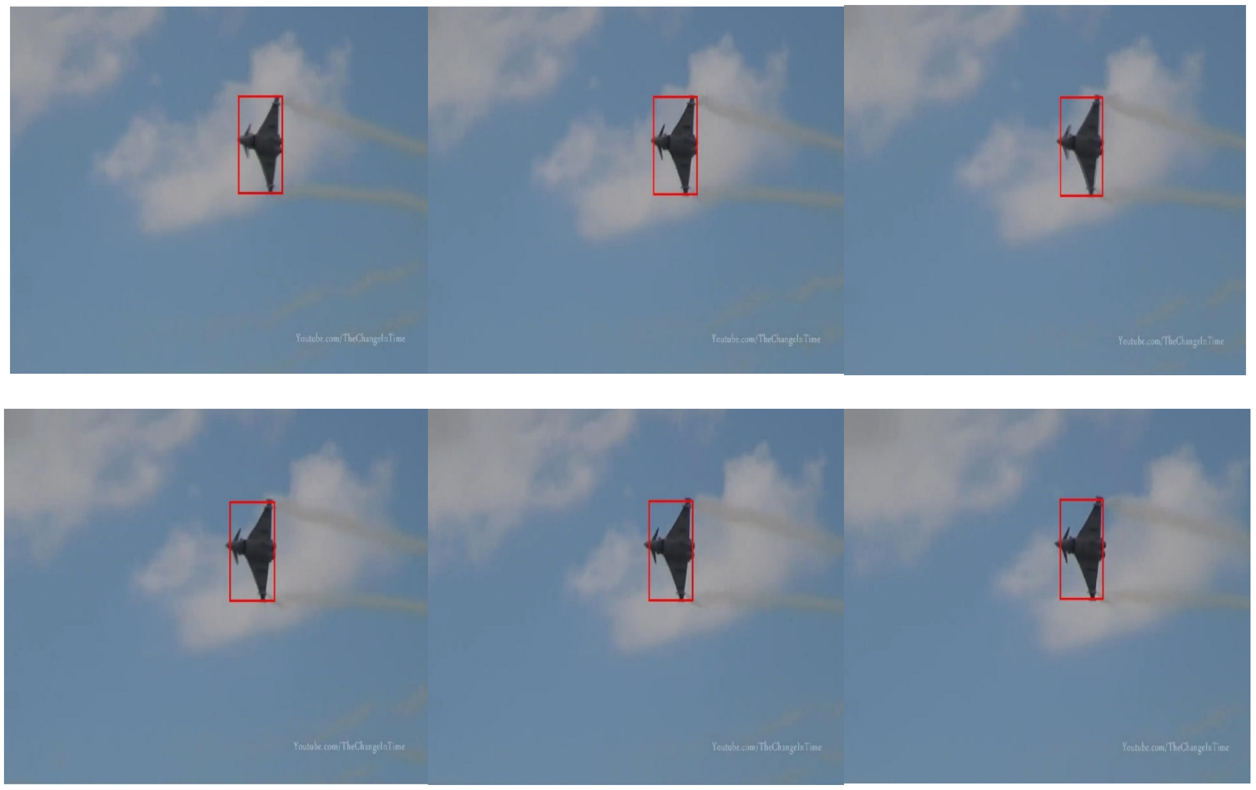


Fig A.6: Snapshots from *airplane_007* dataset showing tracking with the proposed tracker

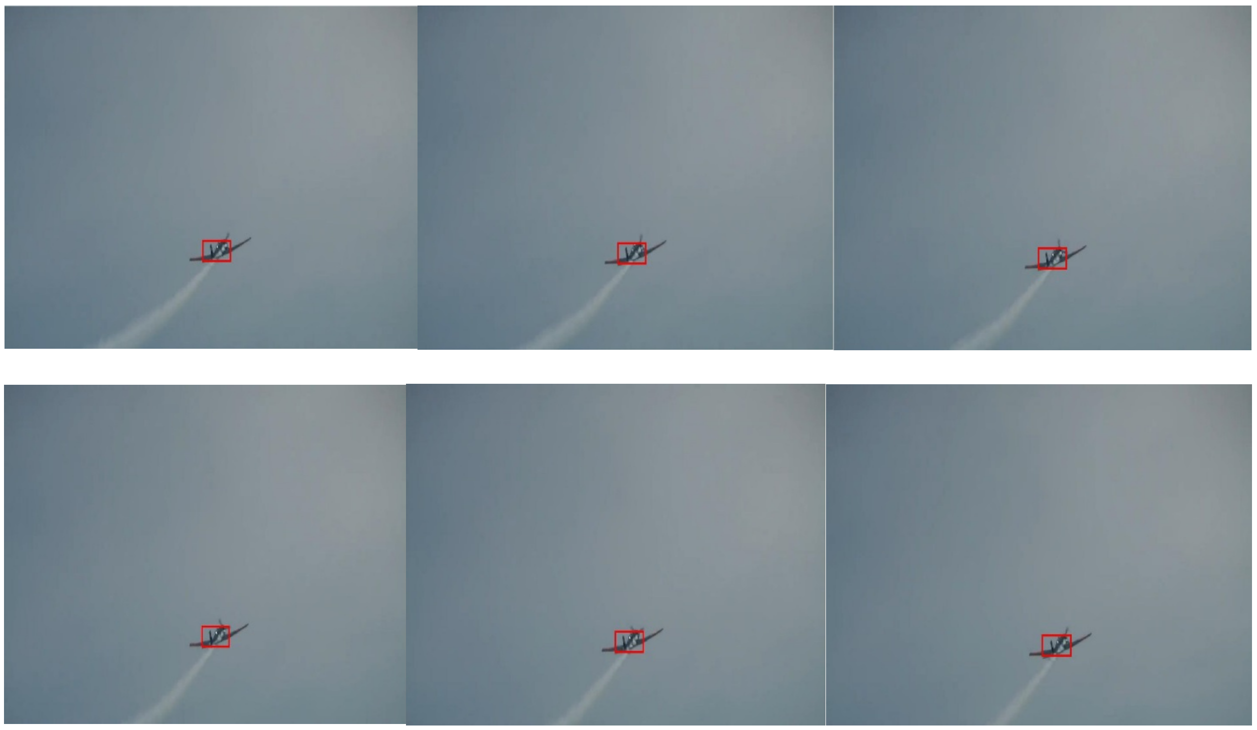


Fig A.7: Snapshots from *airplane_011* dataset showing tracking with the proposed tracker

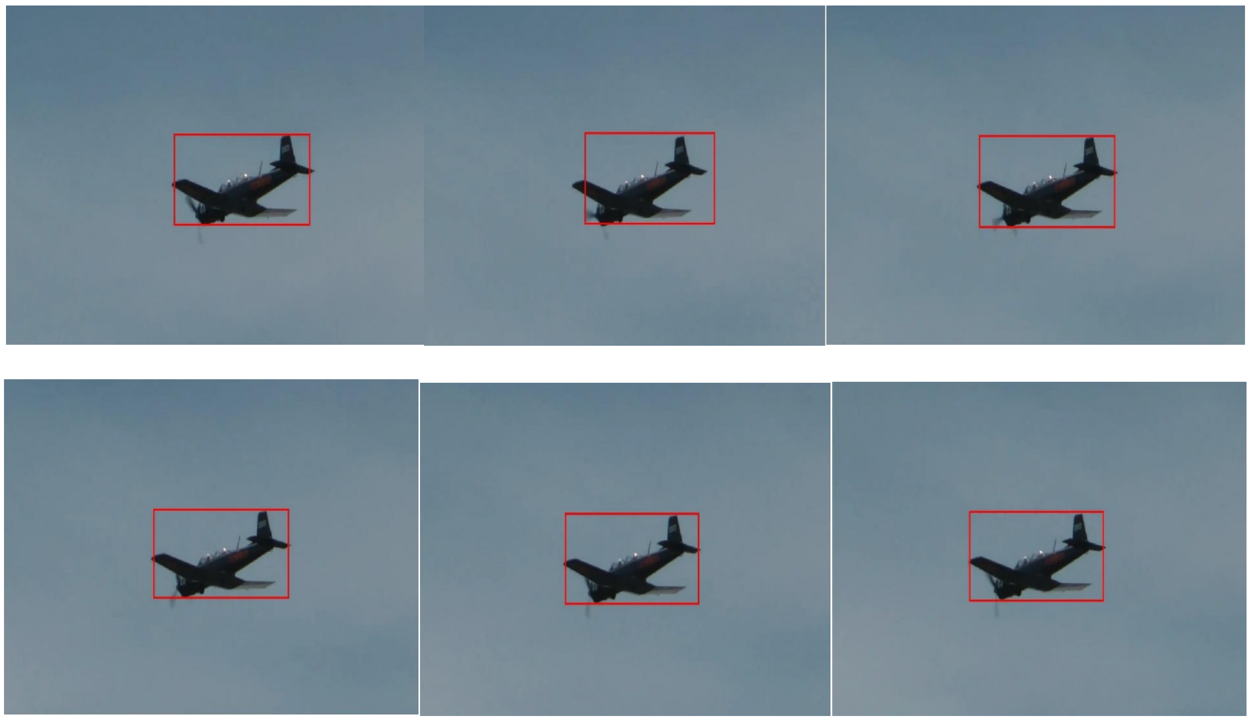


Fig A.8: Snapshots from *airplane_012* dataset showing tracking with the proposed tracker

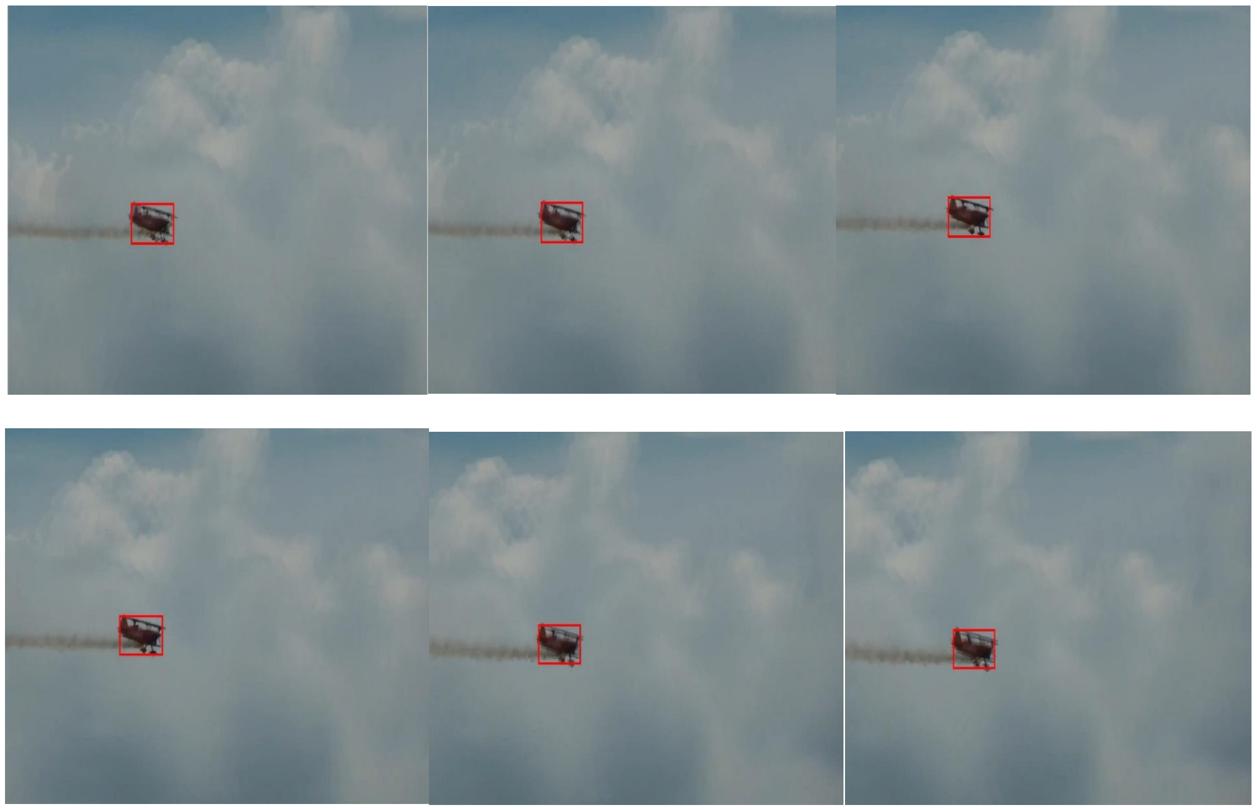


Fig A.9: Snapshots from *airplane_013* dataset showing tracking with the proposed tracker

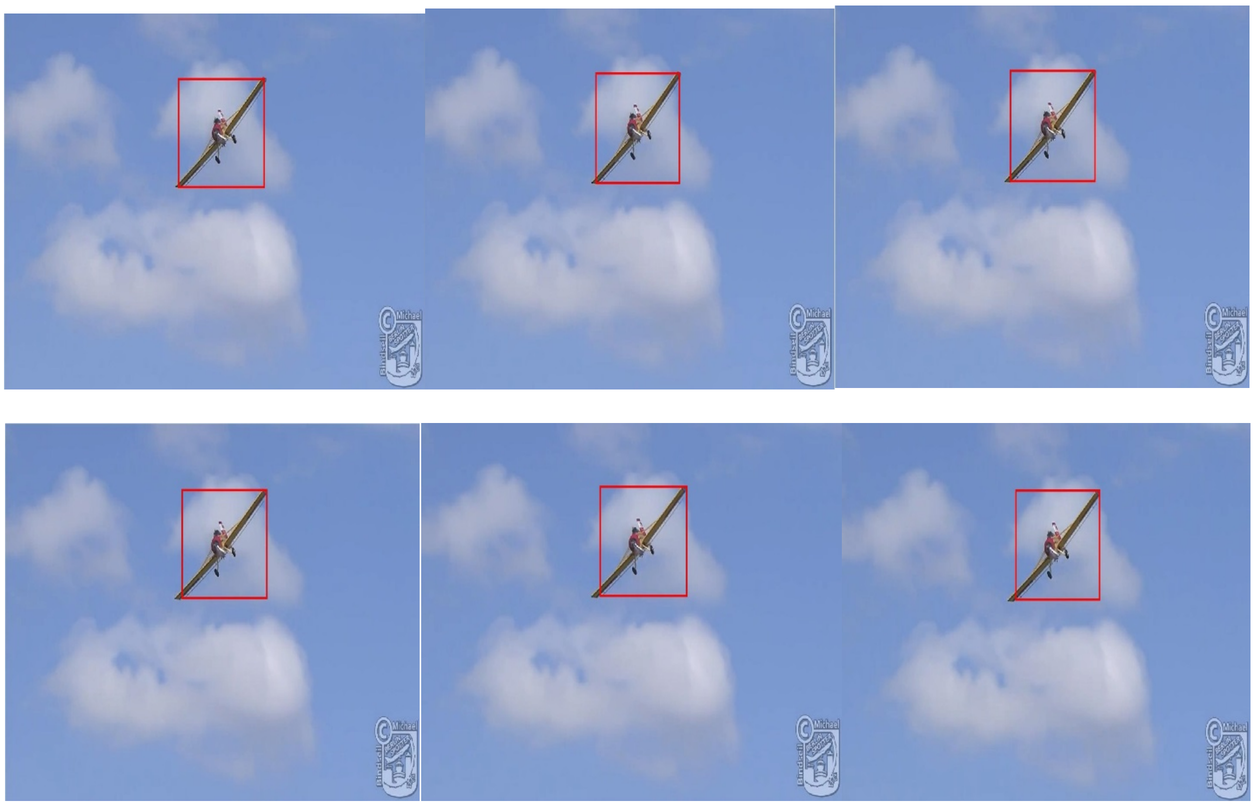


Fig A.10: Snapshots from *airplane_015* dataset showing tracking with the proposed tracker

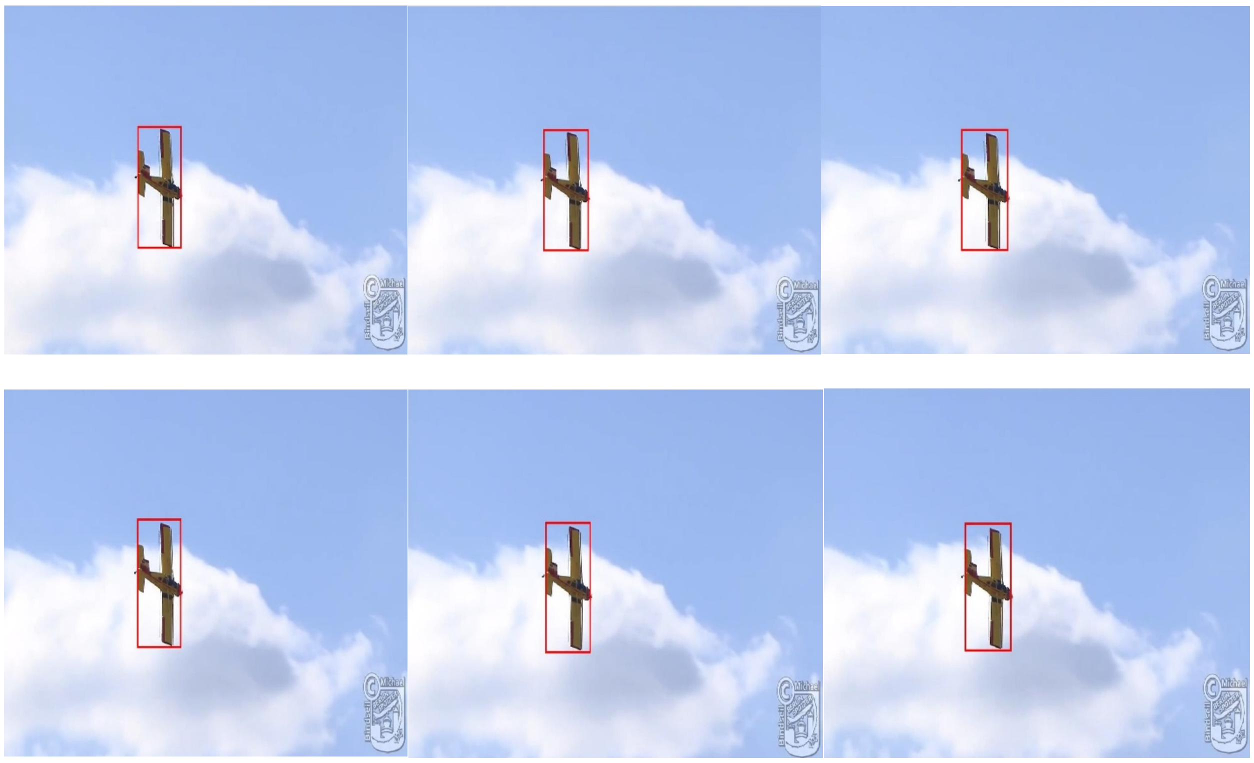


Fig A.11: Snapshots from *airplane_016* dataset showing tracking with the proposed tracker

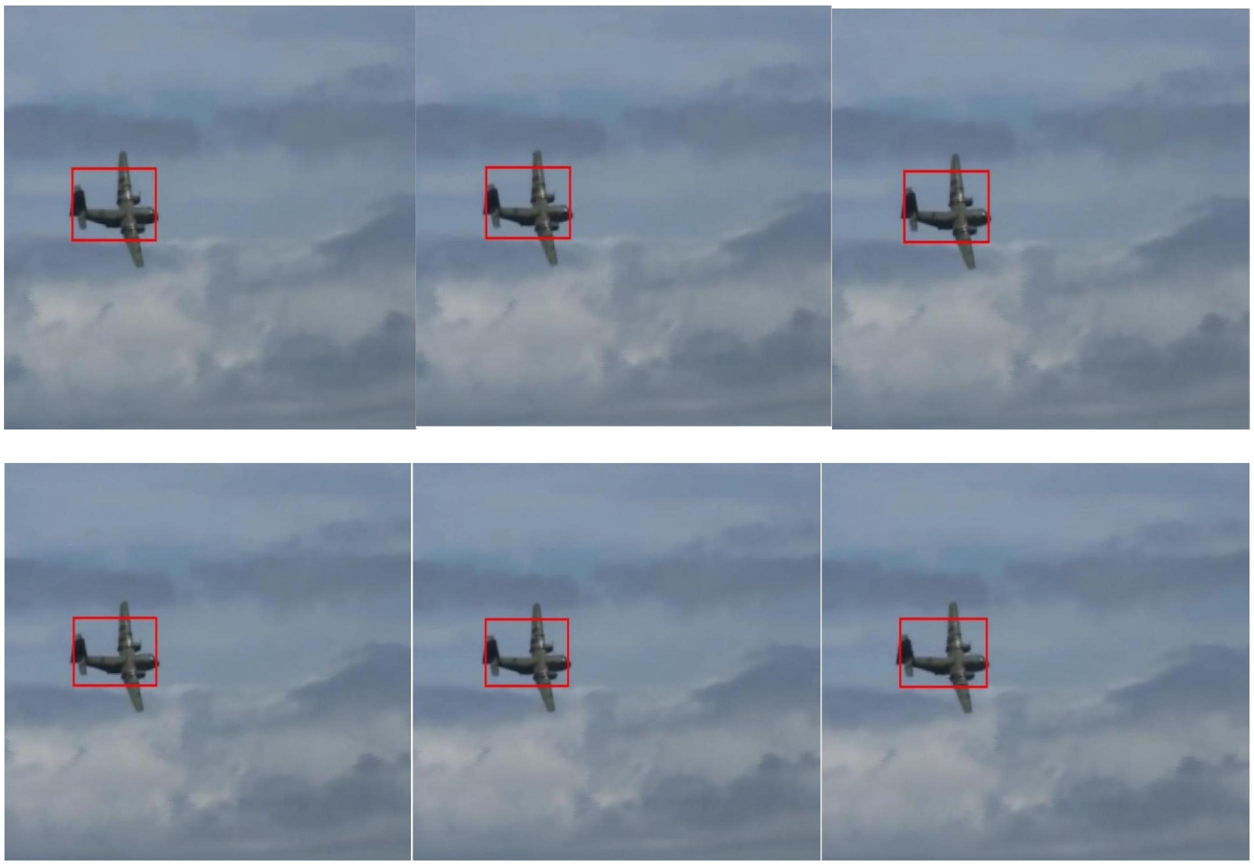


Fig A.12: Snapshots from *big_2* dataset showing tracking with the proposed tracker

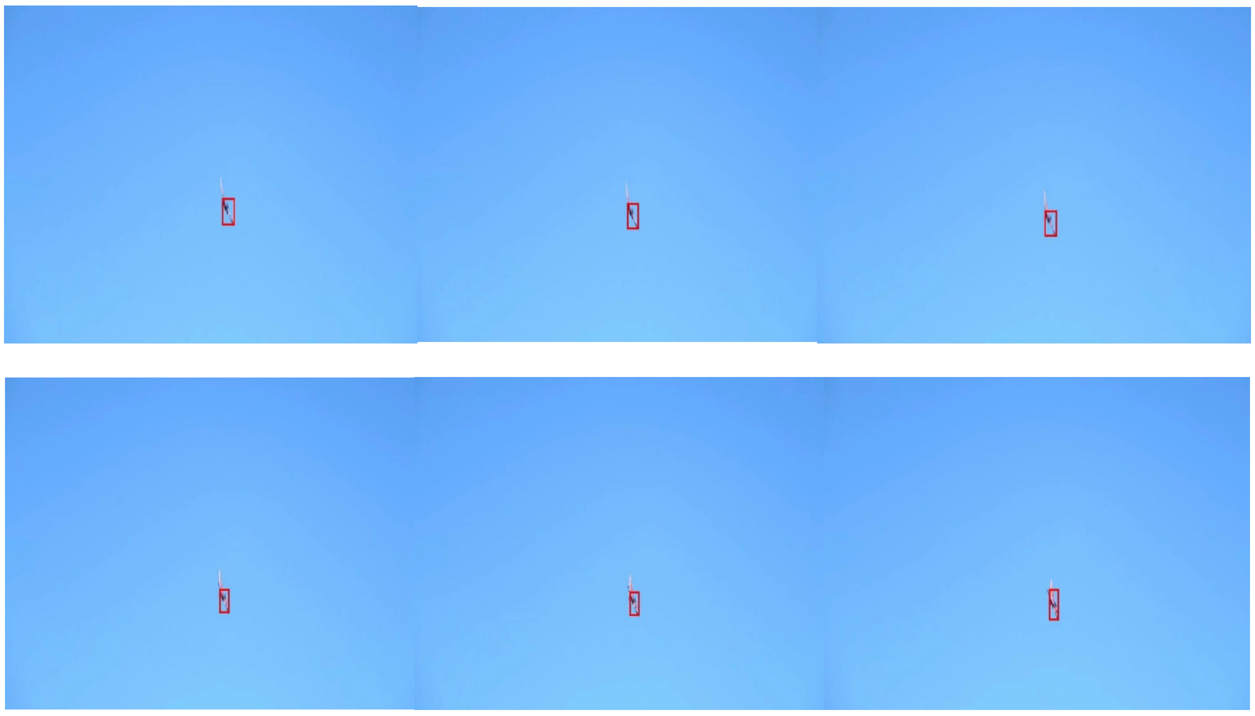


Fig A.13: Snapshots from *youtube_2* dataset showing tracking with the proposed tracker

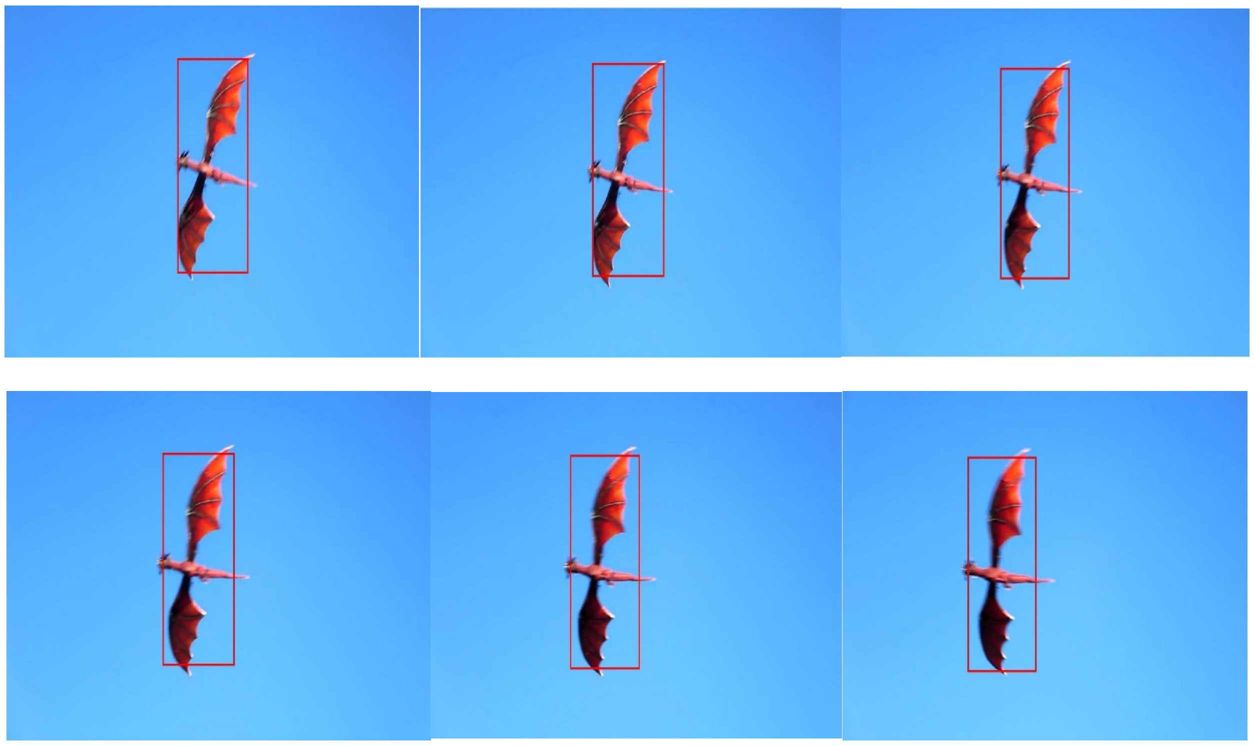


Fig A.14: Snapshots from *youtube_3* dataset showing tracking with the proposed tracker

Appendix B

The pseudocode for our proposed tracker is already presented in Table I in chapter 5. In this section the other MATLAB codes used for tabulating and plotting *OPE* and *TRE* Precision Rate and Success Rate are presented.

B.1 The following MATLAB code tabulates and plots *OPE* PR and SR:

```
% This creates a text file for tabulating the Precision Rate and Success
% Rate for OPE of each dataset for each tracker
% Requirements: All the OPEs and Ground Truth should be in the same folder
% OPE files should follow the format : datasetName_OPE_trackerName.txt
% Ground truth should follow the format: datasetName.txt
% Output file created is : avg_OPE_PR_SR_CLE.txt
% Precision and Success Rates of OPE

clear all;
close('all');
clc;

m1=[];s1=[];cl=[];
```

```
seq =
{'airplane\_001';'airplane\_004';'airplane\_005';'airplane\_006';'airplane\_0
07';'airplane\_011'; ...
       'airplane\_012';'airplane\_013';
'airplane\_015';'airplane\_016';'big\_2';'youtube\_2';'youtube\_3';'Aircraft'
};

for i = 1:7
    %read gt files
    gt1 = fopen('airplane_001.txt','r');
    gt2 = fopen('airplane_004.txt','r');
    gt3 = fopen('airplane_005.txt','r');
    gt4 = fopen('airplane_006.txt','r');
    gt5 = fopen('airplane_007.txt','r');
    gt6 = fopen('airplane_011.txt','r');
    gt7 = fopen('airplane_012.txt','r');
    gt8 = fopen('airplane_013.txt','r');
    gt9 = fopen('airplane_015.txt','r');
    gt10 = fopen('airplane_016.txt','r');
    gt11 = fopen('big_2.txt','r');
    gt12 = fopen('youtube_2.txt','r');
    gt13 = fopen('youtube_3.txt','r');
    gt14 = fopen('Aircraft.txt','r');

    %select tracker
    if (i==1) tracker = 'ours.txt'; end
    if (i==2) tracker = 'CN.txt'; end
    if (i==3) tracker = 'CT.txt'; end
    if (i==4) tracker = 'DSST.txt'; end
    if (i==5) tracker = 'KCF.txt'; end
    if (i==6) tracker = 'SAMF.txt'; end
    if (i==7) tracker = 'STC.txt'; end

    %read all the OPEs
    str1 = strcat('airplane_001_OPE_',tracker);    fileid1 = fopen(str1,'r');
    str2 = strcat('airplane_004_OPE_',tracker);    fileid2 = fopen
(str2,'r');
    str3 = strcat('airplane_005_OPE_',tracker);    fileid3 = fopen(str3,'r');
    str4 = strcat('airplane_006_OPE_',tracker);    fileid4 = fopen
(str4,'r');
    str5 = strcat('airplane_007_OPE_',tracker);    fileid5 = fopen(str5,'r');
    str6 = strcat('airplane_011_OPE_',tracker);    fileid6 = fopen
(str6,'r');
    str7 = strcat('airplane_012_OPE_',tracker);    fileid7 = fopen(str7,'r');
    str8 = strcat('airplane_013_OPE_',tracker);    fileid8 = fopen
(str8,'r');
    str9 = strcat('airplane_015_OPE_',tracker);    fileid9 = fopen(str9,'r');
    str10 = strcat('airplane_016_OPE_',tracker);   fileid10 = fopen
(str10,'r');
    str11 = strcat('big_2_OPE_',tracker);          fileid11 =
fopen(str11,'r');
    str12 = strcat('youtube_2_OPE_',tracker);      fileid12 =
fopen(str12,'r');
    str13 = strcat('youtube_3_OPE_',tracker);      fileid13 =
fopen(str13,'r');
    str14 = strcat('Aircraft_OPE_',tracker);       fileid14 =
fopen(str14,'r');
```

```
    %initiating average precision, success rate and cle
    pr =[]; sr=[]; cle = [];

    %for each dataset, proceed now sequentially
    for j = 1 : 14 %25
        file = strcat ('fileid',num2str(j));
        file = eval(file);
        tline1 = fgetl(file);
        gtfile = strcat ('gt',num2str(j));
        gtfile = eval(gtfile);
        tline2 = fgetl(gtfile);
        C=[];D=[];
        A=[];B=[];
        while (ischar(tline2))
            C1 = textscan (tline1,'%f,%f,%f,%f');
            C2 = textscan (tline2,'%f,%f,%f,%f');

            %Scale adjustment for SAMF
            if i == 6
%                 C1{1} =  C1{1}/2;
%                 C1{2} = C1{2}/2;
                C1{3} = C1{3}*2;
                C1{4} = C1{4}*2;
            end
            % M1 = double (C1{1}) + ((double(C1{3}) - double(C1{1}))/2.0);
            % N1 = double (C1{2}) + ((double(C1{4}) - double(C1{2}))/2.0);
            M1 =  double (C1{1}) + (double(C1{3})/2.0) ; %if output is in
x,y,w,h format
            N1 =  double (C1{2}) + ( double(C1{4})/2.0);
            % M2 = double (C2{1}) + ((double(C2{3}) - double(C2{1}))/2.0);
            % N2 = double (C2{2}) + ((double(C2{4}) - double(C2{2}))/2.0);
            M2 = double (C2{1}) + (double(C2{3})/2.0) ; %if gt is in x,y,w,h
format
            N2 = double (C2{2}) + (double(C2{4})/2.0);
            % for SR
            % w1 = double(C1{3}) - double(C1{1});
            % h1 = double(C1{4}) - double(C1{2});
            w1 =  double(C1{3}); % if output in x,y,w,h format
            h1 =  double(C1{4});
            % w2 = double(C2{3}) - double(C2{1});
            % h2 = double(C2{4}) - double(C2{2});
            w2 = double(C2{3}); % if ground truth in x,y,w,h format
            h2 = double(C2{4});
            X1 = [C1{1} C1{2} w1 h1];
            X2 = [C2{1} C2{2} w2 h2];
            C = vertcat (C,X1);
            D = vertcat (D,X2);
            Z1 = [M1 N1];
            Z2 = [M2 N2];
            A = vertcat (A,Z1);
            B = vertcat (B,Z2);
            tline1 = fgetl(file);
            tline2 = fgetl(gtfile);
        end
       [p,~,c]=precision_plots(A,B,0); % calculate precision
       [s,~] = success_plots(C,D); % calculate SR
       pr = [pr, p]; % since we need to keepaccount of 50 points in each
dataset
       sr  = [sr, s];
       cle = [cle,c];
    end

    pr1 = mean(pr');
    sr1 = mean(sr');
    cle1 = mean(cle');

    pr_final(i,:) = pr1;        % Final PR    For drawing figure with 50
points
    sr_final(i,:) = sr1;        % Final SR
    cl_final(i,:) = cle1;       % Final CLE

    m1 = vertcat(m1, mean(pr)); % Final PR    For calculating the table
    s1 = vertcat(s1,mean(sr));  % Final SR
    cl = vertcat(cl,mean(cle)); % Final CLE
    fclose('all');
end

handle = fopen('avg_OPE_PR_SR_CLE.txt','w+');
%Printing the results to a file
fprintf(handle,'Precision Table\n');
fprintf(handle, '%10s %10s %10s %10s %10s %10s
%10s\n','Ours','CN','CT','DSST','KCF','SAMF','STC');


for jj = 1:14%25
    fprintf(handle,'%10.2g %10.2g %10.2g %10.2g %10.2g %10.2g
%10.2g\n',m1(1,jj),m1(2,jj),m1(3,jj),m1(4,jj),m1(5,jj),m1(6,jj),m1(7,jj));
end


fprintf(handle,'Success Rate Table\n');
fprintf(handle, '%10s %10s %10s %10s %10s %10s
%10s\n','Ours','CN','CT','DSST','KCF','SAMF','STC');

for jj = 1:14%25
    fprintf(handle,'%10.2g %10.2g %10.2g %10.2g %10.2g %10.2g
%10.2g\n',s1(1,jj),s1(2,jj),s1(3,jj),s1(4,jj),s1(5,jj),s1(6,jj),s1(7,jj));
end

fprintf(handle,'\n\nCLE\n + %g',mean(mean(cl)));

figure(1);
plot(pr_final(1,:),'Linewidth',2,'Color','r');
hold on;
plot(pr_final(2,:),'Linewidth',2,'Color','y');
plot(pr_final(3,:),'Linewidth',2,'Color','c');
plot(pr_final(4,:),'Linewidth',2,'Color','m');
plot(pr_final(5,:),'Linewidth',2,'Color','g');
plot(pr_final(6,:),'Linewidth',2,'Color','b');
plot(pr_final(7,:),'Linewidth',2,'Color','k');
xlabel('Center Location Error Threshold');
ylabel('Precision Rate');
title('Precision Plots of OPE');
```

```
legend('Ours','CN','CT','DSST','KCF','SAMF','STC');
grid on;
hold off;

x_axis = linspace(0,1,50);
figure(2);
plot(x_axis,sr_final(1,:),'Linewidth',2,'Color','r');
hold on;
plot(x_axis,sr_final(2,:),'Linewidth',2,'Color','y');
plot(x_axis,sr_final(3,:),'Linewidth',2,'Color','c');
plot(x_axis,sr_final(4,:),'Linewidth',2,'Color','m');
plot(x_axis,sr_final(5,:),'Linewidth',2,'Color','g');
plot(x_axis,sr_final(6,:),'Linewidth',2,'Color','b');
plot(x_axis,sr_final(7,:),'Linewidth',2,'Color','k');
xlabel('Overlap Threshold');
ylabel('Success Rate');
title('Success Plots of OPE');
legend('Ours','CN','CT','DSST','KCF','SAMF','STC');
grid on;
hold off;
```

B.2 The following MATLAB code tabulates and plots *TRE* PR and SR:

```
% This creates a text file for tabulating the Precision Rate and Success
% Rate for TRE of each dataset for each tracker
% Requirements: All the TREs and Ground Truth should be in the same folder
% TRE files should follow the format : datasetName_TRE_trackerName.txt
% Ground truth should follow the format: datasetName.txt
% Output file created is : avg_TRE_PR_SR_tracker.txt
% Precision and Success Rates of TRE

clear all;
close('all');
clc;

numImages = [200,200,200,200,200,300,300,300,300,300,382,475,301,620]; %
total number of images for each dataset

seq =
{'airplane_001';'airplane_004';'airplane_005';'airplane_006';'airplane_007';.
..

'airplane_011';'airplane_012';'airplane_013';'airplane_015';'airplane_016';'b
ig_2';...
       'youtube_2';'youtube_3';'Aircraft'}; % dataset names

tracker = 'DSST';         % tracker to operate on

DSST_prec = [];           % Precision Rate stored here for plotting
DSST_succ = [];           % Success Rate stored here for plotting

m1=[];s1=[];cl=[];

for kk = 1:size(seq,1)
    cntr = 0;
    incr = floor(numImages(kk)/20);
    prec = [];
    succ = [];

    gtfile = strcat(char(seq(kk)),'.txt');
    filegt = fopen(gtfile,'r');

    GT=[]; GTp=[];
    %read and store the gt file in a matrix
    tline1 = fgetl(filegt);
    while ischar(tline1)
        C1 = textscan (tline1,'%f,%f,%f,%f');
        GT = vertcat(GT,[C1{1} C1{2} C1{3} C1{4}]);
        GTp = vertcat (GTp, [C1{1}+(C1{3}/2), C1{2}+(C1{4}/2)]);
        tline1 = fgetl(filegt);
    end
    fclose(filegt);

    % SAMF scale adjustment
%     if kk== 3 || kk == 14 || kk == 15 || kk==16 ||kk==20
%              a = 2;
%          elseif kk == 1 || kk == 2 || kk == 7
%              a = 1.5;
%          else
%              a = 1.1;
%      end

    for ii = 1:incr:numImages %20 iterations for each dataset
        cntr = cntr + 1;
        if cntr == 20 break; end
        if ii == 1
            str = strcat(char(seq(kk)),'_OPE_',tracker,'.txt')
        else
            str =
strcat(char(seq(kk)),'_TRE_',tracker,'_',num2str(ii),'.txt')
        end

        % adjust for tracker stoppage problem
%         if strcmp('airplane_001_TRE_DSST_11.txt',str) break; end
%         if strcmp('airplane_004_TRE_DSST_91.txt',str) break; end
%         if strcmp('airplane_005_TRE_DSST_101.txt',str) break; end
%         if strcmp('airplane_006_TRE_DSST_51.txt',str) break; end
%         if strcmp('airplane_007_TRE_DSST_181.txt',str) break; end
%         if strcmp('airplane_011_TRE_DSST_91.txt',str) break; end
%         if strcmp('airplane_012_TRE_DSST_61.txt',str) break; end
%         if strcmp('airplane_013_TRE_DSST_136.txt',str) break; end
%         if strcmp('airplane_015_TRE_DSST_61.txt',str) break; end
%         if strcmp('big_2_TRE_DSST_191.txt',str) break; end
%         if strcmp('youtube_2_TRE_DSST_70.txt',str) break; end
%         if strcmp('youtube_3_TRE_DSST_196.txt',str) break; end
%         if strcmp('Aircraft_TRE_DSST_187.txt',str) break; end

        % read and store ouput file in a matrix
        fileop = fopen(str,'r');
        tline2 = fgetl(fileop);
        OP = [];OTp=[];
        while ischar(tline2)
            C1 = textscan (tline2,'%f,%f,%f,%f');
            OP = vertcat(OP,[C1{1} C1{2} C1{3} C1{4}]);
            OTp = vertcat (OTp, [C1{1}+(C1{3}/2), C1{2}+(C1{4}/2)]);
            tline2 = fgetl(fileop);
        end
        fclose(fileop);

        % call the precision function
        [p,~,c]=precision_plots(OTp,GTp(ii:end,:),0);

        % call the success function
        [s,~]=success_plots(OP,GT(ii:end,:));

        prec = horzcat(prec,p);
        succ = horzcat(succ,s);
    end
    if size(prec,2) == 1 %adjust only containing OPE for some trackers
        DSST_prec = horzcat(DSST_prec,prec);
        DSST_succ = horzcat(DSST_succ, succ);
    else
        DSST_prec = horzcat(DSST_prec, mean(prec')');
        DSST_succ = horzcat(DSST_succ, mean(succ')');
    end
end

save DSST_tre.mat DSST_prec DSST_succ           % Saves the Precision Rate
and Success Rate in Tracker.mat file to be plotted
fprintf('Precision rate for %s is %.2g.\nSuccess rate for %s is
%.2g\n',tracker, mean(mean(DSST_prec')), tracker, mean(mean(DSST_succ')));

figure(1);
plot(mean(ours_prec'),'Linewidth',2,'Color','r');
hold on;
plot(mean(CN_prec'),'Linewidth',2,'Color','y');
plot(mean(CT_prec'),'Linewidth',2,'Color','c');
plot(mean(DSST_prec'),'Linewidth',2,'Color','m');
plot(mean(KCF_prec'),'Linewidth',2,'Color','g');
plot(mean(SAMF_prec'),'Linewidth',2,'Color','b');
plot(mean(STC_prec'),'Linewidth',2,'Color','k');
xlabel('Center Location Error Threshold');
ylabel('Precision Rate');
title('Precision Plots of TRE');
legend('Ours','CN','CT','DSST','KCF','SAMF','STC');
grid on;
hold off;

x_axis = linspace(0,1,50);
figure(2);
plot(x_axis,mean(ours_succ'),'Linewidth',2,'Color','r');
hold on;
plot(x_axis,mean(CN_succ'),'Linewidth',2,'Color','y');
plot(x_axis,mean(CT_succ'),'Linewidth',2,'Color','c');
plot(x_axis,mean(DSST_succ'),'Linewidth',2,'Color','m');
plot(x_axis,mean(KCF_succ'),'Linewidth',2,'Color','g');
plot(x_axis,mean(SAMF_succ'),'Linewidth',2,'Color','b');
plot(x_axis,mean(STC_succ'),'Linewidth',2,'Color','k');
xlabel('Overlap Threshold');
ylabel('Success Rate');
```

```
title('Success Plots of TRE');
legend('Ours','CN','CT','DSST','KCF','SAMF','STC');
grid on;
hold off;

m1 = vertcat(m1, mean(DSST_prec)); % Final PR of each tracker   For
calculating the table
s1 = vertcat(s1,mean(DSST_succ));  % Final SR of each tracker

handle = fopen('avg_TRE_PR_SR_DSST.txt','w+');
%Printing the results to a file
fprintf(handle,'Precision Table\n');
% fprintf(handle, '%10s %10s %10s %10s %10s %10s
%10s\n','Ours','CN','CT','DSST','KCF','SAMF','STC');
fprintf(handle, '%10s \n','SAMF');
for jj = 1:14
    fprintf(handle,'%10.2g \n',m1(1,jj));
end


fprintf(handle,'Success Rate Table\n');
% fprintf(handle, '%10s %10s %10s %10s %10s %10s
%10s\n','Ours','CN','CT','DSST','KCF','SAMF','STC');
fprintf(handle, '%10s \n','SAMF');
for jj = 1:14
    fprintf(handle,'%10.2g \n',s1(1,jj));
end
```

B.3 The following MATLAB functions were used from [8] to tabulate and plot *OPE* and *TRE* PR and SR in B.1 and B.2 of Appendix B:

```
function [precisions, distances, averageLocationError] =
precision_plot(positions, ground_truth, show)
%PRECISION_PLOT

    max_threshold = 50;  %used for graphs in the paper


    precisions = zeros(max_threshold, 1);

    if size(positions,1) ~= size(ground_truth,1),
        fprintf('Number of ground truth frames does not match number of
tracked frames.\n')
        return
        %just ignore any extra frames, in either results or ground truth
        %n = min(size(positions,1), size(ground_truth,1));
        %positions(n+1:end,:) = [];
        %ground_truth(n+1:end,:) = [];
    end

    %calculate distances to ground truth over all frames
    distances = sqrt((positions(:,1) - ground_truth(:,1)).^2 + ...
                     (positions(:,2) - ground_truth(:,2)).^2);
    distances(isnan(distances)) = [];

    %compute precisions
    for p = 1:max_threshold,
        precisions(p) = nnz(distances <= p) / numel(distances);
    end

    averageLocationError = sum(distances)/numel(distances);

    %plot the precisions
    if show == 1,
%       figure('Number','off', 'Name',['Precisions - ' title])
        grid on;
        plot(precisions, 'k-', 'LineWidth',2)
        xlabel('Local error threshold'), ylabel('Precision')
    end

end



function [ success_rate, OverlapRatio ] = success_plot( positions,
ground_truth )
% positions and ground_truth format: [x,y,w,h] of the top-left point of the
% bounding box


Ratio = bboxOverlapRatio(positions, ground_truth);
OverlapRatio = diag(Ratio);
threshold = [0.02:0.02:1];
N = size(threshold,2);
success_rate = zeros(N,1);
for i = 1:N
    success_rate(i) = nnz(OverlapRatio >= threshold(i)) /
numel(OverlapRatio);
end
%figure, plot(success_rate, 'k-', 'LineWidth',2);
end
```