REAL-TIME OBJECT TRACKING IN AERIAL IMAGES

by

Hüseyin Büyükeşmeli

Submitted to Graduate School of Natural and Applied Sciences
in Partial Fulfillment of the Requirements
for the Degree of Doctor of Philosophy in
Electrical and Electronics Engineering

Yeditepe University

2019

REAL-TIME OBJECT TRACKING IN AERIAL IMAGES

APPROVED BY:

Prof. Dr. Cem Ünsalan .....
(Thesis Supervisor)
(Marmara University)

Prof. Dr. Oğuzhan Urhan .....
(Kocaeli University)

Prof. Dr. Sezer Gören Uğurdağ .....
(Yeditepe University)

Prof. Dr. Yusuf Sinan Akgül .....
(Gebze Teknik University)

Assoc. Prof. Dr. Engin Maşazade .....
(Marmara University)

DATE OF APPROVAL: .... /.... /2019

ACKNOWLEDGEMENTS

This has been a very long and rough journey for me. It has been more painful than it should have been. But during this journey, I am most thankful to my advisor Cem Ünsalan. I am aware that I pushed the limits of his patience, even then he did not give up on me. It was the most memorable moments of my life that making me realize the unnecessity of being happy. I can not thank him enough for his guidance and effort on me. I am also grateful to Assoc. Prof. Dr. Engin Maşazade and Prof. Dr. Oğuzhan Urhan for their support and contributions.

I was not alone in this journey. There is one other person who shares the similar path with me. We shared same class, office, house in past fifteen years. I thank to H. Deniz Gürhan for a friendship that will last forever.

Besides his beautiful friendship, I am thankful to Ahmet Dinçsoy especially for helping me to take footage for this study.

I present my endless gratitude to Sibel Şentürk for the love and compassion that she showed. I am especially grateful that she endured me even though I was a grumpy and annoying person, as I approach the end of the thesis.

Even though her questionable objectivity, I would like to thank to Burçin Başyazıcı for the hand that she offered whenever I was stuck.

Family is a complicated phenomenon. They are most valuable people in my life however, I cannot thank them for this study. So, I thank to Aygün Büyükeşmeli and Mustafa Büyükeşmeli for being who they are.

ABSTRACT

REAL-TIME OBJECT TRACKING IN AERIAL IMAGES

Visual object trackers usually operate in real-time required by their nature. Hence, the processing power demanded by them becomes very important. Moreover, fast trackers with low memory footprint are very important for space, time, and power constrained applications such as UAV on-board visual trackers. In this dissertation, a visual object tracker which can operate on aerial images with low processing load is proposed.

The proposed method consists of two main parts as object representation and tracking. The novelty of our method is in adding a simple yet flexible probabilistic object representation method to the tracking framework. The proposed probabilistic object representation method is based on classic edge or image feature detection methods. This representation allows us modifying the tracker based on Bayesian tracking framework. To do so, we use three methods as follows. First, the grid based pdf approximation is used in tracking. Second, Kalman filter is used to simplify the framework and speed up the tracking process. Third, particle filtering is applied to decrease the computational cost and increase success rate.

The proposed method requires such a low processing power that it can work on an ARM Cortex-M7 microcontroller. Hence, we implemented it on both such a microcontroller and PC. The implementation step also includes novel contributions in terms of decreasing computation cost and speed up processes. Besides, we also compared the proposed method with the state of the art in literature. This allowed us to justify the usefulness of the proposed method especially when implemented on a microcontroller.

ÖZET

HAVA GÖRÜNTÜLERİNDE GERÇEK ZAMANLI NESNE TAKİBİ

Görsel nesne takibi yöntemlerinin, doğası gereği, genellikle gerçek zamanlı çalışmaları istenir. Bu durumda, takip edici algitmanın gerektirdiği işlem yükü oldukça önemlidir. Hızlı ve düşük hafıza gereksinimi ile çalışan görsel takip ediciler zaman, yer ve güç bakımından kısıtlı uygulamalar için çok önemlidir. Bunlara örnek olarak insansız hava araçları üzerinde çalışan takip edici uygulamalar gösterilebilir. Bu tezde hava görüntüleri için çalışacak, işlem yükü oldukça az bir görsel nesne takip edici sunulmaktadır.

Sunulan yöntem, obje temsili ve takibi olmak üzere iki ana kısımdan oluşmaktadır. Bu tezde sunulan yenilik, hedefi basit fakat gayet esnek bir şekilde olasılıksal sonuç veren nesne bulucu bir algoritmanın takip etme yöntemleri ile birleştirilmesidir. Olasılıksal nesne temsil yöntemi geleneksel kenar bulma veya görüntü öznitelikleri temellidir. Olasılıksal gösterim, Bayesçi takip methodunu uygulamamıza olanak verir. Bu amaç için üç farklı yöntem denedik. Bunlar sırası ile şu şekildedir. İlk olarak kafes tabanlı pdf tahmini denendi. İkinci olarak Kalman filtresi ile takip işlemin basitleştirip hızlandırdık. Üçüncü olarak da, parçacık filtresi işlem yükünü azaltmak ve takip başarını arttırmak için kullandık.

Bu tezde sunduğumuz yöntemin gerektirdiği işlem yükü o kadar azdır ki ARM Cortex-M7 mikrodenetleyici üzerinde çalışabilir. Dolayısıyla yöntemden sonuçları elde edebilemk için, hem bilgisayar için hem de mikroişlemci için uyguladık. Uygulama adımı işlem yükü azaltma ve hızlandırma bakımından yenilikler içermektedir. Bunun yanında önerilen yöntemimizi literatüredeki diğer önemli yöntemler ile karşılaştırdık. Böylece yöntemimizin özellikle mikroişlemciler için işlevselliğini göstermiş olduk.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

LIST OF SYMBOLS/ABBREVIATIONS

AUC	Area under the curve
BRISK	Binary robust invariant scalable keypoints
CDF	Cumulative density function
CPU	Central processing unit
FPGA	Field programmable gate array
FPS	Frames per second
GPS	Global positioning system
GPU	Graphical processing unit
HOG	Histogram of oriented gradients
I^2C	Inter-integrated circuit
KCF	Kernelized correlation filter
MAV	Micro aerial vehicle
MSER	Maximally stable extremal regions
OPE	One-pass evaluation
PC	Personal computer
PDF	Probability density function
RAM	Random access memory
ROI	Region of interest
ROM	Read only memory
SIFT	Scale-invariant feature transform
SURF	Speeded up robust features
SLAM	Simultaneous localization and mapping
SVM	Support vector machine
UAV	Unmanned aerial vehicle

1. INTRODUCTION

Visual tracking is one of the fundamental problems in computer vision. It has a broad application area including surveillance. One of the most common data source for tracking is aerial imagery. Real-time applications are highly important for surveillance applications. Unmanned Aerial Vehicles (UAV) are common choice for such applications.

UAVs have been used more often in our daily life compared to the past decade. Their history begun with military applications. However, their usage in commercial or hobby purposes increased exponentially recently. Their structure is quite flexible. Their scale can vary from as small as couple of centimeter to meters [1]. They are also very suitable for autonomous control. Hence, their possible application areas are diverse. UAVs can employ different sensor types. Among these, the most common one is a visual sensor which is a camera. Vision sensor are placed on UAVs for various purposes. They can be used to control the vehicle such as landing spot detection, pose estimation, or motion control. They can also be dedicated to surveillance of the environment. Hence, these sensors became a common aerial image source.

Microcontrollers are inseparable part of UAVs. They are employed in the center of the flight controller unit which is mainly responsible for stable flying of the UAV. In this unit, microcontrollers conduct the control algorithm of the vehicle. They gather the peripheral sensor data as input and control the rotation speed of motors as output according to the command incoming from the user or path planner. A microcontroller is a suitable hardware for such controlling task due to its small size and low power consumption. On the other hand, microcontrollers usually are not the choice for vision tasks due to their speed and memory constraints. However, they are improving.

Usually, on-board visual tasks in UAVs are conducted by a companion processor in additional to the flight controller since such visual tasks bring burden to the flight controller in terms of power consumption and processing load. In this thesis, we propose lightweight visual trackers such that they can operate in real-time within limits of the microcontroller. Thus, we can deploy both visual tracking and flight control process on the same platform.

Our tracker is based on a probabilistic object representation method. Within it, we combine image features such as corner points and apply a probabilistic voting procedure to obtain a probability density function (pdf).

One of the challenges of trackers using image features is to associate them from model to search image [2]. Our voting method eliminates this step also. Moreover, our detection algorithm can operate with fairly simple feature extraction methods such as Harris corner detection [3]. This way, we can achieve fast object detection with fairly simple operations.

We combine our detection algorithm with Bayesian tracking framework. Hence, we can exploit the object trajectory information to improve tracking performance [4]. The Bayesian tracker recursively combines two pdfs, one for prediction and one for existence of the target in order to achieve better tracking performance. We can directly use our object detector with this framework since it provides a pdf. To do so, we proposed three different but closely related methods. First, we use grid based pdf approximation. Here, we benefit from the pdf obtained from the object detection step in tracking. Second, we use the Kalman filter with our object detection result [5]. Third, we use particle filtering to speed up to the tracking process [6].

This thesis consists of six chapters. The first chapter introduces overview of the thesis and general concepts. The second chapter reviews literature. The third chapter introduces theoretical foundations of the proposed tracker. The fourth chapter explains implementation details and variations of the tracker. In the fifth chapter, we present experimental results and comparison with state of the art methods from literature. In the sixth chapter, we conclude our study.

2. LITERATURE SEARCH

In this chapter, we revise and list related work in literature in broad spectrum. First, we revisit different UAV applications for possible source of aerial imagery. Then, we focus on visual processing aspect of the problem. Visual object tracking is our main focus subjects. Therefore, we list visual trackers and their usage with UAVs. Here, we talk about the opportunity that can we fill in the literature. In the last section, we list studies in detail that we used to compare in this study.

2.1. UAV APPLICATIONS

UAVs have been getting more attention in past decade. Thanks to their affordable cost and modular structure, they provide an attractive option for building an autonomous system. Many research have been reported about their usage in applications such as plant inspection, search and rescue, traffic monitoring, and surveillance. In this section, UVAs are reviewed as well as their applications are listed.

Khanna *et al.* [7] presented an application of UAVs in precision agriculture. They proposed a pipeline of offline processing 3D point cloud of crop field. Proposed method is not real time and requires post processing of aerial images with third party software to obtain 3D point cloud.

Faessler *et al.* [8] presented a system including a quadrotor equipped with a monocular camera and laptop serving as ground station. Its purpose is terrestrial 3D reconstruction for search and rescue purposes. This system works in real-time. Its usage for both indoor and outdoor environments are presented. The proposed system consists of three main parts as flight controller, single board PC, and laptop. Single board PC obtains images and applies preprocessing operations to calculate pose estimation and sends images to laptop. 3D reconstruction is implemented on the laptop. Flight controller handles the stable flight of the drone and positioning according to the pose estimation done in single board PC.

Delmerico *et al.* [9] proposed a system for guiding a ground robot with UAV through unknown environment to a destination for search and rescue purposes. A map of the terrain

class and elevation map is built by vision based flying robot. The aim is to minimize the deployment time of the terrain classifier. Therefore, accuracy of the classification is not the main concern. The proposed system can generate the region map after 60 seconds of flight.

Giusti *et al.* [10] presented a novel system based on deep neural networks for navigating a robot in unseen forest track. The system makes a decision about to go whether left, right or straight by processing the image taken by onboard camera. Big dataset is used to train deep neural network. This dataset is obtained with three cameras facing different directions attached to a hiking person in the forest. Training procedure is realized in off-board PC. Therefore, gathering dataset and training takes long time.

Mueggler *et al.* [11] proposed a mission planning algorithm for autonomous guiding ground robot by aerial robot. Ground robot moves through moveable and non-moveable objects in a scenario such as delivering first aid kit. UAV executes its mission without human interaction. No specific object detection algorithm is deployed. Movable or non-movable objects are recognized using AprilTag [12].

Minaeian*et al.* [13] proposed a vision-based crowd detection and geographic information system (GIS) localization algorithm for a cooperative team of one unmanned aerial vehicle and number of unmanned ground vehicles (UGV). UGVs are used to convert the image locations of the detected targets into their GIS coordinates. Moreover, a test-bed consists of unmanned vehicles and an agent-based simulation model are developed to conduct experiments.

Gohl *et al.* [14] presented a study that aims to avoid obstacles based on 3D model of the close environment. A general map is not constructed. In this work eight cameras, one stereo vision for four sides, are employed. FPGA and an on-board PC is used for 3D modeling. Proposed system requires heavy hardware and a lot of power consumption. Reported time of flight is around eight minutes.

Fraundorfer *et al.* [15] combined stereo vision and optical flow in a system for exploration purposes. Vector Field Histogram Plus (VFH+) for local navigation and the frontier-based exploration algorithms are deployed with stereo camera in their system. Moreover, the bug algorithm is implemented with downward looking optical flow camera as a secondary

navigation algorithm in case the first one fails in sparse environment [16]. These algorithms are implemented on-board PC. Also a simultaneous localization and mapping (SLAM) algorithm is implemented off-board in order to complete system.

Schauwecker *et al.* [17] deployed two separate stereo cameras, one forward looking and one downward looking, to navigate an UAV. A SLAM algorithm is implemented with forward looking camera pair. Downward looking camera pair is used for the tasks such that ground plane detection, six degree-of-freedom (6 DoF) pose estimation and motion tracking. Results of two camera pairs are combined with a Kalman filter.

Engel *et al.* [18] proposed a visual navigation system with a low-cost UAV to visually navigate when GPS is not available. Weiss *et al.* [19] presented a framework that increases robustness of navigation of aerial vehicles using multi-sensor fusion. They are both structured on parallel tracking and mapping (PTAM) algorithm [20]. PTAM is a feature-based SLAM algorithm that achieves robustness through tracking and mapping several hundreds of features.

Applications mentioned in this section can be categorized as offline and real-time. Offline applications generally require heavy data processing. One such example for this operation is terrestrial mapping. We can group real-time applications into two groups as off-board and on-board from visual processing perspective. Off-board applications transmit data to the ground processing unit. This transmission requires wireless data link with high bit rate. Therefore, it is power consuming. On-board applications need to carry a companion processing unit. This unit needs space on the board and it consumes valuable onboard power. Hence processing power demand by the visual application becomes very important for on-board applications.

2.2. TRACKING

Visual object tracking is the task of locating a target (or multiple targets) over sequential video images. Visual tracking is a fundamental field of study in computer vision. Its scope of usage varies from human-computer interaction, surveillance, traffic control to medical imaging, and augmented reality. Main objective of object tracking is finding similarities between object in sequential video frames to obtain location, velocity, direction

information of the target. It cannot be said that there is a certain algorithm that provides a superior performance. Performance of trackers can vary depending on the target or environment. Moreover, they may perform differently under various tracking challenges such as varying lighting condition and occlusion. In this section, a collection of trackers are listed. Moreover, studies that implement tracking process using unmanned aerial vehicle are reviewed.

Visual object tracking is a well-studied problem in literature. Every year, challenges are organized across the world. If we look at the historical evolution of visual trackers, we see that generative methods are generally used until mid 2000s. Then discriminative methods are widely studied until 2014. Then, correlation filter based tracker emerged. These trackers demonstrate high tracking performance and operation speed. Nowadays, deep learning based methods yield the best performance in tracking success. However, these methods require high computational power for real-time applications.

Yilmaz *et al.* [2] present a survey on trackers. It is the first comprehensive tracker survey in literature that can be considered as a good reference to understand main types and methodology of tracking problem. Smeulders *et al.* [21] present a more recent survey with nineteen trackers which appear until 2011. This survey covers most of the trackers and it also evaluates and compares trackers on a common dataset. A benchmark on an aerial image is presented by Li *et al.* [22]. In this work a novel camera motion model is implemented and compared with different type of trackers.

Briechle *et al.* [23] presented the most common act of tracking as directly matching target and candidate patches with normalized cross-correlation measure. Baker *et al.* [24] presented a Lucas-Kanade tracker which matches affine-transformed target patch bounding box and candidate patches windows around the previous location. Arandjelović [25] implemented a similar method on aerial images with vehicles as targets. Nguyen *et al.* [26] handled occlusion problem with appearance-predicted matching. The target is represented by 20x20 template intensities each of which is associated with a separate Kalman filter. Adam *et al.* [27] matched ensemble of patches obtained from target bounding box by breaking it into patches. Tracker can handle partial occlusion and pose changing patch by patch. However, fragments or patches are easy to drift away around the ones with similar

appearance. Meer *et al.* [28] performed a mean-shift tracking by matching histograms rather than using any spatial information. Oron *et al.* [29] pursued adaptation in object by matching flexible rigidity. Target patch is represented by super pixels which consist of center of mass and average HSV values. Ross *et al.* [30] presented a particle filter tracker that computes eigen images of the target with incremental PCA over target's intensity-value template. Kwon *et al.* [31] used traditional (translation, scale, rotation) motion types to extend target's appearance model. Maggio and Cavallaro [32] divided object into four non-overlapping rectangle parts. It conducts the object searching by mean shift, which has poor adaptability for appearance and illumination change. Cao *et al.* [33] presented a KLT based tracker using good features to tracking airborne videos [34]. These tracking methods require a model to operate and they need to maintain the condition of the template.

Kim *et al.* [35] proposed an emergency navigation system that works in GPS-denied environments. During the flight, in case of sudden absence of GPS signal, proposed navigation system keeps relative navigation and starts visual object tracking. Hence, the vehicle with the proposed system does not lose its location completely and wonders around a ground land mark even if its location is unknown. Kernelized mean-shift tracker is used to track the landmark. Then, the tracker output is combined with an inertial sensor unit (IMU) through an extended Kalman filter (EKF). Thus, it can perform its state estimation.

Yang *et al.* [36] proposed a method that generates the HSI histogram of the super pixels from extended target region. Using mean-shift clustering algorithm super pixels are grouped with a confidence value. In new frame, candidate windows are sampled with Gaussian distribution. Candidate window with highest confidence value is considered as the target.

Lu *et al.* [37] obtained a probability map using local binary pattern, color histogram local binary pattern (LBP) histogram, RGB color histogram and pixel-pattern-based texture feature (PPBTF) histogram with three different SVM classifier. Then, it uses mean-shift algorithm to obtain modes of the probability map. Afterwards, image features are used for improving mean-shift tracking algorithm. Xiaoyan and Shir [38] used local binary pattern features in addition to color histogram. A Kalman filter is used in case of partial or full occlusion occurs.

Szottka and Butenuth [39] presented an improved version of particle filter. Particles represent state vector as location and direction. Standard particle filter is concerned additions from three aspect has been made, particle distribution, particle guiding and template update. Distribution of particles is implemented efficiently by adding sequential adaptive noise to motion model. Particle guiding is added by changing directions of fraction particles according to best matching particle. Template update is implemented by changing likelihood function adaptively. However proposed system does not include detection of vehicles. Initialization parameters and start location of vehicles are done manually.

Canosa *et al.* [40] presented a real-time method to detect and track moving objects (DATMO) from UAVs using a single camera. Moving objects are detected using optical flow and the targets are tracked using Kalman filter. System can work real-time on an on-board PC. It is vulnerable to sudden camera view changes. Target needs to have minimum speed to be distinguished by optical flow detection algorithm.

In the mentioned studies, there are a number of trackers for different use cases. As for UAVs, real-time tracking operations are conducted rather on-board or off-board. For on-board applications, a secondary processing unit is added to the UAV. This unit can be either a single board PC with high processing power or FPGA or GPU if the main purpose is parallel processing. However, these units require additional space and they increase the power consumption of the vehicle. There are limited studies on implementing a vision task such as tracking on a microcontroller. We aim to fill this gap by proposing a lightweight visual tracker that can run even on a microcontroller.

2.3. OBJECT DETECTION

Object detection is one of the main parts of visual tracking. In this section, we revisit the common object detection methods; then focus on applications with UAVs. Studies in literature can be categorized in different groups yet a certain distinction cannot be made because fusion of multiple methods can be found in many studies. Object detection can be realized based on matching of feature points of the target and the candidate. Harris, KLT and SIFT are the most commonly used features in object detection [3, 41, 42]. Moving objects can be detected by background subtraction. Wu and Zhang [43] and Liu *et al.* [44] presented

object detectors using relatively constant background images. However, these detectors are highly sensitive to illumination. Statistical features of pixels can be used for background subtraction with the mixture of Gaussian method to obtain changing background [45, 46]. Davis *et al.* [47] presented a detector which uses mean, standard deviation, and brightness changing values of pixels.

Object detection in aerial images has been broadly studied in literature. A comprehensive survey is presented by Cheng and Han [48]. This study reviews publications on this field by categorizing them into four groups as template matching, knowledge, feature, and machine learning based object detection methods.

Detection of various objects using UAVs have been submitted. Rodriguez *et al.* [49] developed a low cost UAV for land-mine detection. Malek *et al.* [50] detected palm trees using UAV images. Popp *et al.* [51] and Katrasnik *et al.* [52] proposed power line inspection with mobile robots and discussed for navigation and avoidance purposes. Chen *et al.* [53] presented a study on inspecting buildings in order to detect structural changes using RGB-D camera. Popp *et al.* [54] presented a real time building detection system on UAV. Flores *et al.* [55] presented a system about finding and passing through a window with a micro UAV (MAV) in GPS-denied environments. Yang *et al.* [56] proposed a method for landing area detection using SIFT features. Moranduzzo *et al.* [57] presented an offline car counting implementation in high resolution UAV images. Asphalt areas are obtained from geographic information system (GIS). SIFT features are generated in segmented asphalt areas. Obtained feature points are merged using morphological operations. SVM classifier is deployed to classify merged key points. Tuermer *et al.* [58] presented a pipeline for detecting vehicles in dense urban areas. The road information is obtained using road database and global digital elevation map (DEM). Disparity map is generated on road segments. Irrelevant parts (vegetation, buildings) are eliminated using height information. Remaining parts are classified based on histogram of gradients (HOG) features. For tracking applications, feature points are needed to be matched between target and candidate. Object detection methods based on features like SIFT and HOG are very time and memory consuming methods. Classification processes combined with these features increases demand for the process power from application platform.

Liu and Mattyus [59] proposed a method for detection and classification (car/truck) of vehicles. The vehicle location is detected by using integral channel features in 21-MPixel images. Detected vehicles are classified with a binary classifier which gives the type and orientation of the vehicle. Their system is implemented in an off-board PC. Proposed system takes a few seconds to produce result. Thus it is not suitable for real-time applications.

Chen *et al.* [60] proposed system that is based on a deep neural network to detect vehicles in satellite images. A hybrid deep neural network is implemented in order to learn rich features at many scales. Kluckner *et al.* [61] presented a method for counting cars in aerial images. An on-line AdaBoost algorithm is deployed for learning and selecting features. Also stereo matching information is used to reduce false positive results.

Our object detection method can work with simple feature detection methods and relies on probabilistic detection hence it does not require an additional matching step during the tracking process.

2.4. BENCHMARK METHODS

We selected several trackers from literature in order to compare our tracker's performance. Simple template and histogram matching methods are implemented with Kalman and particle filters. Besides these classical methods, state of the art trackers are also included for the benchmark results. These trackers are selected from survey [22]. Since the main targeted platform of this work is microcontrollers, neural network based trackers which require heavy computing power are excluded.

Kalman filter is one of the most commonly used in Bayesian tracking framework. In order to compare our tracking algorithm with a standard Kalman filter, we implemented a template matching based tracker. Details of the Kalman filter implementation are explained in the following sections. We implemented a template matching detection algorithm as implemented in OpenCV as

$$R(x,y) = \sum_{x',y'} \left(T(x',y') - I(x+x',y+y')\right)^2 \tag{2.1}$$

where T is the template image of the target, I is the input image and R is the resultant map of the correlation calculation [62]. This map is calculated in an interested region and the maximum valued location of the map is accepted as target location. Then this result will become the measurement input to Kalman filter.

We also implemented a reference particle filter tracker for comparison. Particle filter is another method in Bayesian tracking which is described in detail in following sections. For this purpose, we selected a color histogram matching method as introduced in [28]. As mentioned earlier, Meer *et al.* performs a mean-shift algorithm for tracking with color histogram matching using Bhattacharyya distance. They estimate target model pdf as

$$\hat{\mathbf{q}} = \{\hat{q}_u\}_{u=1,2,\cdots m} \quad \sum_{u=1}^{m} \hat{q}_u = 1 \tag{2.2}$$

and candidate model pdf as

$$\hat{\mathbf{p}}(\mathbf{i}) = \{\hat{p}_u(i)\}_{u=1,2,\cdots m} \quad \sum_{u=1}^{m} \hat{p}_u = 1 \tag{2.3}$$

where i denotes the candidate number. Then, the Bhattacharyya distance is calculated for every candidate as $d_i = \sqrt{1 - \rho(\hat{p}(i), \hat{q})}$ where ρ is the Bhattacharyya constant calculated as $\rho(\hat{p}(i), \hat{q}) = \sum_{u=1}^{m} \sqrt{\hat{p}_u(i)\hat{q}_u}$.

Instead of the mentioned Bhattacharyya distance, in this study we used its modified version as $w_i = e^{\alpha d_i}$ for weight distribution in standard particle filter. Here, α is the decaying constant for monotonically decreasing similarity function.

Another tracker that we select as a benchmark is the kernelized correlation filter (KCF) tracker [63]. KCF is one of the famous tracking-by-detection trackers in literature. KCF converts the convolution process with element-wise multiplication by exploiting circular

matrix properties. This process speeds up the detection process such that it can run up to 300 fps. Moreover implementation of KCF is very easy. It can be realized with few line of codes in MATLAB.

Another method that we will be using in comparison is the DAT tracker [64]. DAT uses the target color histogram in order to construct Bayes decision rule. It can potentially detect possible distractions near the target and prevents the tracker to drift away. This approach allows efficient scale estimation. Therefore, it improves the accuracy and robustness performance of the tracker.

Zhang *et al.* [65] proposed a model to solve drift problems in online tracking with using multi-expert ensemble method (MEEM) of a tracker and its past snapshots. Instead of preventing bad updates on model, a correction method is used to minimize the effects of unwanted updates after they happen. The best expert is preferred to update current tracker by choosing the minimum entropy one when a disagreement among experts occur. The main tracker is utilized from online SVM on a budget algorithm and an explicit feature mapping method is used to make more efficient updates and results.

Wang *et al.* [66] studied state-of-the-art trackers in literature. The study focuses on effects of each tracker component on performance. They take into consideration five components of trackers such as motion model, feature extractor, observation model, model updater, and ensemble post-processor. Their results show that the feature extractor is the most important part of tracker to improve tracking performance. Observation model selection also affects the performance if weak features are used. When the observation model and feature extractor effects are analyzed on the tracking performance, motion model has an insignificant effect. Under scale variation and fast motion, design of the motion model properly gets an important role to improve performance of the tracker. Despite model updater have very significant impact, the number of studies are limited. Finally, the ensemble post-processor has a significant contribution on the performance during study with high diversity. We will refer this study as HOGLR in the following sections.

Danelljan *et al.* [67] presented an approach to solve estimating the location of a target in each frame of an image sequence problem. They propose a tracker which learns discriminative

correlation filters for estimating translation and scale separately. We will refer this study as DSST in the following sections.

3. THE PROPOSED METHOD

This chapter explains the proposed object tracking method in detail. General block diagram of our tracker is as in Figure 3.1. As can be seen in this figure, our tracker estimates the target location in the update step by combining two sources of information as measurement and prediction. All steps in this diagram, as measurement, prediction, and update are represented in probabilistic form within our method. This indicates that output of each step is probabilistic as well.

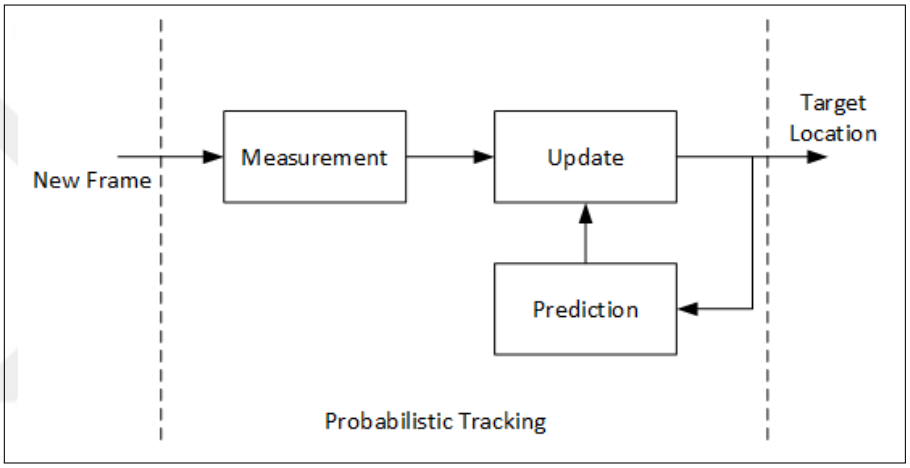

Figure 3.1. Block diagram of the proposed tracker.

We can briefly summarize our tracking framework based on Figure 3.1 as follows. When a new frame comes, we first obtain the probabilistic object representation. This leads to the measurement step, which indicates the existence of the target being tracked. Next, the prediction step generates a pdf from motion of the target. Afterwards, the update step combines these two pdfs to produce the final pdf which locates the target. In this chapter, we will present the calculation and variations of these steps. First, we will construct our probabilistic object representation in Section 3.1. Then in Section 3.2, we will present the Bayesian tracking framework which is a recursive algorithm that associates measurement and prediction pdfs. In Section 3.3, we will discuss how to obtain measurement pdf with our object representation and how to conduct prediction step based on dynamic motion models. Finally, in Section 3.4 we will present different implementations of Bayesian

tracking algorithm as grid-based approach, particle and Kalman filtering.

3.1. PROBABILISTIC OBJECT REPRESENTATION

Assume that we have a sequence of images at hand and we would like to track an object of interest within this sequence. Let's call the image sequence as $I_k(x, y)$ where k represents the image (frame) number in the sequence. We base our method on discrete time instants such that the initial time is taken as $k = 0$. Hence, the first image becomes $I_0(x, y)$. The user selects the target object to be tracked in this first frame. As a result, we have the template image subwindow as $T(x, y) = I_0(x_c \pm w/2, y_c \pm h/2)$ where (x_c, y_c) is the center of the object to be tracked. Here, w and h are the width and height of the subwindow, respectively. We provide the schematic representation of this operation in Figure 3.2. For simplicity target is illustrated as a rectangle box in a region of interest. In an actual operation, the target to be tracked can have a more complex shape.

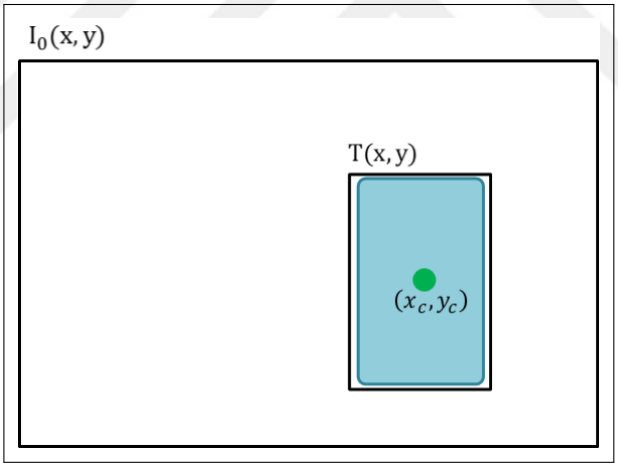

Figure 3.2. Toy image that illustrates the first image of the sequence.

There are three fundamental object representation methods in object tracking literature [21]. The first one is based on 2D array like image data which is used by NCC [23]. Similarly, the KCF method holds the object information as FFT of the template image. This kind of representation is vulnerable to a change in size of the target. Even if the target's shape may be consistent during the tracking sequence, its size may change due to camera or target's motion. This type of challenge can be overcome by increasing the search space of the tracker. Since it needs to search the target for different size values, computational

cost increases too. The second object representation method is to represent the target as a 1D histogram. The mean-shift tracker uses color histogram to represent the target to be tracked. However, this type of representation discards the geometric information. The third representation is using feature vectors. This representation requires matching of features in sequential images [2]. This is a computational power demanding process.

In this study, we benefit from our previous work on object detection in satellite images [68, 69]. There, we proposed a method which can be taken as the simplified form of implicit shape model [70]. Our method depends on feature (keypoint or corner) extraction from the image. Then, each keypoint (corner) votes for a possible object location. This way, the object to be detected is represented as a sample pdf in spatial domain. Modes of this pdf indicate possible objects in the image. In this study, we improve this method such that it fits to the object tracking framework. This allows us to model the object to be tracked with few parameters. Next, we will exploit this functionality to represent the target to be tracked in a given image sequence.

3.1.1. Preprocessing and Corner Extraction

We model the object to be tracked by its corner points in our method. The main advantage of this approach is that we do not have static shape-based model which needs to be updated throughout the tracking process. As for corner point extraction, we pick the Harris corner detector since it is easy to implement and provides fairly good performance for our tracking scenarios. To note here, we also tested other keypoint (corner) extraction methods such as KLT, SURF, and MSER. However, they did not give good results compared to the Harris corner detector as will be analyzed in Chapter 5.

Before extracting Harris corners, we have a preprocessing operation as modifying the image by an appropriate filter such as bilateral, sharpening, low-pass or median. The aim here is to help the corner extraction operation. Hence, we use the subwindow $T(x, y)$ after filtering as illustrated in Figure 3.3. In this figure, we illustrated the search region.

In Figure 3.3, we illustrated the target as a rounded rectangle box. However, in practice it may have a more complex shape. We then extract Harris corner points from the image. We provide the schematic representation of this operation in Figure 3.4. Here, red dots represent

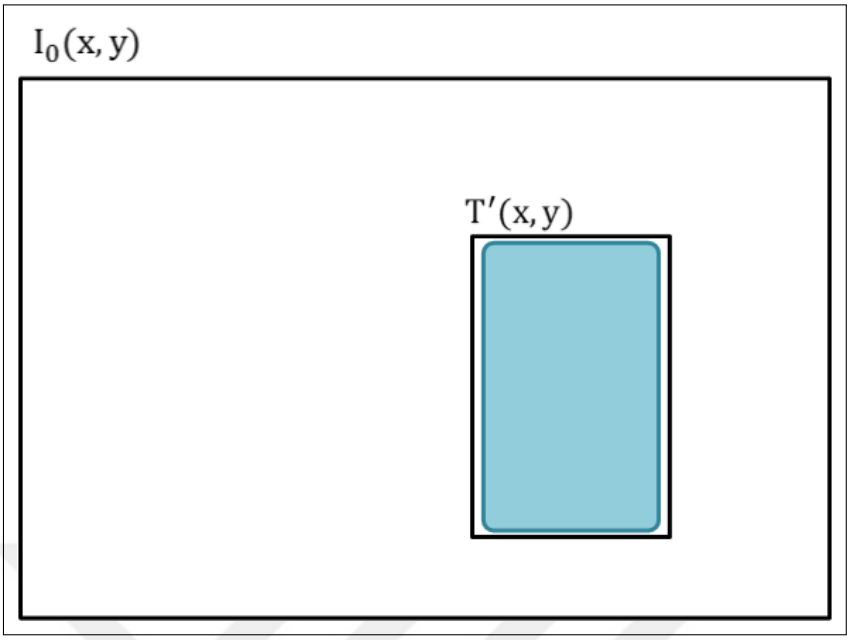

Figure 3.3. Image patch which includes the target to be tracked.

the extracted corner points.

Let's represent the extracted corners at time step k as $\boldsymbol{\alpha}_k^j = (x_j, y_j)$ where $j = 1, \cdots, N_k$ and N_k is the total number of extracted corner points from the subwindow $T(x, y)$ in the given time step. Here, (x_j, y_j) represent the spatial coordinate of the extracted corner point.

3.1.2. Graph-based Clustering

There may be more than one Harris corner point extracted for the actual object corner or multiple corner points may emerge very close to each other. Besides, there may be undesired corners due to noise and other sources in the image. Therefore, we apply a graph-based clustering method on all extracted corner points, $\boldsymbol{\alpha}_0^j$ for $j = 1, \cdots, N_k$, from the first frame.

Graph is a structure that consists of a finite set of vertices V and edges E represented as $G = (V, E)$. Within our graph, vertices are our corner points. Hence, we have $V = \{\boldsymbol{\alpha}_0^0, \cdots, \boldsymbol{\alpha}_0^{N_k}\}$. Edges, E, in the graph represent the link between vertices. For our graph, we have $E = \{(\boldsymbol{\alpha}_0^i, \boldsymbol{\alpha}_0^j)\}$ for $i, j = 1, \cdots, N_k$. Then, we obtain the adjacency matrix D summarizing the relation between vertices of the graph as

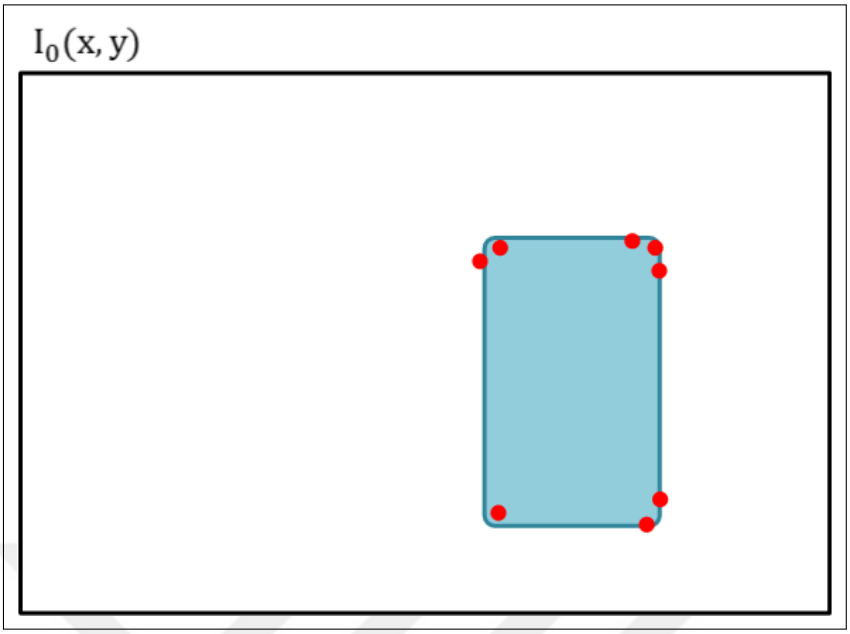

Figure 3.4. Extracted corner points from the image patch.

$$D = \begin{bmatrix} 0 & d_{12} & d_{13} & \cdots & d_{1N} \\ d_{21} & 0 & d_{23} & \cdots & d_{2N} \\ \vdots & & \ddots & & \vdots \\ d_{N1} & d_{N2} & d_{N3} & \cdots & 0 \end{bmatrix} \tag{3.1}$$

where $d_{ij} = ||\boldsymbol{\alpha}_0^i - \boldsymbol{\alpha}_0^j||$.

We threshold the adjacency matrix to obtain

$$D = \begin{bmatrix} 0 & a_{12} & a_{13} & \cdots & a_{1N} \\ a_{21} & 0 & a_{23} & \cdots & a_{2N} \\ \vdots & & \ddots & & \vdots \\ a_{N1} & a_{N2} & a_{n3} & \cdots & 0 \end{bmatrix} \tag{3.2}$$

where

$$a_{ij} = \begin{cases} 1, & \text{if } d_{ij} \leq \epsilon \\ 0, & \text{if } i = j \text{ or } d_{ij} > \epsilon \end{cases} \tag{3.3}$$

Here, ϵ is the threshold value that determines the spatial closeness of different corner point locations.

By thresholding, we obtain the modified adjacency matrix which we can extract subgraphs. Our subgraph representation is such that each subgraph member is connected to at least one of its other subgraph member. In other words, there is always a path between subgraph members. This way, we obtain S subgraphs represented by G^s and $G = \bigcup_{s=0}^{S} G^s$. Each subgraph represents a cluster. Hence, we obtain the graph based clustering this way. Due to the complex characteristics of the objects to be tracked, we set ϵ as 0.005 of the target's patch area which yields $S \approx 10$ for our test case. However, this number can be set by an automated method based on the object characteristics to be tracked. We provide the schematic representation of these operations in Figure 3.5. As can be seen in this figure, corners that are close to each other form subgraphs.

To note here, we do not form the graph representation $G = (V, E)$ besides the first frame in the image sequence. Instead, we use a Gaussian pdf to represent each cluster in subsequent images. Therefore, the proposed method does not have an excessive computation load in object tracking. We explain the Gaussian pdf based probabilistic object modeling next.

3.1.3. Probabilistic Object Modeling

We represent each cluster by a Gaussian pdf $\mathcal{N}(\boldsymbol{\mu}_0^s, \boldsymbol{\Sigma}_0^s)$ in the first frame in order to handle the uncertainty in the extracted corner points and clusters extracted from them. Here, $\boldsymbol{\mu}_0^s = (x_s, y_s)$ is the sample mean of the corners (vertex locations) and $\boldsymbol{\Sigma}_0^s$ is the covariance matrix of corners taken as

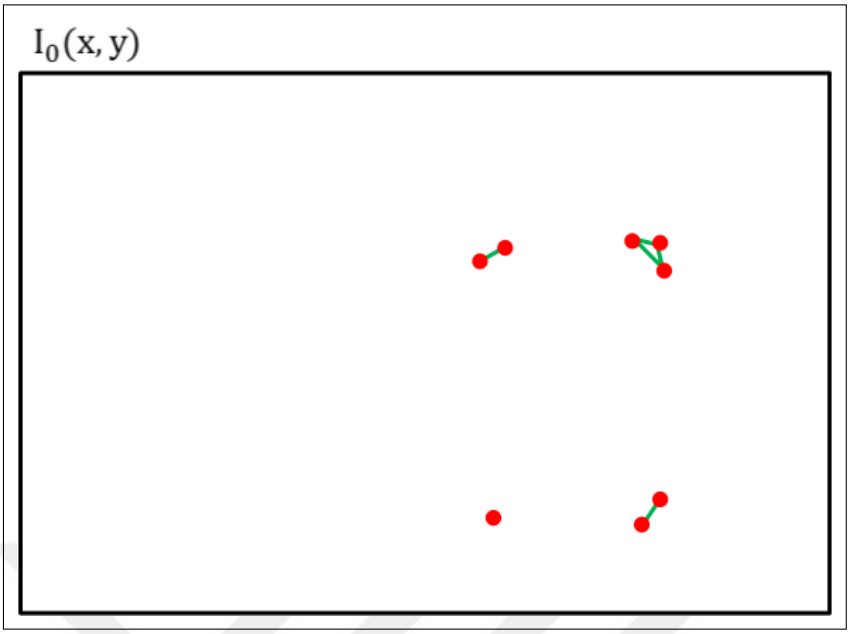

Figure 3.5. Subgraphs obtained from extracted corner points.

$$\mathbf{\Sigma}_0^s = \begin{bmatrix} \sigma & 0 \\ 0 & \sigma \end{bmatrix} \tag{3.4}$$

where $\sigma = \max(\sigma_x, \sigma_y)$ and σ_x and σ_y are the standard deviation of vertex locations of the subgraph G^s in x and y coordinates, respectively. We select $\mathbf{\Sigma}_0^s$ as in Eqn. 3.4 so that we have a symmetric Gaussian pdf. This allows us to have some degree of flexibility to object representation changes such as object rotation in the tracking process. We provide the schematic representation of this operation in Figure 3.6. Here, we represented each cluster location by a Gaussian pdf.

As we have the Gaussian pdf representing clusters, $G^s = \mathcal{N}(\boldsymbol{\mu}_0^s, \mathbf{\Sigma}_0^s)$ for $s = 1, \cdots, S$, we can represent the object to be tracked by the center point of the subwindow $T(x, y) = I_0(x_c \pm w/2, y_c \pm h/2)$. To do so, we ask the user to label the center point of the object to be tracked (x_c, y_c) in the first frame, as mentioned earlier. We represent this point with another Gaussian pdf as $\mathcal{N}(\boldsymbol{\mu}_0^c, \mathbf{\Sigma}_0^c)$ where $\boldsymbol{\mu}_0^c = (x_c, y_c)$. $\mathbf{\Sigma}_0^c$ is taken as the identity matrix in the first frame. At this stage, we associate G^s with the labeled object center point, by forming

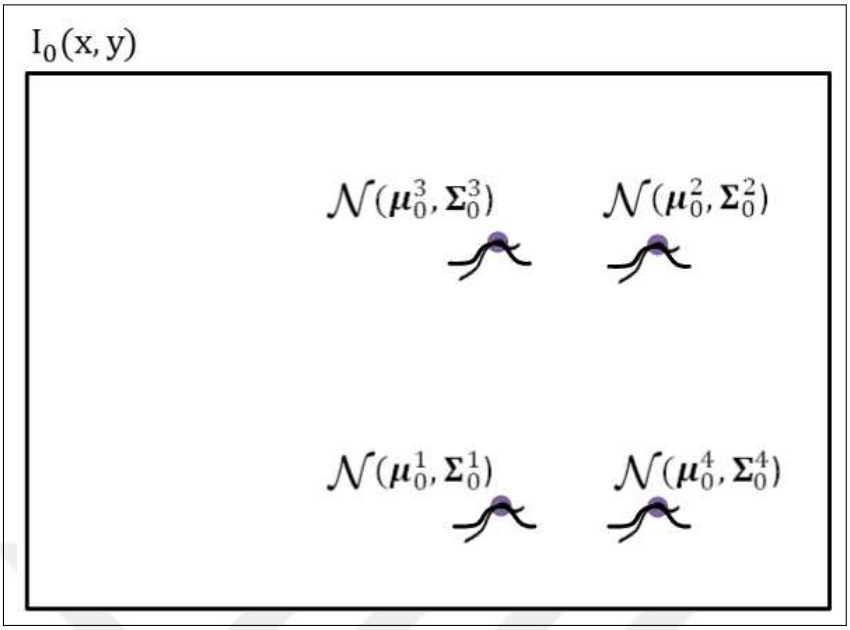

Figure 3.6. Representation of clusters with Gaussian pdf.

voting vectors between $\boldsymbol{\mu}_0^s$ for $s = 1, \cdots, S$ and $\boldsymbol{\mu}_0^c$. The vote vector of each cluster is then represented as $\boldsymbol{v}^s = \boldsymbol{\mu}_0^s - \boldsymbol{\mu}_0^c$ for $s = 1, \cdots, S$. We provide the schematic representation of this operation in Figure 3.7. In this figure, vote vector for each cluster is obtained by subtracting cluster's mean location from the object center location.

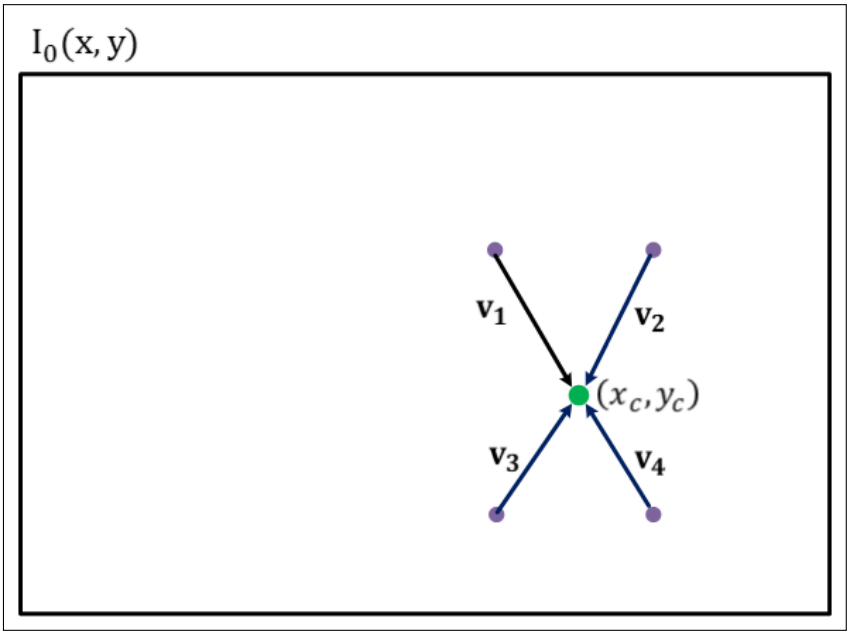

Figure 3.7. Vote vectors defined for each cluster.

Each cluster G^s is then represented by its vote vector $\boldsymbol{v}^s$, mean vector $\boldsymbol{\mu}_0^c$, and covariance matrix $\boldsymbol{\Sigma}_0^c$. Based on this representation, these clusters vote for the center of the object to be tracked. We also have $\mathcal{N}(\boldsymbol{\mu}_0^c, \boldsymbol{\Sigma}_0^c)$ to represent the object center. We fuse these two representations by multiplying the associated pdfs. As a result, we represent the object as a pdf in spatial (image) domain. To note here, we are calculating pdf values in discrete spatial domain. Hence, the pdf value also represents the corresponding probability value of that pixel. This is because of the definition in discrete random variables. Hence, we will use the probability and pdf naming interchangeably from this point on.

We can obtain the pdf of the object for a given set of clusters as

$$p(\mathbf{x}|G^s) = \mathcal{N}(\boldsymbol{\mu}^c, \boldsymbol{\Sigma}^c) \prod_{s=1}^{S} \mathcal{N}(\boldsymbol{\mu}^{s'}, \boldsymbol{\Sigma}^{s'}) \tag{3.5}$$

where $\boldsymbol{\mu}^c$ is the center point of the clusters. $\boldsymbol{\mu}^{s'}$ is the location which maximizes the probability indicated by a cluster. We can obtain this location as $\boldsymbol{\mu}^{s'} = \boldsymbol{v}^s + \boldsymbol{\mu}^s$, adding cluster vote vector to cluster mean location. We provide the schematic representation of this operation in Figure 3.8. In this figure, each cluster mean is translated according to its vote vector. We fit Gaussian to each translated location and also center of the clusters. Then, we multiply all these Gaussian pdfs as in Eqn. 3.5.

As mentioned earlier, this probabilistic representation originates from our previous work. The probabilistic object representation method proposed in this study is the enhanced version of our previous work since it is specifically formed to track objects in image sequences. Besides, it will be merged with the Bayesian object tracking framework in a synergistic way to be explained in detail in Section 3.4.

Till this point, we formed a structure to represent the object to be tracked in a probabilistic framework. Our aim here was to form a flexible yet powerful method such that the object can be tracked in a robust manner under various disturbances such as occlusion, object size and appearance change due to changing camera location, object color change due to lighting

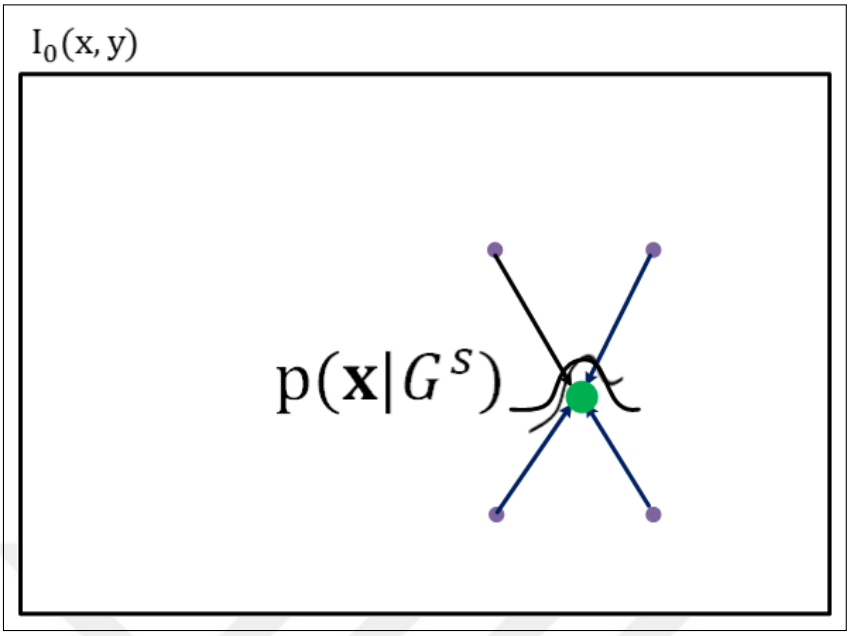

Figure 3.8. Obtaining the object pdf.

effects, and viewpoint change due to changing camera location.

With our probabilistic object representation, we keep very few parameters about the target. Yet, we can provide a generalized and flexible object model. This is the most powerful aspect of our approach. We can emphasize this by a simple example. Let's say the object to be tracked is given in a 100×100 image patch. If we want to represent our target with the patch directly, we need to hold 10000 parameters. Moreover, this patch needs to be updated according to changes in the sequence which also increases the computational load. Let's say that we chose the cluster size as 10 with our probabilistic object model. We can represent the same object with $10 \times 2 \times 1 \times 2 = 40$ parameters. This difference is highly crucial for object tracking implementations on hardware limited platforms such as microcontroller. One can also argue that, the object to be tracked can be represented with color histogram having 256 parameters. The first limitation here is selecting the proper bin size. More importantly, we discard the information about geometrical shape of the target if we follow this path.

3.2. BAYESIAN OBJECT TRACKING FRAMEWORK

The next step in our method is forming the object tracking structure. Here, we benefit from the Bayesian tracking framework based on a probabilistic recursive algorithm that enables

association of target locations through successive time steps. Since we have a probabilistic object representation as introduced in the previous section, it naturally fits to this framework. Before going further, we should explain the general Bayesian tracking framework. To do so, we extensively benefit from the seminal work of Arulampalam *et al.* [6] in the notation and description steps.

3.2.1. State Representation

We represent the target being tracked by a vector $\mathbf{x}_k$. This is called *state vector* of the target within the Bayesian tracking framework. We can form the state vector for our tracking method as

$$\mathbf{x}_k = \begin{bmatrix} x_k \\ y_k \\ \dot{x}_k \\ \dot{y}_k \\ \ddot{x}_k \\ \ddot{y}_k \end{bmatrix} \tag{3.6}$$

where (x_k, y_k) represent the target location, $(\dot{x}_k, \dot{y}_k)$ represent the velocity, and $(\ddot{x}_k, \ddot{y}_k)$ represent the acceleration at time step k in x and y coordinates, respectively.

3.2.2. Dynamic System and Measurement Models

We represent the state evolution by two recursive equations as dynamic system and measurement models within the Bayesian tracking framework. We have the *dynamic system model* as

$$\mathbf{x}_k = f_k(\mathbf{x}_{k-1}, \boldsymbol{w}_{k-1}) \tag{3.7}$$

where the function f_k relates the current and previous states of the target with an associated noise parameter, $\boldsymbol{w}_{k-1}$, called the process noise. This relation is obtained from the motion of the target.

At each time step, we also make measurements from the tracking operation. This is called the *measurement model* represented as

$$\mathbf{z}_k = h_k(\mathbf{x}_k, \mathbf{n}_k) \tag{3.8}$$

where the function h_k relates the state and measurement values of the target with uncertainty as associated noise parameter, $\mathbf{n}_k$, called measurement noise.

3.2.3. Probabilistic Representation

The Bayesian tracking framework represents the present state of the object in a probabilistic way based on previous state and measurement values. In other words, we frame the tracking problem as obtaining $p(\mathbf{x}_k|\mathbf{z}_{1:k})$. Here, we have probability of the target being tracked as $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ at each time step. To note here, we do not take any measurements in the first frame. Therefore, the time index in the measurement vector always starts with value 1 instead of 0 in our operations.

We predict the target's current state based on previous measurements $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \int p(\mathbf{x}_k|\mathbf{x}_{k-1})p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})d\mathbf{x}_{k-1} \tag{3.9}$$

where $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ can be calculated using the dynamic system model in Eqn. 3.8, and $p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1})$ is the posterior target state probability in the previous time step including the all the observations from time steps 1 to $k-1$. Then, $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is called the prior probability for the target being tracked for the time step k.

Since our ultimate goal is to estimate $p(\mathbf{x}_k|\mathbf{z}_{1:k})$, we can rewrite the prediction probability using Bayes' rule as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \frac{p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{z}_{1:k-1})p(\mathbf{x}_k|\mathbf{z}_{1:k-1})}{p(\mathbf{z}_k|\mathbf{z}_{1:k-1})} \tag{3.10}$$

Note that the denominator in Eqn. 3.10 is independent of the target state. Thus, it can be expressed as normalizing factor η. With Markov process assumption, the current measurement only depends on current state of the target. Hence, $p(\mathbf{z}_k|\mathbf{x}_k, \mathbf{z}_{1:k-1}) = p(\mathbf{z}_k|\mathbf{x}_k)$. This probability can be obtained from the measurement model. In Eqn. 3.10, $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is the prediction probability as in Eqn. 3.9. We can write the posterior probability of the target state $\mathbf{x}_k$ given all measurements up to and including time step k as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \eta p(\mathbf{z}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \tag{3.11}$$

We can associate the state of target being tracked between time steps by computing the prior probability $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ by using the dynamic system model and the likelihood probability $p(\mathbf{z}_k|\mathbf{x}_k)$ based on the measurement model. To note here, we can track the target solely based on our object representation as explained in Section 3.5. However, using the Bayesian framework improves the performance of our tracker. We will show this effect in Chapter 5.

3.3. MERGING PROBABILISTIC OBJECT REPRESENTATION AND BAYESIAN TRACKING FRAMEWORK

We can use our probabilistic object representation, introduced in Section 3.1, in the Bayesian tracking framework. To do so, we explain how our object representation fits in the dynamic system and measurement models. We also consider how prediction and update operations are done in the Bayesian framework in this section.

3.3.1. Our Dynamic System and Measurement Models

We can represent the dynamic system model in Eqn. 3.7 for our problem as follows. Due to characteristics of the motion of objects to be tracked, we can assume that the target state $\mathbf{x}_k$ evolves according to

$$\mathbf{x}_k = \mathbf{F}\mathbf{x}_{k-1} + \boldsymbol{\omega}_{k-1} \tag{3.12}$$

where $\mathbf{F}$ is obtained from the motion model.

The motion model is the essential part of tracking systems. There are different options for the motion model in literature for dynamic systems [71]. Generally, there are two motion model types as non-maneuvering and maneuvering target dynamic models. In our method, we assume that our target is not maneuvering in consecutive frames. Therefore, constant velocity or constant acceleration models are used for our tracker. The motion model is the mathematical representation of the motion of the target. One dimensional linear motion can be expressed as

$$distance = velocity \times time + \frac{1}{2} acceleration \times time^2 \tag{3.13}$$

Since the acceleration is second derivative of the distance, this is a second order differential equation. The constant velocity model represents this differential equation as matrix. For

the state representation given in Eqn. 3.6, the constant velocity model leads to

$$\mathbf{F} \triangleq \begin{bmatrix} 1 & 0 & \Delta & 0 & \frac{\Delta^2}{2} & 0 \\ 0 & 1 & 0 & \Delta & 0 & \frac{\Delta^2}{2} \\ 0 & 0 & 1 & 0 & \Delta & 0 \\ 0 & 0 & 0 & 1 & 0 & \Delta \\ 0 & 0 & 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 0 & 0 & 1 \end{bmatrix} \tag{3.14}$$

Here, Δ is the target sampling interval (time difference between two frames).

The process noise at time step k, $\boldsymbol{\omega}_k$, in Eqn. 3.7 is assumed to be multivariate Gaussian with zero mean and covariance matrix $\mathbf{Q}$ as

$$\mathbf{Q} \triangleq \rho \begin{bmatrix} \frac{\Delta^6}{36} & 0 & \frac{\Delta^5}{12} & 0 & \frac{\Delta^4}{6} & 0 \\ 0 & \frac{\Delta^6}{36} & 0 & \frac{\Delta^5}{12} & 0 & \frac{\Delta^4}{6} \\ \frac{\Delta^5}{12} & 0 & \frac{\Delta^4}{4} & 0 & \frac{\Delta^3}{2} & 0 \\ 0 & \frac{\Delta^5}{12} & 0 & \frac{\Delta^4}{4} & 0 & \frac{\Delta^3}{2} \\ \frac{\Delta^4}{6} & 0 & \frac{\Delta^3}{2} & 0 & \Delta & 0 \\ 0 & \frac{\Delta^4}{6} & 0 & \frac{\Delta^3}{2} & 0 & \Delta \end{bmatrix} \tag{3.15}$$

Here, ρ denotes the process noise parameter derived from jerk. We assume that we have perfect information about the target process (or motion) model in Eqn. 3.12.

We formulate the measurement model separately for object tracking using particle filter and Kalman filter. We will introduce these in detail later in this section.

3.3.2. Our Prediction Operations

For prediction operations, we first crop a search region from current image $I_k(\hat{x}_{k|k-1} \pm w'/2, \hat{y}_{k|k-1} \pm h'/2)$ where $(\hat{x}_{k-1|k-1}, \hat{y}_{k-1|k-1})$ is the target location estimate for time step

$k-1$. $(\hat{x}_{k|k-1}, \hat{y}_{k|k-1})$ is their propagated version based on Eqn. 3.12. Here, w' and h' are width and height of the search region.

Let $\mathbf{\Sigma}^s_{k-1|k-1}$ be the covariance matrix of each cluster at time step $k-1$. From $\boldsymbol{\mu}^s_{k-1|k-1}$, we then propagate the mean location of each cluster for time step k, $\boldsymbol{\mu}^s_{k|k-1}$ based on the process model in Eqn. 3.12. We assume that predicted covariance matrix of each cluster remains unchanged as $\mathbf{\Sigma}^s_{k|k-1} = \mathbf{\Sigma}^s_{k-1|k-1}$. To note here, we made this assumption because $\mathbf{\Sigma}^s_{k|k}$ has relatively small effect on association process where it will be used. This process will be explained later in the next section. This parameter may be updated based on scale change of the target. We provide the schematic representation of these operations in Figure 3.9. In this figure, we illustrated the propagation of the clusters according to predicted target location.

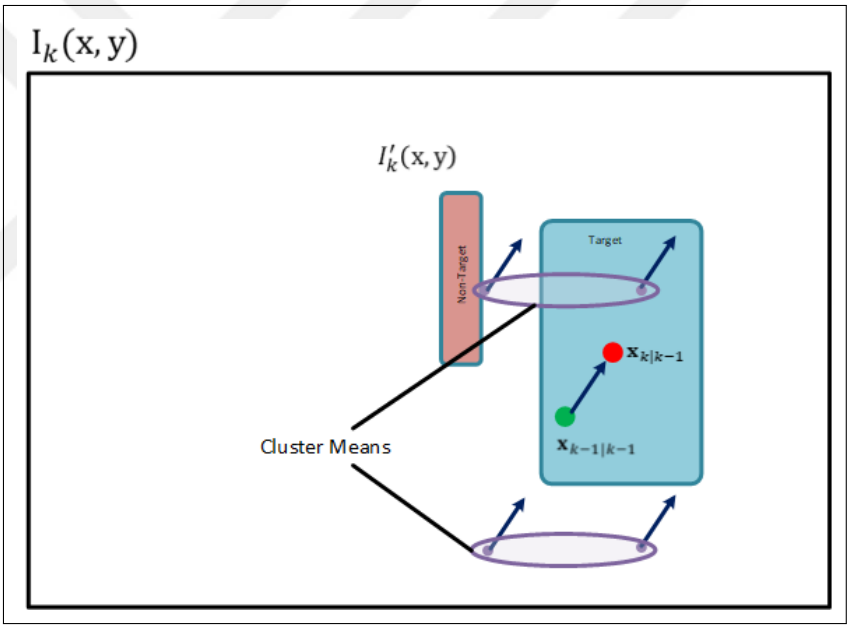

Figure 3.9. Predicted location calculation from the dynamic system model.

3.3.3. Our Update Operations

We obtain the corner points $\boldsymbol{\alpha}_k$ within $I_k(\hat{x}_{k|k-1} \pm w'/2, \hat{y}_{k|k-1} \pm h'/2)$ for the update operations during time step k. Then, we associate each new corner point to one of the propagated clusters. We exclude the outlier corner points from the extracted ones. Here, we take a corner point to be an outlier if it is 3σ away from the propagated cluster mean $\boldsymbol{\mu}^s_{k|k-1}$. We select the 3σ value as threshold since it covers the 99.7 percent of the Gaussian pdf. Hence, we obtain a set of corner points as $\boldsymbol{\alpha}_k = \{\boldsymbol{\alpha}^j_k\}^{N_k}_{j=1}$. We provide the schematic

representation of these operations in Figure 3.10. Here, we illustrated the association process of the corners to clusters. The corner point at middle left remain not associated because it is considered as outlier.

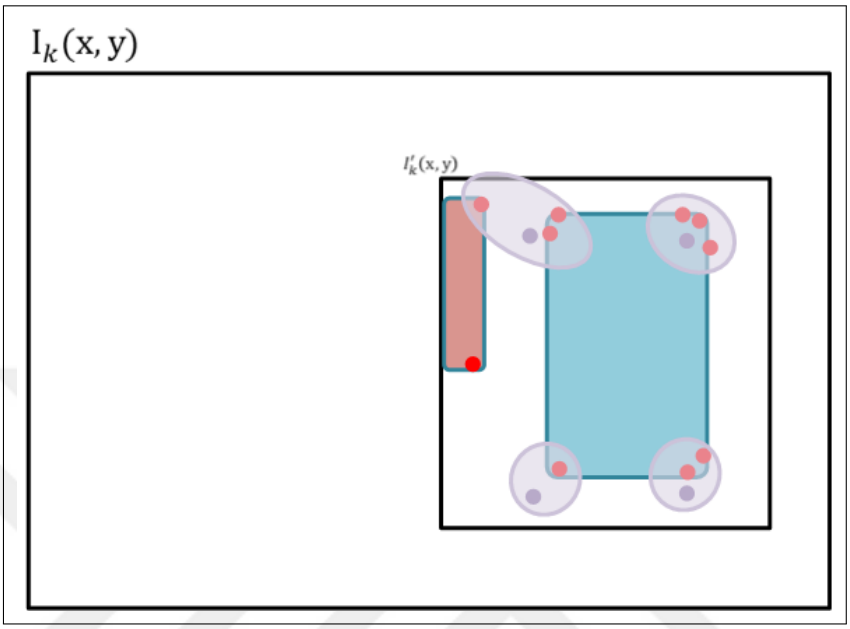

Figure 3.10. Association of corner points to propagated clusters.

We can rephrase the operations in Figure 3.10 as follows. $\boldsymbol{\alpha}_k^j$ is assigned to a cluster with respect to its distance to the propagated cluster mean $\boldsymbol{\mu}_{k|k-1}^s$. Each new corner $\boldsymbol{\alpha}_k^j$ then indicates the presence of the target at location $\mathbf{m}_k^j$ subject to the vote vector of its cluster $\mathbf{v}^s$. In other words, we have $\mathbf{m}_k^j = \boldsymbol{\alpha}_k^j + \mathbf{v}^s$ for $j \in \{1, 2, \cdots, N_k\}$. The set of all mean target locations pointed by the corners are represented as $\mathbf{m}_k = [m_k^1, m_k^2, \cdots, m_k^{N_k}]$. We provide the schematic representation of these operations in Figure 3.11. Here, we illustrate locations which maximize the target location probability of each corner point.

We use yet another indication of the target in order to improve measurement. To do so, we obtain the center point $\boldsymbol{\mu}_c$ from bounding box of corner points, $\mathbf{m}_k$, which were included in the association step. We provide the schematic representation of this operation in Figure 3.12. Here we draw a rectangle that surrounds the associated corner points and calculate the center of rectangle.

Given the unobserved target state $\mathbf{x}_k$, the first set of measurements is the corner points $\boldsymbol{\alpha}_k$. The second set of observations is the indicated target locations $\mathbf{m}_k$ based on the corner

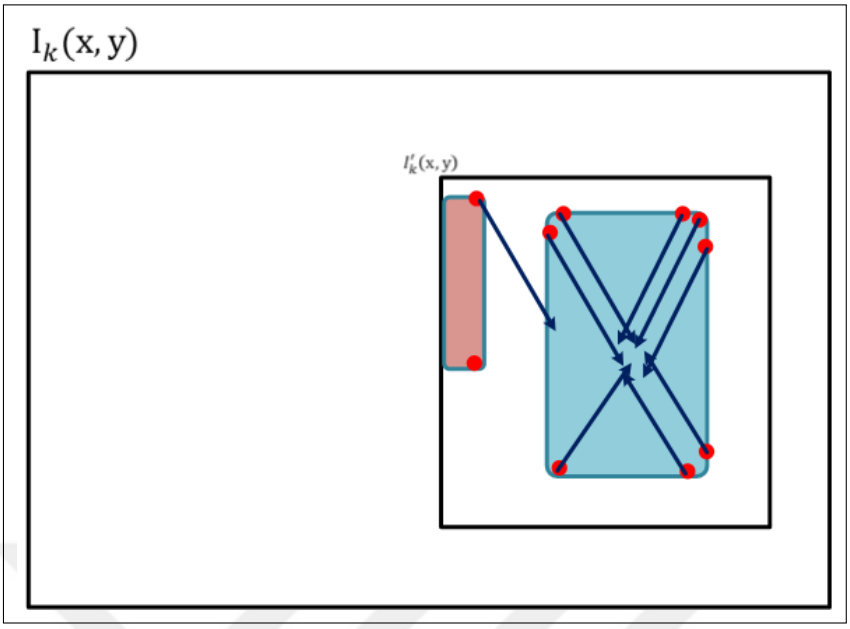

Figure 3.11. Indication of the target based on corner locations after association.

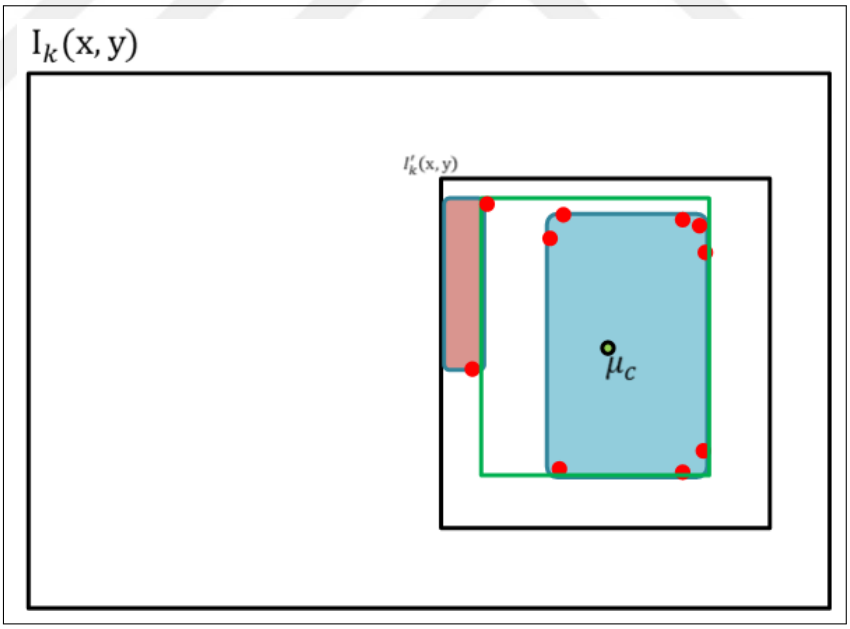

Figure 3.12. Center of the rectangle which surrounds the associated corner points.

points. We can then have an observed target location $\mathbf{z}_k$ from $\mathbf{m}_k$ and $\boldsymbol{\mu}_c$ where we define the mode of $p(\mathbf{m}_k, \boldsymbol{\mu}_c|\mathbf{x}_k)$ as the observed target location $\mathbf{z}_k$. As a result, we get the chain of observations from the unknown target state $\mathbf{x}_k$ from a single frame at time step k as $\mathbf{x}_k \rightarrow \boldsymbol{\alpha}_k \rightarrow \mathbf{m}_k, \boldsymbol{\mu}_c \rightarrow \mathbf{z}_t$.

We can then define the likelihood pdf $p(\mathbf{z}_k|\mathbf{x}_k) \approx p(\mathbf{m}_k, \boldsymbol{\mu}_c|\mathbf{x}_k)$ used in Eqn. 3.11 with these measurements at hand. To note here, we use the Eqn. 3.5 for probabilistic object representation. Here, we use the same relation to estimate target state. Assuming conditional independence,

$$p(\mathbf{m}_k, \boldsymbol{\mu}_c|\mathbf{x}_k) = p(\mathbf{m}_k|\mathbf{x}_k)p(\boldsymbol{\mu}_c|\mathbf{x}_k) \tag{3.16}$$

Here, $p(\mathbf{m}_k|\mathbf{x}_k)$ is the conditional pdf of corner locations for given target location. We can expand this pdf as $p(\mathbf{m}_k|\mathbf{x}_k) = p(m_k^1, m_k^2, \cdots, m_k^{N_k}|\mathbf{x}_k)$. Assuming conditional independence again, we will have

$$p(\mathbf{m}_k|\mathbf{x}_k) = \prod_{j=1}^{N_k} p(m_k^j|\mathbf{x}_k) \tag{3.17}$$

After the measurement is complete, we conduct the updating operation as in Eqn. 3.11. We obtain the prior pdf $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ as in Eqn. 3.9 and likelihood pdf $p(\mathbf{z}_k|\mathbf{x}_k)$ as in Eqn. 3.16. By multiplying these two pdfs, we obtain the target pdf $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ for time step k. We illustrate the pdfs before update step in Figure 3.13(a). We illustrate the final pdf after update step in Figure 3.13(b). We accept the mode of this pdf as location of the target being tracked.

In this study, we use $\mathbf{m}_k$ and $\boldsymbol{\mu}_c$ as our measurements from the target state $\mathbf{x}_k$ to estimate the target state $\hat{\mathbf{x}}_k$ since we have the proposal pdf $p(\mathbf{x}_k|\mathbf{x}_{k-1})$ which can be used for particle filtering. On the other hand, we use $\mathbf{z}_k$ as our observation in order to find the estimated

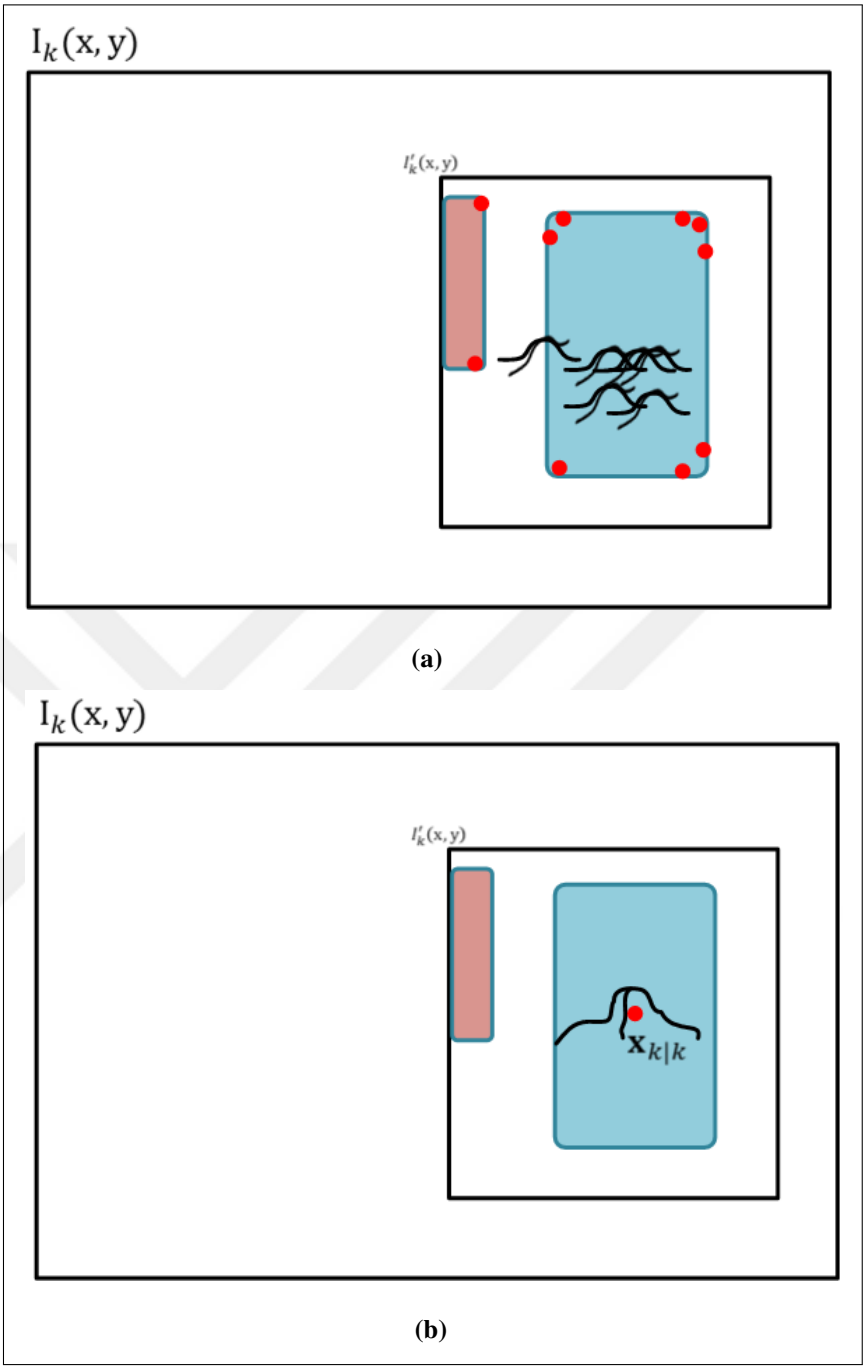

Figure 3.13. Illustration of our update operation. (a) Pdfs before the update step (b) Final pdf after the update step.

target location $\mathbf{x}_{k|k}$ in Kalman filtering. We will explain these in detail in Section 3.4.

3.3.4. Updating the Probabilistic Object Representation

Target properties such as shape, rotation, and scale may change during the tracking process. In order to adapt such changes, trackers generally update their object representations in two different ways as short- and long-term update. Short-term update is used to handle more temporal or small changes in the target such as partial occlusion. On the other hand, long-term update is used to cover more persistent or major changes such as scale or shape. Usually short-term update is applied at ever time step. On the other hand, long-term update is applied less frequently such as on every 10 or 20 steps.

For our tracker, we also define short- and long-term updates as well. We update the cluster means in every time step as for short-term update. We update the vote vectors which are associated to clusters in every 10 steps as for long-term update. Let's explain them in detail next.

We can explain the short-term update in our method as follows. After estimating the target location $\hat{\mathbf{x}}_k$ at time step k, we update the probabilistic representation of each cluster G^s as follows. We translate the target location in reverse direction of the cluster vector $\mu^{s'} = \hat{\mathbf{x}}_k - \mathbf{v}^s$ for each cluster. We then compute the distance of each corner located in G^s to $\mu^{s'}$ as $\lambda^{s,j} = \|\mu^{s'} - \boldsymbol{\alpha}_k^{s,j}\|$ where $\left\{\boldsymbol{\alpha}_k^{s,j}\right\}_{j=1}^{N_k^s}$ and N_k^s are the total number of corners located in G^s at time step k. Upon normalizing the weights for each cluster as

$$\lambda^{s,j} = \frac{\lambda^{s,j}}{\sum_{j=1}^{N_{s,k}} \lambda^{s,j}} \tag{3.18}$$

We update the mean cluster location $\boldsymbol{\mu}_{k|k}^s$ as

$$\boldsymbol{\mu}_{k|k}^s = \sum_{j=1}^{N_k^s} \lambda^{s,j} \boldsymbol{\alpha}_k^{s,j} \tag{3.19}$$

We provide the schematic representation of this operation in Figure 3.14. Here, purple points are obtained from back propagation of the object center for each cluster. Corner points are weighted according to distance to these purple points. Then, new cluster means become the weighted sum of the corner points.

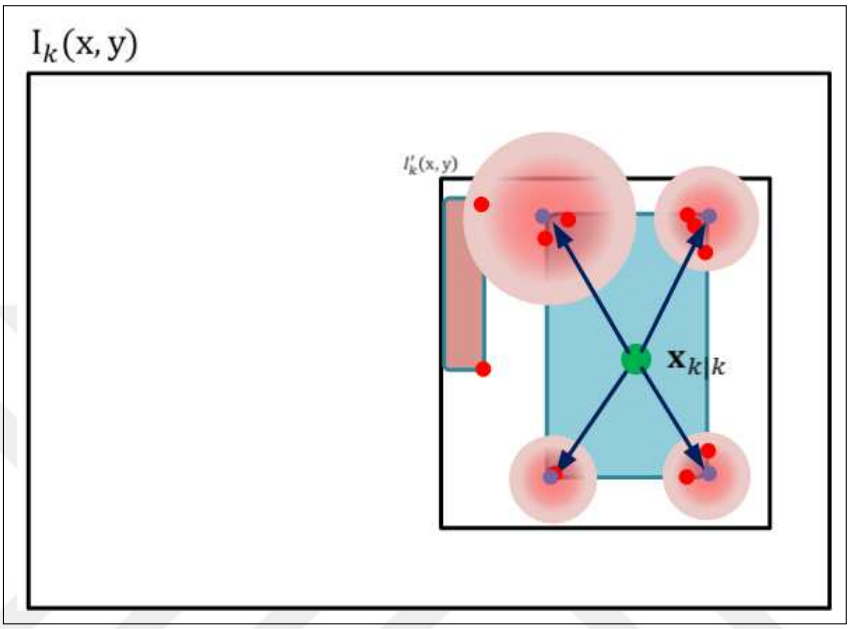

Figure 3.14. Updating cluster means.

We can explain the long-term update in our method as follows. We update the cluster vote vector $\mathbf{v}^s$ in every 10 frames. Vote vectors contain geometric information about the target. In a streaming video sequence, usually changes in physical shape of the target in consecutive frames are small. Therefore, we don't need to update vote vectors in every time step. Moreover, changing cluster vote vectors in every time step can cause to drift away of the tracker in some cases such that if there are too many redundant corner points in the background. Finally note that, we take precaution in case of drifting away from the cluster mean. If any corner location is not assigned to a cluster in the association step for three consecutive time steps, we relocate the cluster mean where closer to object center. To do so, we decrease the length of the cluster's vote vector to half.

3.4. APPROXIMATE BAYESIAN TRACKING METHODS

The merging operation introduced in Section 3.3 is applicable to the generalized Bayesian tracking framework. In this section, we simplify this operation using three approximation

methods within the Bayesian framework as grid, particle filter, and Kalman filter based. We explain implementation details for each method in detail next.

3.4.1. Grid Based Approximation

In practice, our search space is limited with image plane. Therefore we can approximate prediction, measurement and update pdfs with discrete states which corresponds to image pixel locations. Let's assume the state space at time $k-1$ consists of discrete states $\mathbf{x}_{k-1}^d, d = 1, \cdots N_d$ where $\mathbf{x}_{k-1}^d$ represent the pixel location and N_d is the total number of pixels.

Based on Arulampalam *et al.* [6], posterior pdf defined in Eqn. 3.11 at previous time step $k-1$ can be approximated as

$$p(\mathbf{x}_{k-1}|\mathbf{z}_{1:k-1}) = \sum_{d=1}^{N_d} w_{k-1|k-1}^d \delta(\mathbf{x}_k - \mathbf{x}_k^d) \tag{3.20}$$

where δ is the Dirac delta function and $w_{k-1|k-1}^d$ is the weight obtained from pdf value at given pixel location. Hence, we obtain the prior probability in Eqn. 3.11 by substituting Eqn. 3.20 into Eqn. 3.9

$$p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) = \sum_{d=1}^{N_d} w_{k|k-1}^d \delta(\mathbf{x}_k - \mathbf{x}_k^d) \tag{3.21}$$

Similarly, we can apply this procedure to the update step defined in Eqn. 3.11. Hence, we can obtain the update probability as

$$p(\mathbf{x}_k|\mathbf{z}_{1:k}) = \sum_{d=1}^{N_d} w_{k|k}^d \delta(\mathbf{x}_k - \mathbf{x}_k^d) \tag{3.22}$$

where,

$$w_{k|k-1}^{d} \triangleq \sum_{d'=1}^{N_d} w_{k-1|k-1}^{d'} p(\mathbf{x}_k^d | \mathbf{x}_{k-1}^{d'}) \tag{3.23}$$

$$w_{k|k}^{d} \triangleq \frac{w_{k|k-1}^{d} p(\mathbf{z}_k | \mathbf{x}_{k-1}^{d})}{\sum_{d'=1}^{N_d} w_{k|k-1}^{d'} p(\mathbf{z}_k | \mathbf{x}_k^{d})} \tag{3.24}$$

We accept the target location as highest weighted pixel obtained in Eqn. 3.22.

3.4.2. Particle Filter

For particle filtering, we consider the mean target location voted by each corner point $\mathbf{m}_k = [m_k^1, m_k^2, \ldots, m_k^{N_k}]^T$ as observations from the unknown target location $\mathbf{x}_k$ for the time step k. Here, N_k is the total number of corner points.

In the Bayesian framework, the minimum mean square error (MMSE) estimate of the target state at time step k is

$$\hat{\mathbf{x}}_{k,MMSE} = \int \mathbf{x}_k p(\mathbf{x}_k | \mathbf{m}_{1:k}) d\mathbf{x}_k \tag{3.25}$$

where $p(\mathbf{x}_k | \mathbf{m}_{1:k})$ is called the posterior probability of $\mathbf{x}_k$ given all the observations up to and including time step k, $\mathbf{m}_{1:k} = \{\mathbf{m}_1, \mathbf{m}_2, \cdots, \mathbf{m}_k\}$.

Using only the posterior probability $p(\mathbf{x}_k | \mathbf{m}_{1:k})$, we can also have the maximum a posteriori

(MAP) estimate of the target location at time step k as

$$\hat{\mathbf{x}}_{k,MAP} = \arg\max_{\mathbf{x}_k} p(\mathbf{x}_k|\mathbf{m}_{1:k}) \tag{3.26}$$

Furthermore, $p(\mathbf{x}_k|\mathbf{m}_{1:k})$ is conditioned as $p(\mathbf{x}_k|\mathbf{m}_{1:k}) \approx p(\mathbf{m}_k|\mathbf{x}_k)p(\mathbf{x}_k|\mathbf{m}_{1:k-1})$ to compute Eqn. 3.25 or 3.26. As a result, current observation only depends on the current target state $p(\mathbf{m}_k|\mathbf{x}_k, \mathbf{m}_{1:k-1}) = p(\mathbf{m}_k|\mathbf{x}_k)$. Here, $p(\mathbf{x}_k|\mathbf{m}_{1:k-1})$ is the prior probability of $\mathbf{x}_k$ given observations $\mathbf{m}_{1:k-1}$ up to and including time step $k-1$.

Here, we model $p(\mathbf{x}_k|\mathbf{m}_{1:k})$ with a total of N_Q particles $\mathbf{x}_k^q$ with associated weights w_k^q as

$$p(\mathbf{x}_k|\mathbf{m}_{1:k}) = \sum_{q=1}^{N_Q} w_k^q \delta\left(\mathbf{x}_k - \mathbf{x}_k^q\right) \tag{3.27}$$

Due to sifting property, MMSE estimate of the target becomes

$$\hat{\mathbf{x}}_{k,MMSE} = \int \mathbf{x}_k \left(\sum_{q=1}^{N_q} w_k^q \delta\left(\mathbf{x}_k - \mathbf{x}_k^q\right) \right) d\mathbf{x}_k \tag{3.28}$$

$$= \sum_{q=1}^{N_q} w_k^q \mathbf{x}_k^q \tag{3.29}$$

Likewise, MAP estimate of the target becomes

$$\hat{\mathbf{x}}_{k,MAP} = \arg\max_{\mathbf{x}_k^q} w_k^q \quad q \in \{1, 2, \ldots, N_Q\} \tag{3.30}$$

We represent the uncertainty of each corner point around its voting direction with a Gaussian pdf with mean $\mathbf{m}_k^j = \boldsymbol{\alpha}_k^j + \mathbf{v}^s$ and covariance matrix $\boldsymbol{\Sigma}_{k|k-1}^j$ at time step k. Here, we assume that $E[\mathbf{m}_k^j] = \mathbf{x}_k$. We further assume $\boldsymbol{\Sigma}_{k|k-1}^j = \boldsymbol{\Sigma}_s$ to be fixed for all the corner points within cluster s, namely $j \in \{1, 2, \cdots, N_k^s\}$ as in Eqn. 3.4. Along with observations $\mathbf{m}_k$, we further define a rectangle which covers all the corners $\{\boldsymbol{\alpha}_k^j\}_{j=1}^{N_t}$ in the subwindow. Then, we represent the center of the rectangle as $\boldsymbol{\mu}_c$. We also use $\boldsymbol{\mu}_c$ as a measurement along with the pointed target locations $\mathbf{m}_k$. The measurement pdf now becomes

$$p(\mathbf{m}_k, \boldsymbol{\mu}_c|\mathbf{x}_k) = p(\mathbf{m}_k|\mathbf{x}_k)p(\boldsymbol{\mu}_c|\mathbf{x}_k) \tag{3.31}$$

by using the conditional independence assumption. Here, $p(\mathbf{m}_k|\mathbf{x}_k)$ is given in Eqn. 3.32 and $p(\boldsymbol{\mu}_c|\mathbf{x}_k)$ is assumed to have a Gaussian pdf as $\mathcal{N}(\boldsymbol{\mu}_c, \boldsymbol{\Sigma}_c)$. For simplification, we abuse the notation and call $p(\mathbf{m}_k, \boldsymbol{\mu}_c|\mathbf{x}_k)$ with only $p(\mathbf{m}_k|\mathbf{x}_k)$ while stating all the propagation and update equations. Finally, we model the joint pdf of $\{\mathbf{m}_k^j\}_{j=1}^{N_k}$ given the target location $\mathbf{x}_k$ as

$$\begin{aligned} p(\mathbf{m}_k|\mathbf{x}_k) = \frac{1}{\gamma} \exp\left\{\frac{1}{2}\left(\mu_c - \mathbf{x}_k\right)^T {\boldsymbol{\Sigma}_c}^{-1} \left(\mu_c - \mathbf{x}_k\right)\right\} \\ \prod_{j=1}^{N_k} \exp\left\{\frac{1}{2}\left(\mathbf{m}_k^j - \mathbf{x}_k\right)^T \boldsymbol{\Sigma}^{s-1} \left(\mathbf{m}_k^j - \mathbf{x}_k\right)\right\} \end{aligned} \tag{3.32}$$

where γ is the scaling parameter to ensure $p(\mathbf{m}_k|\mathbf{x}_k)$ is a valid pdf.

Since we use a set of particles $\mathbf{x}_k^q$ with their associated weights w_k^q to approximate $p(\mathbf{x}_k|\mathbf{m}_{1:k})$, we determine the weight of each particle according to Eqn. 3.31 as

$$w_k^q \propto p(\mathbf{m}_k|\mathbf{x}_k^q) \tag{3.33}$$

The Algorithm 3.1 then shows the brief generic particle filtering implementation as mentioned in [6]. In this work, we picked the MAP estimate rather than MMSE estimate. Because in final form, mode of the pdf which is the location that maximize the probability is target location. MMSE estimate may result in another location different than the mode if there is an accumulation of the corner points which do not belong the target. Finally, the resampling step dismisses the particles which have zero weight and resets the weights of significant particles back to N_Q^{-1}.

Algorithm 3.1. SIR based Particle Filtering for Target Tracking

Set $k = 1$. Generate initial particles for $k = 0$ from a known distribution $\mathbf{x}_0^q \sim p(\mathbf{x}_0)$ with $\forall q \ , w_0^q = N_Q^{-1}$.

while $k \leq T_S$ **do**

Propagate particles $\mathbf{x}_k^q = \mathbf{F}\mathbf{x}_{k-1}^q$ and $p(\mathbf{x}_k|\mathbf{m}_{1:k-1}) \approx \frac{1}{N_q}\sum_{q=1}^{N_q}\delta(\mathbf{x}_k - \mathbf{x}_k^q)$.

Receive observations, $\mathbf{m}_k = \{\mathbf{m}_k^1, \mathbf{m}_k^2, \ldots, \mathbf{m}_k^{N_k}\}$.

$w_k^q \propto p(\mathbf{m}_k|\mathbf{x}_k^q)$. (Updating weights based on Eqn. 3.31),

$w_k^q = \frac{w_k^q}{\sum_{j=1}^{N_q} w_k^j}$. (Normalizing weights).

$\hat{\mathbf{x}}_{k,MAP} = \arg\max_{\mathbf{x}_k^q} w_k^q \quad q \in \{1, 2, \cdots, N_S\}$ (MAP estimate).

$\{\mathbf{x}_k^q, N_Q^{-1}\} = \text{Resampling}(\mathbf{x}_k^q, w_k^q)$

$k = k + 1$

end while

3.4.3. Kalman Filter

We generate the observed target location $\mathbf{z}_k$ from the mode of the pdf $p(\mathbf{m}_k|\mathbf{x}_k)$ and determine the state estimate $\hat{\mathbf{x}}_{k|k}$ using the propagation and update properties of the Kalman filter. In other words, we fit a single multivariate Gaussian to Eqn. 3.32 as

$$p(\mathbf{m}_k|\mathbf{x}_k) \approx p(\mathbf{z}_k|\mathbf{x}_k) = \frac{\exp\left\{-\frac{1}{2}\left(\mathbf{z}_k - \mathbf{H}\mathbf{x}_k\right)^T \mathbf{R}_k^{-1}\left(\mathbf{z}_k - \mathbf{H}\mathbf{x}_k\right)\right\}}{(2\pi)^2|\mathbf{R}_k|^{1/2}} \tag{3.34}$$

for Kalman filtering. Here, $\mathbf{z}_k$ is called the observed target location and represents the pixel which maximizes $p(\mathbf{m}_k|\mathbf{x}_k)$. We assume $E[\mathbf{z}_k] = \mathbf{x}_k$ and approximate the uncertainty of $p(\mathbf{m}_k|\mathbf{x}_k)$ with a single measurement error covariance matrix as $\mathbf{R}_k$. The measurement error $\mathbf{n}_k$ between the observed target location $\mathbf{z}_k$ and the actual target location $\mathbf{x}_k$ is represented as $\mathbf{n}_k = \mathbf{z}_k - \mathbf{H}\mathbf{x}_k$. This yields the measurement model

$$\mathbf{z}_k = \mathbf{H}\mathbf{x}_k + \mathbf{n}_k \tag{3.35}$$

For our tracking problem, we have

$$\mathbf{H} = \begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \end{bmatrix} \tag{3.36}$$

We further assume that the measurement error $\mathbf{n}_k$ follows multivariate Gaussian pdf with zero mean and covariance matrix, $\mathbf{R}_k$ as described in Eqn. 3.34.

Let $\hat{\mathbf{x}}_{k|k}$ be the a posteriori state estimate of the unknown state $\mathbf{x}_k$ at time step k given observations up to and including time step k and $\mathbf{P}_{k|k}$ be the a posteriori estimate covariance matrix.

Prediction and update steps of the Kalman filter using the process and measurement models given in Eqns. 3.12 and 3.35 are formulated as follows. Let $\hat{\mathbf{x}}_{k|k-1}$ be the predicted (a priori) state estimate of the unknown state $\mathbf{x}_k$ and $\mathbf{P}_{k|k-1}$ be the predicted estimate covariance. Then, $\hat{\mathbf{x}}_{k|k-1}$ and $\mathbf{P}_{k|k-1}$ are determined from $\hat{\mathbf{x}}_{k|k}$ and $\mathbf{P}_{k|k}$ according to

$$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k-1|k-1} \tag{3.37}$$

$$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k-1|k-1}\mathbf{F}^T + \mathbf{Q} \tag{3.38}$$

where $\mathbf{Q}$ is the process noise covariance matrix given as in Eqn. 3.15 . Given that the observed target location is available at time step k, $\mathbf{z}_k$, update step evolves according to

$$\tilde{\mathbf{y}}_k = \mathbf{z}_k - \mathbf{H}\hat{\mathbf{x}}_{k|k-1} \tag{3.39}$$

$$\mathbf{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}_k \tag{3.40}$$

$$\mathbf{K}_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1} \tag{3.41}$$

$$\hat{\mathbf{x}}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k \tag{3.42}$$

$$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{L}_k\mathbf{H})\mathbf{P}_{k|k-1} \tag{3.43}$$

where $\tilde{\mathbf{y}}_k$ is the innovation residual, $\mathbf{S}_k$ is the innovation covariance, $\mathbf{K}_k$ is the Kalman gain. Then, $\hat{\mathbf{x}}_{k|k}$ and $\mathbf{P}_{k|k}$ represent the updated (a posteriori) state estimate and a posteriori estimate covariance, respectively.

Kalman filter is initialized with $\hat{\mathbf{x}}_{0|0}$ and its covariance matrix $\mathbf{P}_{0|0}$. We determine the predicted target location and its covariance matrix as in Eqn. 3.37. Upon having the observation $\mathbf{z}_k$, we execute the update step in Eqn. 3.39.

3.5. TRACKING BY DETECTION ONLY

In the absence of a motion model (or prior information), the unknown target state $\mathbf{x}_k$ is estimated by maximizing $p(\mathbf{m}_k|\mathbf{x}_k)$ as

$$\hat{\mathbf{x}}_{k,ML} = \arg\max_{\mathbf{x}_k} p(\mathbf{m}_t|\mathbf{x}_k) \tag{3.44}$$

where $\hat{\mathbf{x}}_{k,ML}$ is called the maximum likelihood (ML) estimate of $\mathbf{x}_k$.

Here, we partition the region of interest into discrete states $\mathbf{x}_k^r$ where $r \in \{1, 2, \cdots, N_P\}$. Note that such discrete states do not change with time since we do not use the dynamic system model in Eqn. 3.12. In our application, N_P represents the total number of pixels in one frame. Then, the target is localized from the frame received at time step k as

$$\hat{\mathbf{x}}_{k,ML} = \arg\max_{\mathbf{x}_k^r} p(\mathbf{m}_k|\mathbf{x}_k^r) \quad r \in \{1, 2, \cdots, N_P\} \tag{3.45}$$

where $p(\mathbf{m}_k|\mathbf{x}_k^r)$ is computed according to Eqn. 3.8.

4. EMBEDDED IMPLEMENTATION

We run our tracker on an embedded platform, more specifically on a microcontroller. Hence, it is the hardware limitation that will determine the real-time performance of the tracker. Therefore, selected hardware setup should be efficient, cost effective, and capable of fulfilling real-time constraints. Besides, the software implementation should be optimized for the selected hardware. This chapter covers the selected hardware setup and software optimization steps. We will also discuss implementation details of the proposed tracker. Therefore, we will first introduce the hardware components of our system as processor and camera. Then, we will explain the important points of software implementation in detail.

4.1. HARDWARE

There are many hardware options as for the embedded platform on the market for a given application. We can group these into three categories. As the first group, there are high level processors such as Broadcom Arm Cortex A53 or Samsung Exynos5 Octa. These are available on boards like Raspberry Pi 3 B+ or Odroid XU4. These processors have high processing capability. Some of them contain on-chip graphics processor as in Exnynos5 Octa with Mali GPU provided by ARM. Likewise, NVIDIA's Jetson development boards contain GPUs. However, high processing power comes with high power consumption.

The second group consists of microcontrollers offering compact size, low footprint, and low power consumption. However, they have lower processing power compared to the previous category. ARM Cortex-M series microcontrollers are typical example for this group. Cortex-M has a variety of solutions for different applications. For example, the Cortex-M0 based core is targeted for low power consumption. Cortex-M4 based microcontrollers offer cheap and battery friendly performance. The Cortex-M7 is upgraded architecture of Cortex-M4 processors in clock speed and memory. Cortex-H7 microcontrollers are based on Cortex-M7 core. They offer more processing power but Cortex-M7 processors are cheaper. ST Microelectronics produces STM32F4 series microcontrollers based on Cortex-M4 core and STM32F7 series microcontrollers based on Cortex-M7 core. NXP semiconductors produce MX RT1050 and RT1060 series microcontrollers based on Cortex-M7 core. The

current consumption of mentioned processors and microcontrollers is tabulated in Table 4.1.

Table 4.1. Average current and power consumption of processors.

Processor	**Clock Speed (MHz)**	**Current Consumption (mA)**	**Power Consumption (mW)**
Broadcom BCM2837 (Idle)	1200	350	1750
Broadcom BCM2837 (Full Load)	1200	980	4900
Exynos5 Octa (Average)	1800	750	3750
MX RT 150	600	75	225
STM32F7	216	178	587
STM32F4	180	44	145

FPGAs are the third group. They differ from the previous two categories in hardware level. Thus, their design procedure is totally different. Hence, they are out of scope of this study.

In a typical UAV system, flight controller card employs a Cortex-M4 based microcontroller. If a vision task needs to be done on the UAV, a companion computer accompanies the flight controller card since most operations require heavy processing power such as stereo vision or visual tracking.

We aim to implement our visual tracking method on a microcontroller. Hence, the companion computer can be discarded. As a result, significant room and power can be saved in the limited UAV platform. The proposed tracker can be run on either Cortex-M4 or Cortex-M7 platform if memory is available. Cortex-M4 processors are significantly smaller and have less current consumption compared to Cortex-M7 devices. However, a Cortex-M7 device is selected in this study in order to meet real-time constraints. Since STM32F746 Discovery board is suitable and budget friendly, we selected it rather than NXP MX RT 1050.

We selected two main components as STM32F746 Discovery board (as processing platform)

and STM32F4DIS-CAM (as image source). The STM32F746 Discovery board is a development platform for ARM Cortex-M7 core based STM32F746NGH6 microcontroller. Cortex-M7 series microcontrollers are superior in terms of clock speed and memory in ARM Cortex-M family. Therefore, they are more suitable for tasks requiring more computation power such as image processing. STM32F4DIS-CAM is a digital camera module which is designed by Embest. It is compatible with many development boards. It contains an OV9655 chip which is a 1.3 megapixel CMOS SXGA image sensor.

We provide the hardware block diagram of our tracker in Figure 4.1. Digital Camera Interface (DCMI) module will be used for grabbing incoming image. I^2C module will provide an access to the camera chip for configuring camera registers which contain operation settings of the camera. SDRAM is an external memory chip that is mounted on board. The image will be stored in this memory area. SDRAM will be accessed through the memory controller module. Results can be displayed using LCD module. These modules will be explained in following sections.

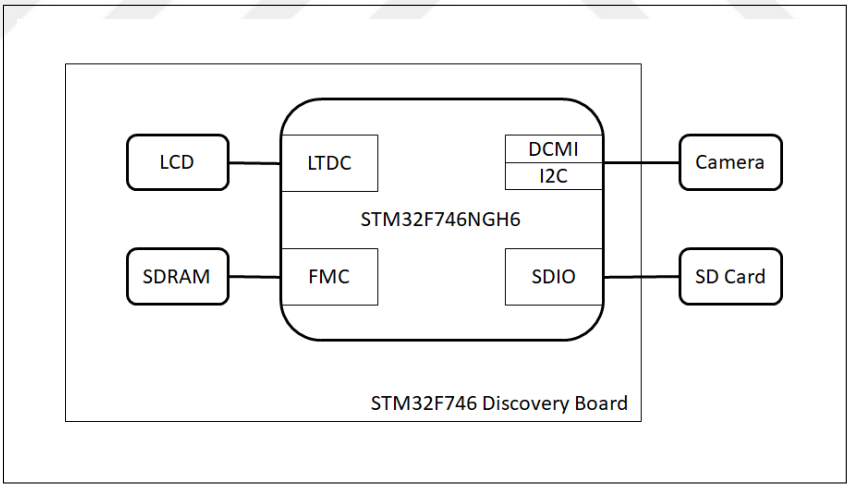

Figure 4.1. Hardware block diagram.

4.1.1. Microcontroller

STM32F746NGH6 is a microcontroller based on ARM Cortex-M7 core developed by ST Microelectronics. This microcontroller has several peripheral units such as DCMI, four I^2Cs, SDMMC, four UARTs, three 12-bit ADCs, two 12-bit DACs, 8- to 14-bit digital

camera module interface, internal 320+16+4-Kbyte SRAM and 1-Mbyte Flash memory, USB HS OTG, USB FS OTG, Ethernet MAC, FMC interface, Quad-SPI interface. In this section, we will explain main modules that we use for our tracker.

4.1.1.1. Camera Module

There is a camera module on the STM32F746 Discovery board that enables communication between microcontroller and digital camera chips using dedicated connector. This module consists of DCMI module, I^2C module, 24 MHz oscillator clock and GPIO pins. Schematic of camera connector is given in Figure 4.2 [72].

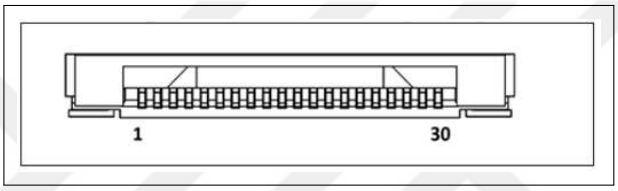

Figure 4.2. Camera connector schematic.

The DCMI module is able to connect with camera modules through 8-, 10-, 12- or 14-bit parallel data bus to receive image data. It can sustain a data transfer rate up to 54 Mbytes per second. DCMI module supports data formats such as monochrome, RGB565, YCbCr422. User may run this module in continuous or snapshot modes. It also has capability to crop image automatically. This module can generate interrupts when a line or a frame is acquired.

The camera clock is set to 24 MHz that is connected to on-board oscillator to feed camera. DCMI_SDA, DCMI_SCL are I^2C module pins. They are used to access camera registers in order to configure its operation. DCMI_PWR_EN and DCMI_NRST pins are GPIO pins that are used for holding camera in standby mode and hardware resetting camera respectively.

4.1.2. STM32F746 Discovery Board

The STM32F746 Discovery board is a development platform equipped with various hardware peripherals such as USB OTG HS, USB OTG FS, 10/100-Mbit Ethernet, micro SD card, USART, SAI Audio DAC stereo with audio jack input and output, MEMS digital microphones, SDRAM, Quad-SPI Flash memory, 4.3” color LCD-TFT with a

capacitive multi-touch panel. The board also contains Arduino Uno V3 connectors which make it possible to easily connect extension shields. An embedded in-circuit debugger and programmer is provided by the integrated ST-LINK/V2-1 for the STM32F746NGH6 microcontroller.

4.1.2.1. LCD

STM32F746 Discovery board employs a 4.3" 480×278 resolution LCD display on it. The STM32F746NGH6 microcontroller has a specific module named LCD TFT Display Controller (LTDC) to use this display. This module transfers image from a predefined address to the LCD in a fast manner. It provides 24-bit parallel digital RGB signals and delivers them directly to LCD up to XGA (1024×768) resolution. LTDC has two layers to show two images at the same time. These two images (in two separate memory locations) can be displayed either separately or in combined form using these layers and each layer can be configured for eight different color formats.

4.1.2.2. SDRAM

There is a 128-Mbit SDRAM memory chip on the STM32F746 Discovery board. We will benefit from it in image processing and LCD display applications. Data transfer to the SDRAM is controlled by the Flexible Memory Controller (FMC) module in the STM32F746NGH6 microcontroller.

4.1.2.3. SD Card

There is a micro SD card socket on the STM32F746 Discovery board. We used this module to access dataset images for tracker testing. FatFS library is used to store images in SD card [73].

4.1.3. Camera

STM32F4DIS-CAM board is selected as image acquisition module. This board employs OV9655 image sensor chip by OmniVision. We will next provide detailed information about internal functionality of this chip. OV9655 includes SXGA (1280×1024) resolution color image sensor and image processor. It supports various formats and resolutions in 8-bit or 10-

bit parallel bus. OV9655 can operate up to 15 frames per second (fps) for SXGA or 30 fps for VGA (640×480) resolution. This chip offers image processing functions such as exposure control, gamma, white balance, color saturation, hue control, white pixel canceling. Image formats and image processing functions can be configured through Serial Camera Control Bus (SCCB) interface [74].

The OV9655 image sensor supports several output formats. For RGB565 pixel format, one pixel is represented with 16 bits, distributed as 5 bits for red value, 6 bits for green value and 5 bits for blue value. Distribution of these bits is given in Table 4.2.

Table 4.2. RGB565 output format.

Pixels	**Bytes**	**D7 to D0**
Pixel	Even	R7 R6 R5 R4 R3 G7 G6 G5
	Odd	G4 G3 G2 B7 B6 B5 B4 B3

OV9655 is capable of giving output in YUV422 format. Distribution of bits is given in Table 4.3. In this format, each pixel has individual 8-bit Y value. Two consecutive pixels share the same U and V values. The advantage of this format is that, the Y value corresponds to illumination value. Hence, the grayscale image can be retrieved directly from camera by reading Y value of the stream.

4.2. SOFTWARE

The proposed method is implemented on two different platforms. The first platform is the PC with Intel Core i7 processor. The second platform is the microcontroller with ARM Cortex-M7 core. In this section, we will introduce our development environment and software implementation details for the PC and microcontroller.

4.2.1. Development Environment

On the PC side, MATLAB is used as the main software development tool. It includes all main computer vision algorithm implementations and additional components either hand

Table 4.3. YUV422 output format.

Pixels	Bytes	D7 to D0
Pixel 1	1^{st}	Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
	2^{nd}	U7 U6 U5 U4 U3 U2 U1 U0
	4^{th}	V7 V6 V5 V4 V3 V2 V1 V0
Pixel 2	3^{rd}	Y7 Y6 Y5 Y4 Y3 Y2 Y1 Y0
	2^{nd}	U7 U6 U5 U4 U3 U2 U1 U0
	4^{th}	V7 V6 V5 V4 V3 V2 V1 V0

coded or imported from file exchange community [75]. Comparison of all trackers are implemented on MATLAB as well. For the microcontroller environment, we used Atollic TrueSTUDIO provided by ST Microelectronics.

4.2.2. Embedded Software Packages

On the microcontroller side, there is no compact library for image processing. Therefore, all of the required functions are hand coded. ST Microelectronics provides device firmware package for their products. It contains HAL libraries which is low-level drivers for microcontrollers and board support packages (BSP). In our tracker implementation, HAL library is used to access and configure device peripherals such as DCMI, LTDC etc. We used BSP provided for STM32F746 Discovery board in order to access board components such as LCD, SDRAM, SD card.

4.2.3. CMSIS Library

While implementing tracker components, heavy mathematical and vector operations are used. ARM provides such a library named Cortex Microcontroller Software Interface Standard (CMSIS) [76]. It is an extensive software package aimed for Cortex-M based devices. CMSIS-DSP package in the library is used to implement vector manipulation, matrix operations, statistical calculations and fast mathematical functions. We used this library for matrix, vector multiplications and FFT calculations.

4.3. TRACKER IMPLEMENTATION

We will explain details of software implementation of our tracker in this section. Hence, it is divided into three parts. In the first part, initialization procedure of the tracker is explained. In the second part, we will explain implementation in details. In the third part, implementation details of the proposed tracking method with measurement is explained.

4.3.1. Initialization

We need to initialize our tracker according to the target before running. For online trackers, it is assumed that location and the bounding box of the target is known in the first frame. These adjustments can be made for the use case for some parameters. Other parameters can be set based on model image of the target for every tracking sequence. Adjusted parameters are as follows.

Almost every image feature extraction method has some degree of quality measure in order to distinguish a feature location. This value directly effects the number of features which will be extracted. Usually, this value is adjusted according to operation condition of the application.

4.3.2. Object Representation

Pseudo code for object representation algorithm is given in Algorithm 4.1. Implementation of this procedure's structure is same for both the PC and microcontroller. Some optimizations for clock speed and arrangements for memory usage are made on microcontroller implementation. Details of this algorithm is presented in the following sections.

Our object representation method first starts with corner extraction. Types and their effects are discussed in Section 5.4. On MATLAB version of implementation, any corner extraction method can be chosen. Most commonly used algorithms are already implemented in MATLAB image processing toolbox. On the embedded side, a resource friendly option is needed to be selected. Harris corner detector suits this purpose. Other feature detection algorithms can be used as well. However, it should be run on microcontroller. Implementation details are explained next.

Algorithm 4.1. Object Representation pseudocode

```
Convert first image to grayscale
Extract feature points
Cluster feature points
Calculate mean and standard deviation for each cluster
Calculate vote vector for each cluster
```

4.3.2.1. Harris Corner Detection

The proposed object representation method can use various feature detectors working as corner extractors. When it comes to microcontroller implementation, the method should be fast and efficient. The easiest method for implementation is edge locations. However, this method produces excessive amount of points. Hence, it is more convenient to use more specific points such as corners. Harris and KLT operators are the simplest, efficient and reliable corner detectors in literature. Compared to KLT, Harris operator produces more reliable corner points as in repeatability and immunity to lighting conditions. Therefore, the Harris corner detector is selected as the feature extraction method on the microcontroller. In order to calculate the Harris response map, we need to obtain first derivatives of the image in x and y directions. This operation is done 2D convolution of Sobel filter kernels and structure tensor matrix as

$$\text{M(x,y)} = \sum_{i=x-1}^{x+1} \sum_{j=y-1}^{y+1} \begin{bmatrix} D_x(i,j)^2 & D_x(i,j)D_y(i,j) \\ D_x(i,j)D_y(i,j) & D_y(i,j)^2 \end{bmatrix} \tag{4.1}$$

where D_y and D_y are the derivative values in x and y coordinates, respectively. Then, the Harris response (M_R) is calculated as $M_R = det(M) - k(trace(M))^2$. Here, k is set to 0.04 empirically [3]. H_R is considered as the Harris response map. Local peaks of this map are considered as corner points. Extraction of these locations is handled by procedure named non-maxima suppression.

4.3.2.2. Non Maxima Suppression

Non-maxima suppression (NMS) is an essential post-processing operation in many computer vision applications [77, 78, 79]. It is used to obtain multiple location points or bounding boxes in a smooth response map, in our case corner locations, by extracting the local maxima. There are various methods while implementing NMS [80]. The common approach is greedy procedure which is threshold. Due to its simplicity, we used this method in our microcontroller implementation.

NMS is required for our corner detection method since we should locate the pixel which maximizes the Harris response locally. Locating the corner point falsely may result in shifted direction of the gradient. Since the method needs to run on a microcontroller, fast and memory efficient version of this method is implemented.

We assumed that a blobs will left from in Harris response map H_rR when we threshold. Highest response value of the blobs will be extracted as the corner locations. A recursive algorithm which extract these locations is implemented as in Algorithm 4.2 and 4.3. In Algorithm 4.2 we search the area for new corner point. When we meet a pixel value higher than threshold, we recursively call the function named NMS. This function is defined in Algorithm 4.3. It searches and returns maximum valued location in the blob.

Advantage of this recursive implementation is that it does not require any extra image sized memory. It also does not require dynamic memory allocation like one-pass or two-pass labeling algorithms. It boils down thresholding, labeling, and non maxima suppression.

Algorithm 4.2 demonstrates application of the non maxima suppression for a given Harris response map obtained from its normalized version. In this pseudocode, every pixel is scanned for possible blob for corner location. If a pixel value is over the threshold, it is accepted as a new blob for corner location. The variable that holds the maximum value of the blob is initialized and the recursive function name NMS is called. We provide pseudocode for this function in Algorithm 4.3. This function scans and returns the corner blob for maximum valued location. Also, it uses only memory for dedicated to response map and assigns pixel values higher than one. Thus it can yield a labeling procedure. In our application, we did not use the labeling value. However, it is a side product of

Algorithm 4.2. Non-maxima suppression pseudocode

H_R; Normalized Harris response map

ϵ; Corner detection threshold value

$n = 0$; Corner count

for $i = 1$ **to** $ImageWidth$ **do** $i = i + 1$; Subwindow column index

for $j = 1$ **to** $ImageHeight$ **do** $j = j + 1$; Subwindow row index

if $H_R(i, j) \geqslant \epsilon$ **AND** $H_R(i, j) < 1$ **then**; New Corner Location Condition

$MaxValue = 0$; Initialized for detected corner blob

if $H_R(i, j) \geqslant MaxValue$ **then**

$MaxValue = HR(i, j)$

$MaxLocation = (i, j)$

end if

$CornerLocation[n] =$ NMS($H_R, i, j, Threshold, MaxValue$); Add this location to corner locations array

$n = n + 1$

else

$H_R(i, j) = 0$; Set pixels to zero under threshold

end if

end for

end for

implementation.

Algorithm 4.3. NMS function pseudocode

function NMS(I,i,j,Threshold,MaxValue,MaxLocation)
  for $k = -1$ **to** 1 **do** $k = k + 1$; Subwindow column index
    for $l = -1$ **to** 1 **do** $l = l + 1$; Subwindow row index
      if $k = 0$ **AND** $l = 0$ **then**
        continue
      end if
      if $I(i + l, j + k) \geqslant Threshold$ **AND** $I(i + l, j + k) < 1$ **then**
        if $I(i + l, j + k) \geqslant MaxValue$ **then**
          $MaxValue = I(i + l, j + k)$
          $MaxLocation = (i + l, j + k)$
        end if
        $I(i + l, j + k) = I(i + l, j + k) + 1$
        $MaxLocation =$ NMS$(I, i + l, j + k, Threshold, MaxValue)$
      end if
    end for
  end for
  return $MaxLocation$
end function

4.3.2.3. Clustering

After extracting corner points, the clustering operation is applied to the points. Pseudo code for clustering is given in Algorithm 4.4. Corner points that are close to each other for given distance are grouped together and labeled. After this procedure is run, same labeled corner locations are considered as in the same group. Hence, the voting procedure is conducted accordingly.

On the PC implementation, the overall pdf can be calculated by Eqn. 3.32 However, this operation is very time consuming on the microcontroller. Therefore, predefined fixed

Algorithm 4.4. Grouping procedure pseudocode

$CornerLocations$; Array that extracted locations are stored

Set $ClusterLabels$ elements to 0; Same length array with locations, stores group labels

n; Number of found feature point

Set $LabelCounter = 0$

for $i = 0$ **to** $n - 1$ **do** $i = i + 1$; Label array index

  if $ClusterLabels[i] = 0$ **then**

    $LabelCounter+ = 1$

    $ClusterLabels[i] = LabelCounter$

  end if

  for $j = i$ **to** n **do** $j = j + 1$;

    $D =$Distance($CornerLocation[i]$,$CornerLocation[j]$)

    if $D \leqslant Threshold$ **then**

      if $ClusterLabels[j] = 0$ **then**

        $ClusterLabels[j] = ClusterLabels[i]$

      else

        Set all $ClusterLabels[j]$ to $ClusterLabels[i]$ in $ClusterLabels$

      end if

    end if

  end for

end for

windows are stored in header files for the voting operation.

4.3.3. Bayesian Tracking

In this section, we will discuss implementation details of our tracking algorithm with Bayesian filtering. As mentioned earlier, we implemented our trackers on both the PC and microcontroller. We realized our tracker in MATLAB on the PC side as explained in Section 3.4. On the other hand, we made some optimizations for memory and speed during the implementation on microcontroller. We will present these procedures next.

Pseudo codes for Bayesian tracker implementations are given in Algorithm 4.6, 3.1, and 4.5. Kalman filter calculation are done using matrix multiplication in CMSIS-DSP libraries. In grid based approximation, implementation a Gaussian window is used with pre-calculated header file. Particle filter implementation contains numerous exponential function calculation and random number generation. Implementation details for these operations are given in following sections.

4.3.3.1. Grid-Based Approximation

We provide the pseudocode of grid based pdf approximation method implementation of our tracker in Algorithm 4.5. In this version, we calculate two pdfs as $p(\mathbf{z}_k|\mathbf{x}_k)$ and $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$. $p(\mathbf{x}_k|\mathbf{z}_{1:k})$ is obtained as described in Section 3.4.1. At each time step, $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ is obtained by translation of pre-calculated 2D array which stores a Gaussian pdf. We can obtain prediction location from any motion model. We can calculate the update step by element wise multiplying these two pdfs.

4.3.3.2. Particle Filter

We provide pseudocode of particle filter implementation of our tracker in Algorithm 3.1 in the previous section. In implementation, we select our initial particles randomly around the target location with equal weight value in the first frame. Then, at each time step we translate particle locations according to the predicted location obtained from the motion model. Any motion model can be used in this step. Similar to grid based method, we calculate the pdfs $p(\mathbf{z}_k|\mathbf{x}_k)$ as in Eqn. 3.31. Different from the previous method, we only calculate these pdfs

Algorithm 4.5. Tracker Grid-Based Approximation pseudocode

Initialize Tracker
Initialize State Vector $\mathbf{x}_0$
Set $k = 1$
while Frame Available **do**
    $I =$ GetFrame()
    Calculate $p(\mathbf{z}_k|\mathbf{x}_k)$
    Predict $\mathbf{x}_{k|k-1}$ using motion model
    Translate $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$ according to $\mathbf{x}_{k|k-1}$
    $p(\mathbf{x}_k|\mathbf{z}_{1:k}) = p(\mathbf{x}_k|\mathbf{z}_{1:k-1}) \times p(\mathbf{z}_k|\mathbf{x}_k)$
    Update $\mathbf{x}_k$ with maximum Detection of $p(\mathbf{x}_k|\mathbf{z}_{1:k})$
    $k+ = 1$
end while

in particle locations. We assigned value of pdf as particle weights and accept the maximum weighted particle as target location. For the next time step, we resample particles with their weight. Therefore, we can discard particles with low probability value and move them to regions with higher probability value. Since we apply weighted sampling, it is highly likely to draw the same particle with multiple times. We add zero mean Gaussian noise to location of sampled particles at each time step.

We applied two extra procedures specific to the microcontroller in particle filter implementation. The first one is the random number generation. In order to resample particles and generate zero mean Gaussian noise, we need a random number generator. MATLAB has a function which produces random numbers with desired distribution. However, we rely on the random number generation module (RNG) on the microcontroller. We will explain the usage of this module in the next section. The second procedure is introduced while obtaining $p(\mathbf{z}_k|\mathbf{x}_k)$ and $p(\mathbf{x}_k|\mathbf{z}_{1:k-1})$. Here, we had to conduct too many exponential function calculations. An optimized version of this procedure will also be presented in the following sections.

4.3.3.3. Random Number Generation

There are two types of random number generators as true random and pseudo random number generator. Particle filter implementation requires a lot of normally distributed random numbers. The RNG module provides random numbers from uniform distribution. However, we need Gaussian distributed random numbers. There are several methods for this purpose. Box-Muller algorithm and ratio of uniforms are easy to implement methods [81, 82]. However, they are very slow. Inverse CDF is another method. Marsaglia and Tsang [83] introduced improved version of it named Zigguart method. It can generate numerous random numbers based on normally distributed random number. Original paper presents a C implementation of the method.

4.3.3.4. Exponential Function

Float number multiplication and exponential function are computationally expensive operations. If the microcontroller has a Floating Point Unit (FPU), float number multiplications can be done in a fast manner. Most microcontrollers in the market have the FPU because float number multiplication is a widely used operation. Standard C library contains an exponential function in `math.h` header file. However, it remains slow when many calculations in number is required. Therefore, it can become a bottleneck of the main process.

We overcome this bottleneck by using pre-calculated tables in Kalman filter and grid based approximation implementations. Since whole pdf needs to be calculated in both methods, using fixed tables speeds up the process significantly. These tables can be stored in the flash memory of the microcontroller since they are only needed to be read. Therefore, no additional memory is consumed from RAM. One drawback of this method is standard deviation of the pdf has to be fixed and can not be changed on the run.

In particle filter implementation, pdfs are calculated on average of 200 points. When it is compared to whole pdf, which is typically a 120×120 matrix, there is a significant gain. Therefore, we can use the exponential function on the run while computing weight of particles. Yet we do not need high precision when we calculate the exponential value. Schraudolph [84] proposed a method that approximates the exponential function by

manipulating components of a standard floating-point representation. This method yields six times faster implementation of exponential function. This method still remains slow compared to fixed tables for Kalman and grid based approximation methods. However, it gives ability to manipulate standard deviation of the calculated pdf.

4.3.3.5. Kalman Filter

We provide pseudocode of Kalman filter implementation of our tracker in Algorithm 4.6. In this version, we generate measurement pdf and take mode of the pdf as measurement. Then, we realize prediction and update steps as in Section 3.4.3. We can use the constant acceleration motion model for F_k. We assign constant diagonal matrices for noise covariance parameters Q_{k-1} and R_k.

In order to speed up measurement, we realized voting process with predefined arrays on the microcontroller as mentioned earlier in Section 3.4.3. Moreover, we use CMSIS-DSP library to realize matrix multiplications.

Algorithm 4.6. Tracker with Kalman filter pseudocode

function KALMAN FILTER

Initialize Tracker

Initialize State Vector $\mathbf{x}_0$

Set $k = 1$

while Frame Available **do**

$I =$ GetFrame()

Calculate $p(\mathbf{z}_k|\mathbf{x}_k)$

Predict:

$\hat{\mathbf{x}}_{k|k-1} = \mathbf{F}\hat{\mathbf{x}}_{k|k}$

$\mathbf{P}_{k|k-1} = \mathbf{F}\mathbf{P}_{k|k}\mathbf{F}^T$

Update:

$\mathrm{S}_k = \mathbf{H}\mathbf{P}_{k|k-1}\mathbf{H}^T + \mathbf{R}_k$

$K_k = \mathbf{P}_{k|k-1}\mathbf{H}^T\mathbf{S}_k^{-1}$

$\mathbf{x}_{k|k} = \hat{\mathbf{x}}_{k|k-1} + \mathbf{K}_k\tilde{\mathbf{y}}_k$

$\mathbf{P}_{k|k} = (\mathbf{I} - \mathbf{L}_k\mathbf{H})\mathbf{P}_{k|k-1}$

$k+ = 1$

end while

end function

5. RESULTS AND DISCUSSION

In this chapter, we present performance results and comparison of our tracker with the state of the art in literature. The proposed tracker and its variations are implemented on both PC and microcontroller. Trackers mentioned in Section 2.4 are also implemented on PC. Some of them are realized on the microcontroller as well. We tested implemented trackers on standard datasets.

5.1. DATASET USED IN EXPERIMENTS

The main aim in developing our tracker was using it on airborne images acquired by an UAV. Based on this aim, we formed a testbed to measure the performance of our tracker. To do so, we gathered 80 video sequences from seven different standard and publicly available tracking datasets. We provide the detailed list of video sequences as well as the challenges they have in Appendix. The standard tracking datasets that we used are as follows.

Nus-Pro [85] is a general tracking dataset published in 2016. It contains more than 300 image sequences. There are 12 different type of targeted moving rigid objects in the dataset. We used 16 sequences from this dataset.

The Stanford drone dataset has been published by Robicquet *et al.* [86] in 2016. This dataset is composed of bird eye view aerial images acquired by a drone. This is the best scenario for our tracking framework. The dataset consist of eight scenes and total of 60 videos. In video sequences, there are various moving objects such as cars, people, and bikers. We specifically targeted vehicles in this dataset. However, other objects can be tracked in this dataset as well. Therefore, nine sequences are cropped from videos which only contain movement of a specific target.

Vivid [87] tracker benchmark dataset is published in 2005. It contains video sequences taken from an aerial platform. We especially used the sequences in this dataset to test our tracker's performance with low resolution imagery. We used five sequences from this dataset.

OTB70 [22] is published in 2017. It aims to cover tracking problem in videos taken from UAVs. It contains 70 video sequence with various subject of interest. We used nine

sequences from this dataset.

Amsterdam Library of Ordinary Videos, (ALOV300) [21], aims to cover different type of visual tracking challenges. It contains more than 300 video sequences targeted 11 different type of circumstances. This is a general tracking dataset. Within it, we picked the sequences that are recorded from an aerial platform. We used six sequences from this dataset.

VisDrone [88] is an annual challenge for visual trackers. Dataset first published in 2018 and contains video sequences recorded with UAVs. It has different subsets that aims for different problems such as detection, single object or multi-object tracking. We used 4 sequences from this dataset.

UAV123 [89] is published in 2016. It mainly targets long term tracking problems. It contains more than 100 videos containing various subjects. We used 31 sequences from this dataset.

We composed a set of image sequences from these dataset which fits our aimed problem. Let's remind our problem limitations. We focus on aerial imagery. However, our tracker does not have any physical dependence on this setup. The main requirement for our problem is to have a fairly stable background image. In other words, if the background image is no changing much and there is no excessive clutter in the background, our tracking method works as expected. These constrains are most of the times satisfied for aerial images. Therefore, we focused on them in experiments.

Our main targeted object is vehicles. But we also considered other objects such as human, motorcycle, and boat. We grouped our test set into four main challenging circumstances as occlusion, low resolution, moving with similar objects, and scale/aspect ratio change.

Occlusion is partially or fully absence of the target for a short time interval. We also considered situation such that the object goes out of view briefly and comes back as in occlusion as well. We provide a sample sequence for this case in Figure 5.1. In this figure, target falls behind the bushes between first and third images.

If the target has very small size or low contrast, we consider this type of circumstance as low resolution. We provide a sample sequence for this case in Figure 5.2. In this figure, target is contained in a very small and low contrasted image patch.

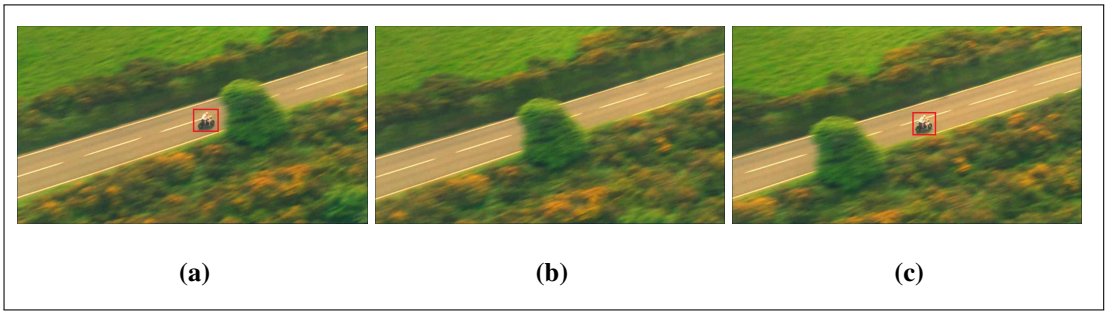

Figure 5.1. Sample sequence for occlusion (a) First image (b) Second image (c) Third image.

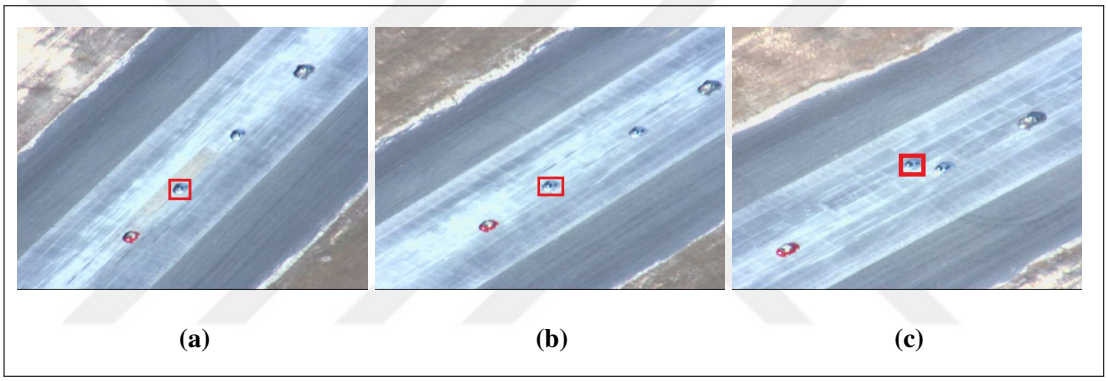

Figure 5.2. Sample sequence for low resolution (a) First image (b) Second image (c) Third image.

We next consider the scenario such that object being tracked is moving close to other object that are similar. We provide a sample sequence for this case in Figure 5.3. In this figure, target is a soldier who walks in a parade. Hence, there are similar dressed soldiers around the target.

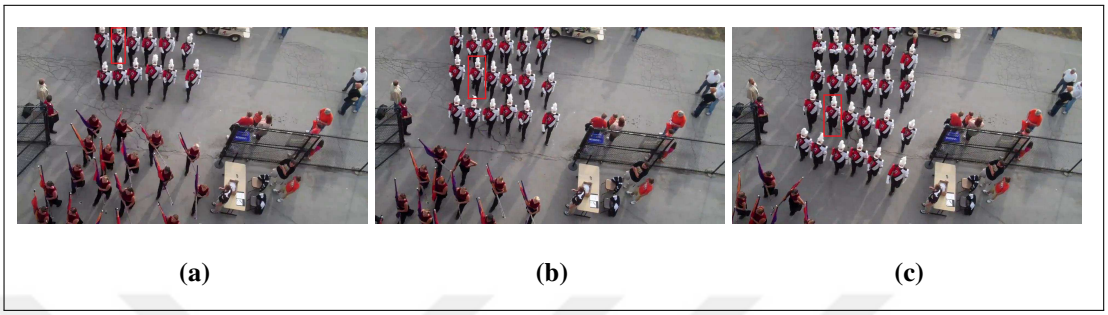

Figure 5.3. Sample sequence for being around similar objects (a) First image (b) Second image (c) Third image.

If the object's size or aspect ratio changes due to camera motion, we consider this situation as another challenge. We provide a sample sequence for this case in Figure 5.4. In this figure, camera starts recording from right side of the target vehicle ends in closer and right side of the target. Hence, both perspective and target size changes during the sequence.

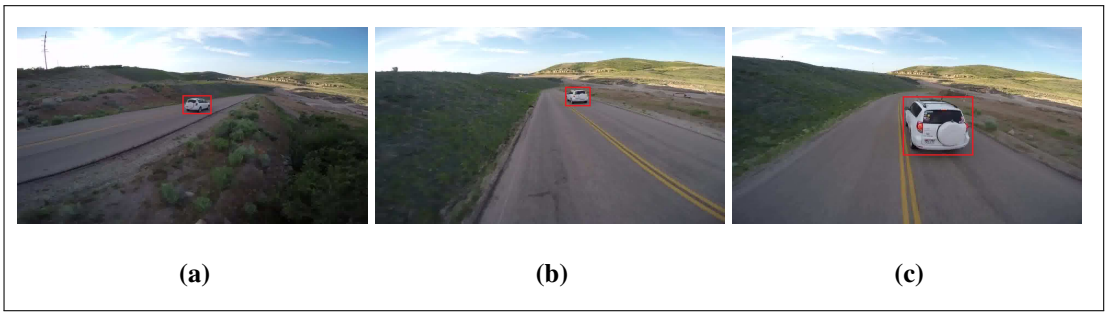

Figure 5.4. Sample sequence for changing scale and aspect of the object (a) First image (b) Second image (c) Third image.

Next, we will report the performance of our tracker compared to other trackers in literature. We will present overall and specific to each challenge performances.

5.2. EXPERIMENTS ON PC

We present results for five trackers in Figures 5.5 and 5.6. We listed and colorized individual results for better comparison for all sequences. Values in the figures are percentage of number of frames with difference between tracker result and ground truth is less than 50 pixels. Higher scores are colored as green and lower scores are colored as red. The tracker named Ours is the particle filter implementation of our method. We can see that our tracker out perform all other trackers in sequences *car4*, *bookstore6*, and *airplane_006*. These sequences include partial occlusion that our tracker can recover from and others cannot. On the other hand, our tracker performs worse than other trackers in sequences *soccer_3*, *soccer_10*, and *SpeedCar4*. That is because these sequences include very similar objects to the target object.

Success and precision plots are used to measure the trackers' performances. Both plots show the percentage of successfully tracked frames with respect to the threshold. The success plot thresholds the intersection over the union (IOU) metric and the precision plot thresholds the center location error. To rank the trackers, two types of ranking scores are used. The first one is the area under curve (AUC) metric for the success plot. The second one is the representative precision score at the threshold of 20 for the precision plot. Different from [90], only one-pass evaluation (OPE) is kept.

We provide the result of our tracker implementation and the ones in literature in Figure 5.7. Based on the figure, we can conclude that all trackers perform close to each other when we compare them on PC. Grid-based approach is slightly worse than other trackers. Our Kalman filter and tracking by detection methods lost the target in several sequences. Therefore, its precision and success plots have lower values.

As can be seen in Figures 5.7, 5.5, and 5.6, there is no dominance of a single tracker over all test datasets.

Timing performance of the considered trackers are given in Table 5.1. As can be seen in this table, KCF is the fastest tracking algorithm. Our method yields a moderate performance on PC. Hence, we pick KCF as a benchmark method for embedded implementation. Our trackers' timing performance may vary depending on number of extracted corner points.

	Ours	DSST	HOGLR	DAT	KCF	Mean Shift
01-Light_video00012	79.62	100.00	89.17	100.00	100.00	63.63
01-Light_video00014	70.63	77.72	59.74	13.99	71.83	20.38
01-Light_video00024	80.77	100.00	100.00	100.00	100.00	92.30
09-Confusion_video00002	50.45	100.00	100.00	78.83	100.00	99.09
11-Occlusion_video00006	100.00	100.00	98.33	96.67	99.44	93.88
11-Occlusion_video00007	98.68	97.85	97.52	68.48	98.02	70.95
airplane_002	92.00	99.50	100.00	99.50	100.00	93.00
airplane_003	52.50	62.00	82.50	66.00	37.50	28.00
airplane_004	90.50	84.00	73.00	51.00	96.50	97.00
airplane_005	100.00	47.00	47.00	48.50	37.00	43.00
airplane_006	100.00	34.50	100.00	100.00	21.00	100.00
airplane_007	85.50	61.00	61.00	74.00	61.00	90.50
airplane_008	78.00	26.00	70.00	34.50	73.50	15.00
airplane_009	100.00	79.00	98.00	10.50	8.50	66.50
airplane_010	100.00	68.50	100.00	100.00	95.50	100.00
airplane_011	76.00	78.33	51.33	97.67	93.33	62.66
airplane_012	91.67	100.00	9.33	37.00	98.33	7.00
airplane_013	78.33	18.33	65.67	97.00	52.00	91.33
airplane_014	96.33	19.67	99.00	100.00	17.00	12.33
airplane_015	94.33	83.00	91.33	55.00	86.00	46.00
airplane_017	55.00	55.33	40.00	60.33	36.00	94.67
airplane_018	77.67	56.00	80.67	99.33	56.67	97.33
airplane_020	65.33	51.67	38.67	49.00	58.33	64.33
boat2	100.00	100.00	100.00	100.00	100.00	100.00
boat4	34.18	39.42	75.77	40.14	40.14	66.18
boat5	8.12	51.49	51.49	32.67	53.86	36.83
boat6	33.04	35.90	34.91	35.53	35.28	37.14
boat8	32.99	14.89	25.99	27.74	14.89	43.36
bookstore	100.00	100.00	100.00	100.00	100.00	100.00
bookstore_2	100.00	100.00	53.53	100.00	100.00	100.00
bookstore_3	100.00	100.00	100.00	100.00	100.00	100.00
bookstore_4	100.00	100.00	100.00	100.00	100.00	100.00
bookstore_5	100.00	100.00	100.00	100.00	100.00	100.00
bookstore_6	99.45	6.45	2.88	2.47	11.39	4.39
car_5	100.00	100.00	84.52	48.81	100.00	100.00
car_6	75.85	65.62	91.86	62.47	69.82	61.68
car_8	80.25	100.00	97.49	98.12	94.36	97.49
car1	29.52	57.89	58.24	21.38	57.21	57.89
car10	100.00	100.00	100.00	100.00	100.00	100.00

Figure 5.5. Results for each sequence.

	Ours	DSST	HOGLR	DAT	KCF	Mean Shift
car11	32.64	21.66	16.02	22.85	25.22	21.07
car14	61.57	61.87	61.49	69.40	65.86	61.57
car2	42.54	44.28	99.02	99.17	99.17	99.17
car3	90.33	100.00	99.48	100.00	100.00	100.00
car4	100.00	30.86	44.09	29.96	29.74	87.06
deathcircle_1	100.00	100.00	52.02	100.00	100.00	100.00
deathcircle_2	100.00	100.00	99.29	100.00	100.00	100.00
deathcircle_3	99.50	100.00	100.00	99.34	100.00	100.00
egtest01	100.00	100.00	19.62	100.00	100.00	100.00
egtest02	42.15	96.00	43.23	44.46	95.38	42.31
egtest03	8.99	76.38	21.17	20.51	97.16	21.09
egtest04	28.33	41.43	10.59	42.69	43.12	81.44
egtest05	13.27	14.41	13.95	13.56	13.84	12.76
ManRunning1	92.73	15.35	98.06	100.00	57.03	93.86
person1	96.75	96.50	96.25	97.00	96.50	98.62
person11	42.86	24.97	33.43	22.61	24.83	99.17
person15	73.71	100.00	100.00	13.97	100.00	100.00
person2	99.05	74.00	99.77	100.00	100.00	85.70
person22	100.00	100.00	100.00	96.98	100.00	100.00
person23	93.95	100.00	100.00	93.70	100.00	69.27
person3	100.00	100.00	100.00	100.00	100.00	100.00
person6	100.00	100.00	100.00	100.00	68.81	88.12
RcCar5	5.45	23.03	87.58	43.64	4.85	98.79
RcCar8	4.27	5.69	8.53	8.06	100.00	94.79
RcCar9	13.46	10.10	99.52	21.63	92.31	97.60
soccer_001	50.00	74.35	58.70	100.00	50.87	69.57
soccer_002	92.80	54.24	25.00	48.31	100.00	74.15
soccer_003	65.71	100.00	100.00	100.00	100.00	100.00
soccer_004	80.29	1.43	23.43	23.71	76.00	23.43
soccer_005	6.64	62.89	81.64	100.00	33.59	64.06
soccer_006	60.98	6.20	6.20	6.20	6.20	5.94
soccer_007	59.25	87.17	90.19	89.06	86.79	100.00
soccer_008	30.00	87.00	99.50	100.00	91.50	80.50
soccer_009	50.00	18.75	42.97	100.00	100.00	100.00
soccer_010	53.61	100.00	98.06	100.00	90.56	53.06
SpeedCar2	100.00	78.25	97.73	80.52	100.00	41.23
SpeedCar4	35.37	100.00	100.00	89.02	100.00	100.00
uav0000126_07915_s	0.00	100.00	97.46	95.24	84.29	71.43
uav0000238_01280_s	38.88	100.00	100.00	100.00	100.00	88.32
uav0000252_00001_s	90.76	92.45	71.74	89.45	88.67	85.42
uav0000303_01250_s	41.462	99.140	98.956	52.58	98.83	18.18

Figure 5.6. Results for each sequence. (Continued)

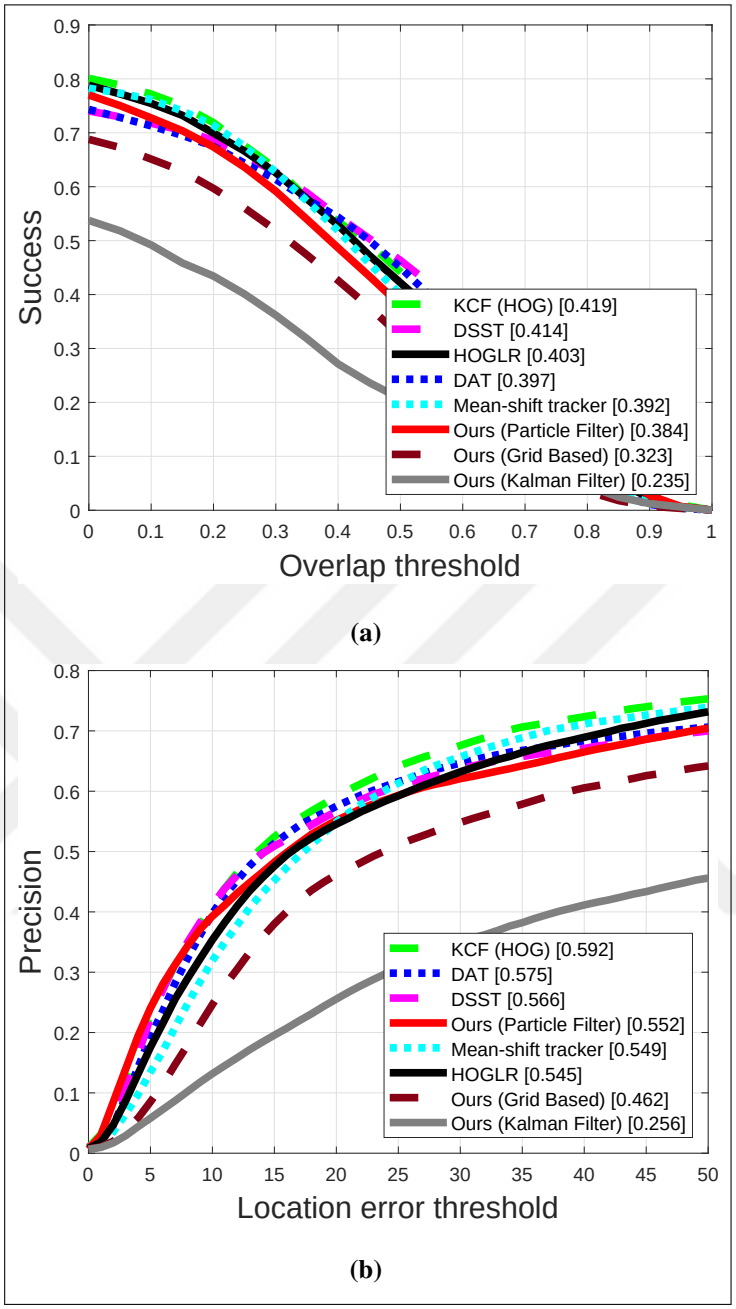

Figure 5.7. Benchmark trackers performance (a) Success plot (b) Precision plot.

We provide an average number for PC implementation. We examine timing performance on embedded performance in detail.

Table 5.1. Timing performance of trackers on PC.

Method	**FPS**
Ours (Particle Filter)	**32.72**
HOGLR	4.88
KCF	320.82
DAT	13.85
DSST	30.49
Mean Shift	30.17

5.3. PERFORMANCE ON CHALLENGES

We labeled every test sequence with the challenge it contains. We give detailed list of the sequences and challenges in Table A.1 and A.4. In this section, we compare the trackers according to specific challenges.

5.3.1. The Effect of Occlusion

Occlusion is the temporal loss of the target in the visual sight. This can be caused because another object can partially or fully block the target or target can go outside the camera frame. We present the results only for the sequences that have occlusion in Figure 5.8. As can be seen in this figure, our tracker can recover the temporal absence of the target and produce competitive results compared to other trackers.

5.3.2. The Effect of Changing Scale

Target appearance or size can change during the sequence due to camera or target movement. We present the results only for the sequences that includes changing scale of the target in Figure 5.9. As can be seen in this figure, our tracker can handle the scale change and provides competitive results compered to other trackers.

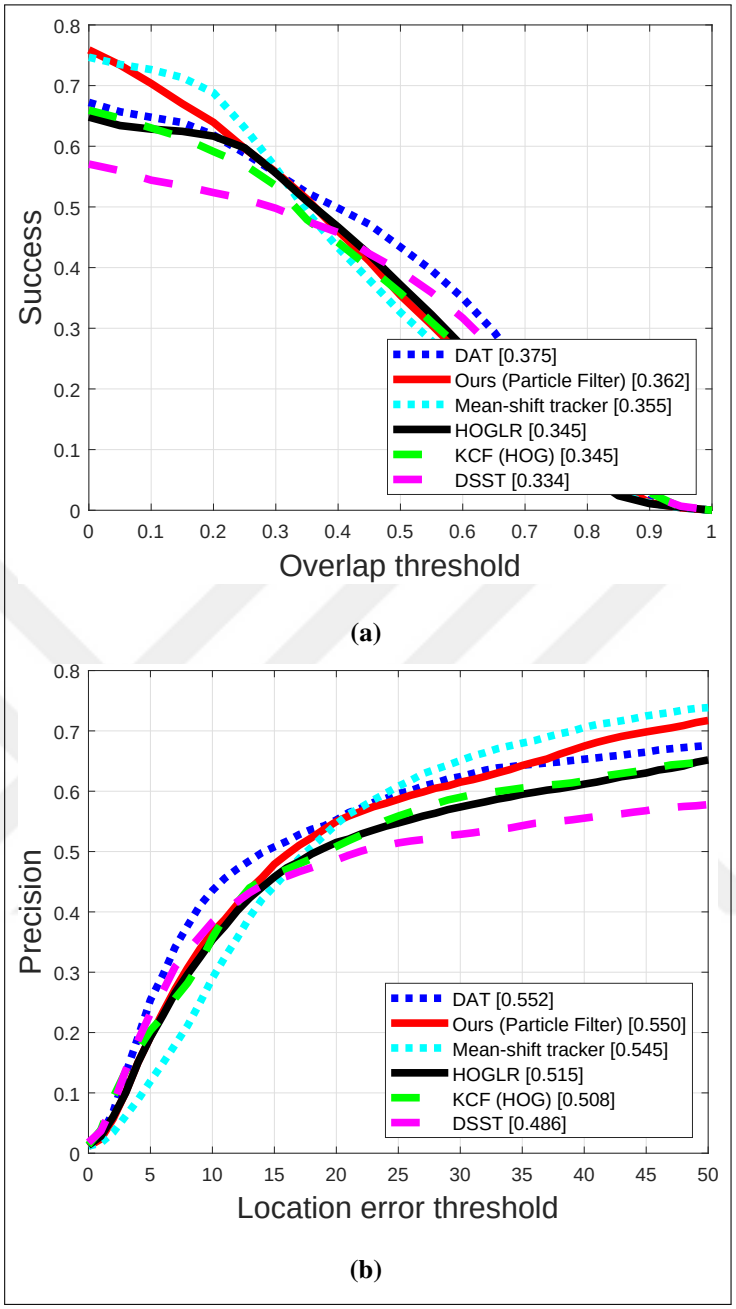

Figure 5.8. Results for the sequences which have occlusion (a) Success plot (b) Precision plot.

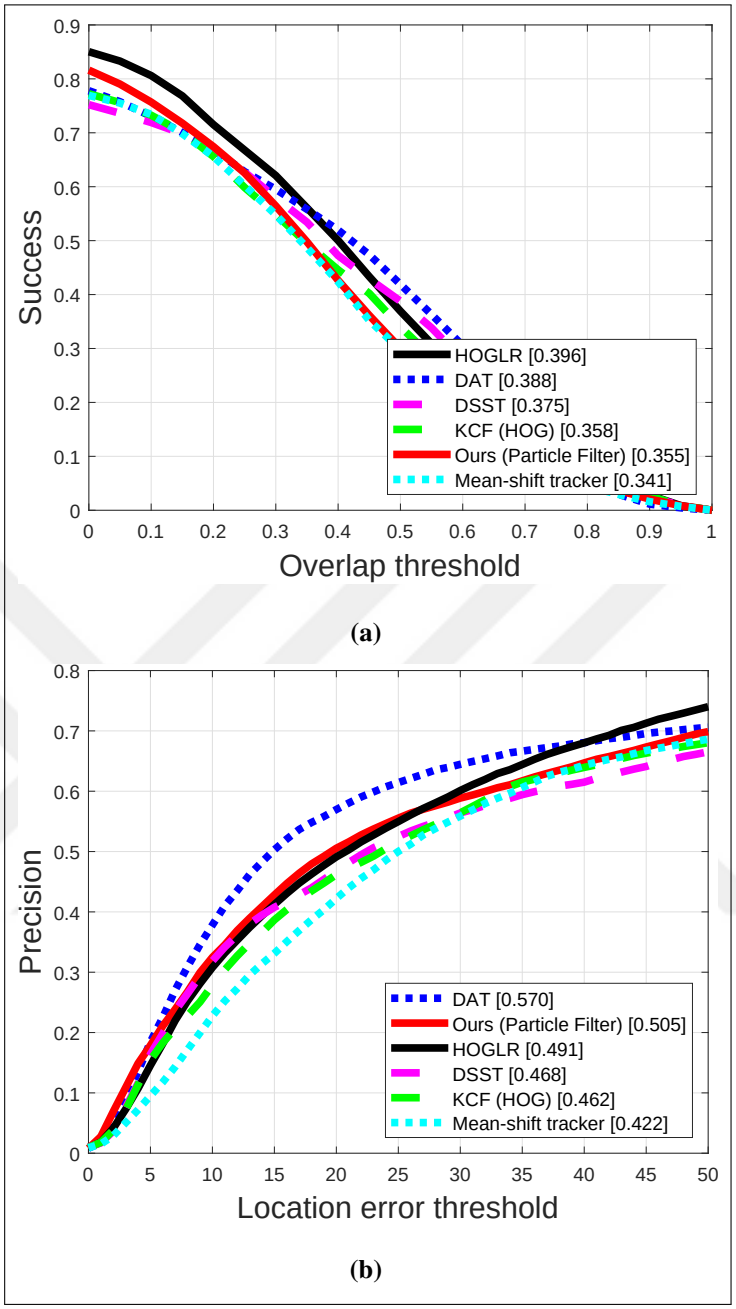

Figure 5.9. Results for the sequences with scale change of the target (a) Success plot (b) Precision plot.

5.3.3. The Effect of Image Resolution

In some images, target can be defined poorly in the sequence which means target can be very small portion of the image because of the distance to camera. Or image can be flue due to camera focus. Hence, low contrasted images can be occurred. We present the results only for the sequences with low resolution in Figure 5.10. As can be seen in figure, our tracker performs poorly in such conditions. This is because our target representation relies on the corner points which requires definitive contrast. In these kind of images, corner detection performs poorly. Therefore, our tracker's performance reduces.

5.3.4. The Effect of Similar Objects

We present the results only for the sequences that target moves close to look alike objects in Figure 5.11. Our tracker performs very poorly in these kind of sequences because we do not consider if a corner point belongs to target or not. We process all emerged corner points in the search region. We can overcome this problem by using by more definitive features such as SIFT, and conduct a matching process between current and initial image.

5.4. THE EFFECT OF CORNER EXTRACTOR ON PERFORMANCE

Our object representation method can run with various types of corner (feature) extraction methods. We analyzed the effect of these corner (feature) extractors in this section.

Corner detection methods such as Harris or FAST, feature generation methods such as SIFT, BRISK, MSER or simply edge points can be used within the proposed tracker. Performance of the object representation method is directly related to the number of voting points. Hence, edge points yield better results compared to other corner detection methods. However, computational cost increases with the number of the voters.

In Figure 5.12 performance comparison of feature detection methods are presented. We tested feature points with tracking by detection method in order to emphasize the effect of feature extraction methodology. We also did not use any other filter or procedure such as grouping. Harris, minimum eigenvalue (KLT), BRISK and FAST are corner detection algorithms available in MATLAB. Other feature detection algorithms available in MATLAB (such as KAZE and MSER) are also included. Another corner detection method, L-shaped

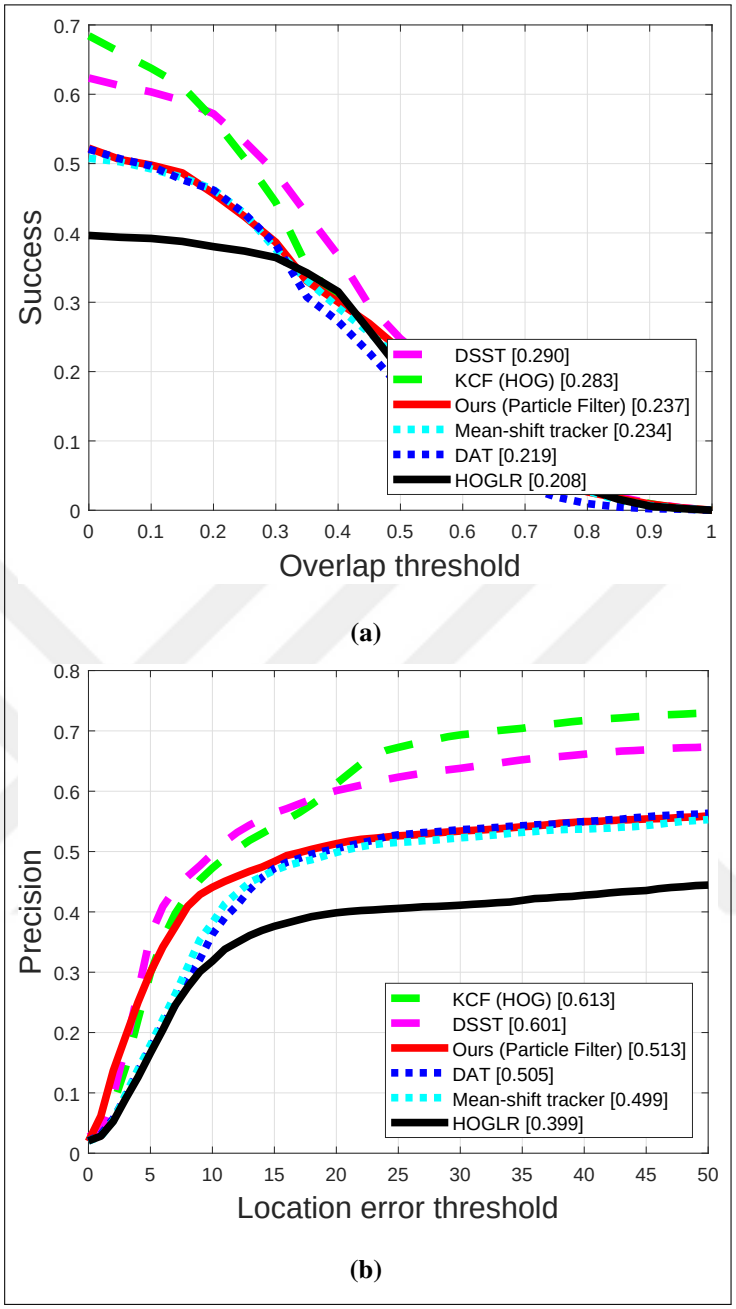

Figure 5.10. Results for the sequences with low resolution (a) Success plot (b) Precision plot.

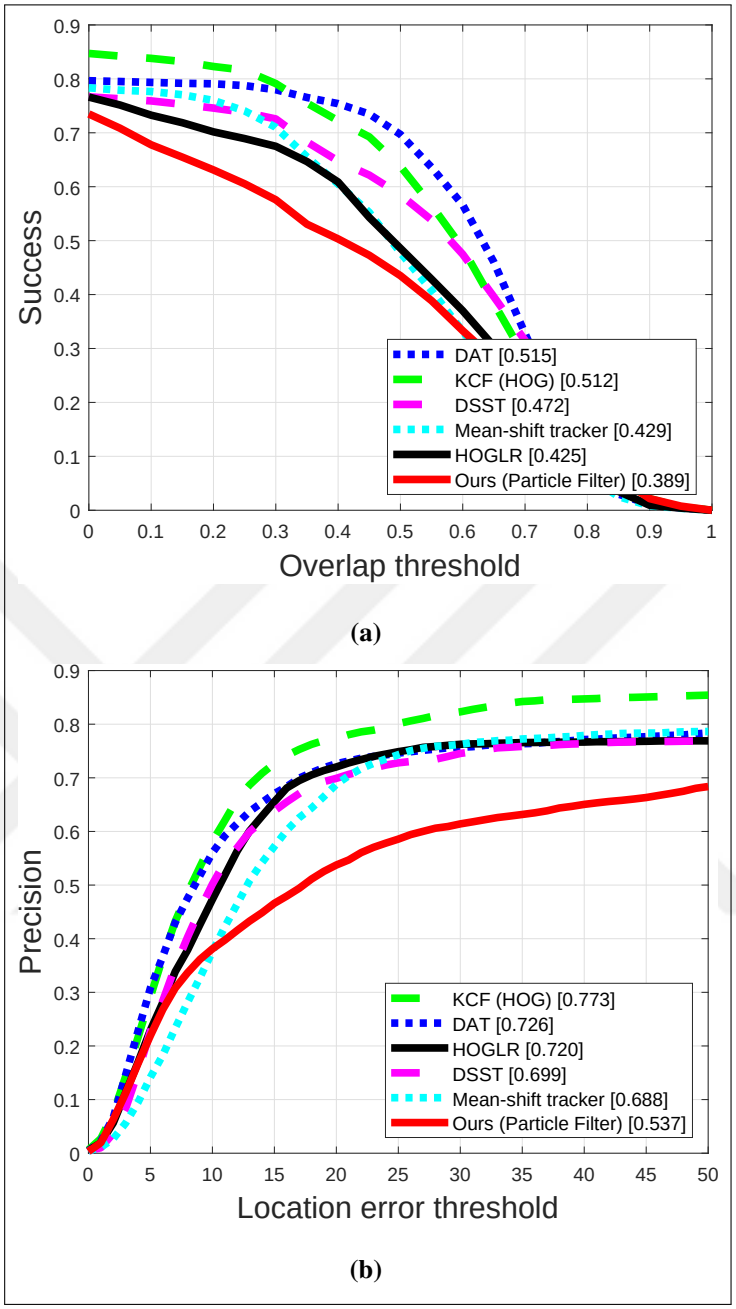

Figure 5.11. Results for the sequences that the object is close to similar objects (a) Success plot (b) Precision plot.

steerable filter and Sobel edge detection are also implemented.

As can be seen in Figure 5.12, edge detection method has the best performance as predicted. Because more voting points belonging to the target will yield more accumulation on the target. Hence, target object will be the dominant location in the measurement pdf. Steerable filter gives the second best result because more definitive corner points can be obtained with this method. However, this method requires an angular search of corners. Both edge and steerable filter features are computationally costly. If we use edge locations, voting procedure will take too long. On the other hand, if we use steerable filter feature extraction, it will take too long because of angle sweeping. Effect of other features are also given in Figure 5.12. We selected Harris corner method for further experiments.

5.5. APPROXIMATE BAYESIAN TRACKING METHODS

Within the Bayesian tracking framework, there three options as Kalman filter, particle filter, and grid based approximation. In this section, performance of these options are discussed.

Kalman filter accepts prediction and measurement probabilities only as Gaussian pdf. Therefore, it only takes detected target location and assumes normally distributed pdf. We can say that with Kalman filter, we can not use full benefit of our object representation method.

We present the precision and success plots in Figure 5.13. Kalman filter it can not be very successful since it ignores the non-gaussian structure of the measurement pdf. Grid based approximation gives better result since it cover the whole pdf in object representation and update steps. In theory we expect to grid based and particle filter produce similar results. However there gap between grid based and particle filter approaches in Figure 5.13. This is because, during the experiments we down sample the input image to speed up the computation since calculating whole pdfs takes too long. Another reason, during the development of the algorithm we tested particle filter more and it is more optimized. Kalman filter has the advantage of computational low cost compared to grid based approach. However, particle filter overcomes Kalman filter and grid based in speed and performance. Due to our object representation method's structure, particle filter can speed up the measurement and Bayesian filter methods at the same time.

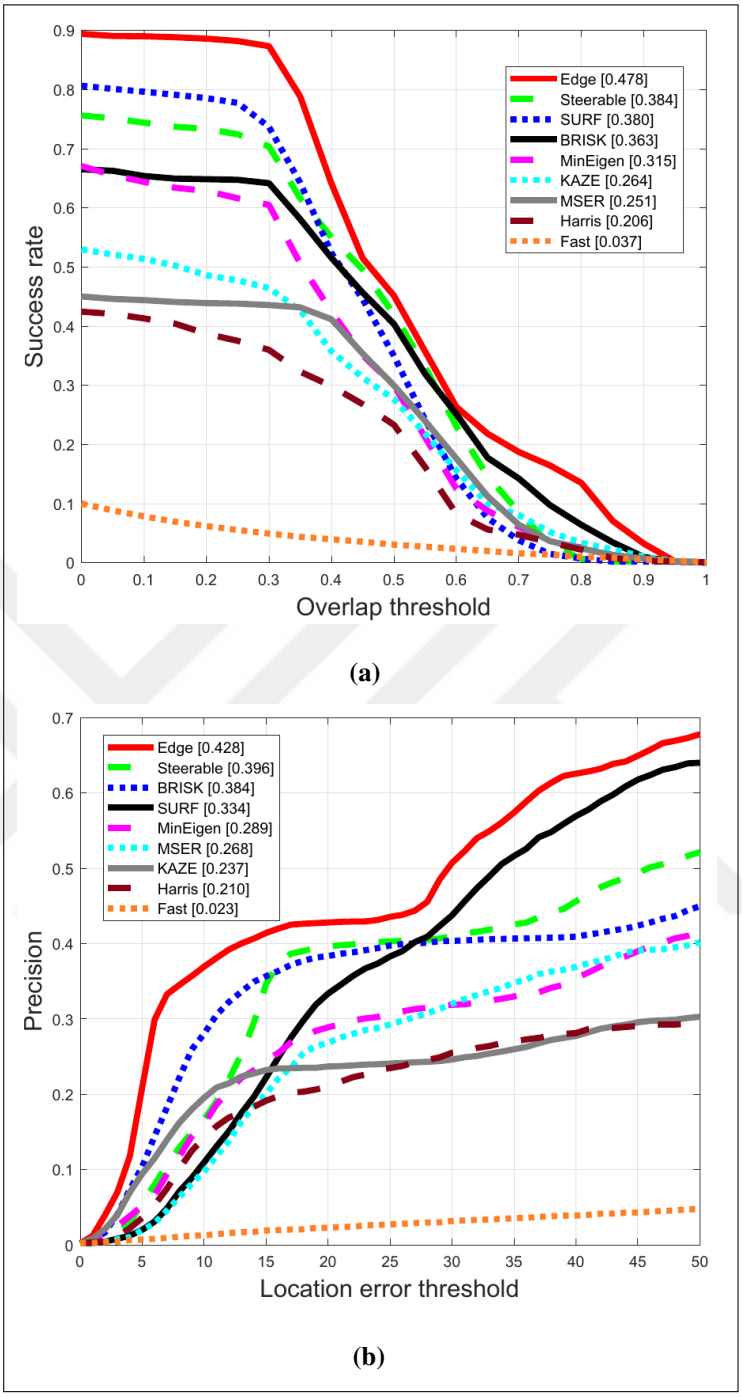

Figure 5.12. Success and precision plots for different feature detection methods (a) Success plot (b) Precision plot.

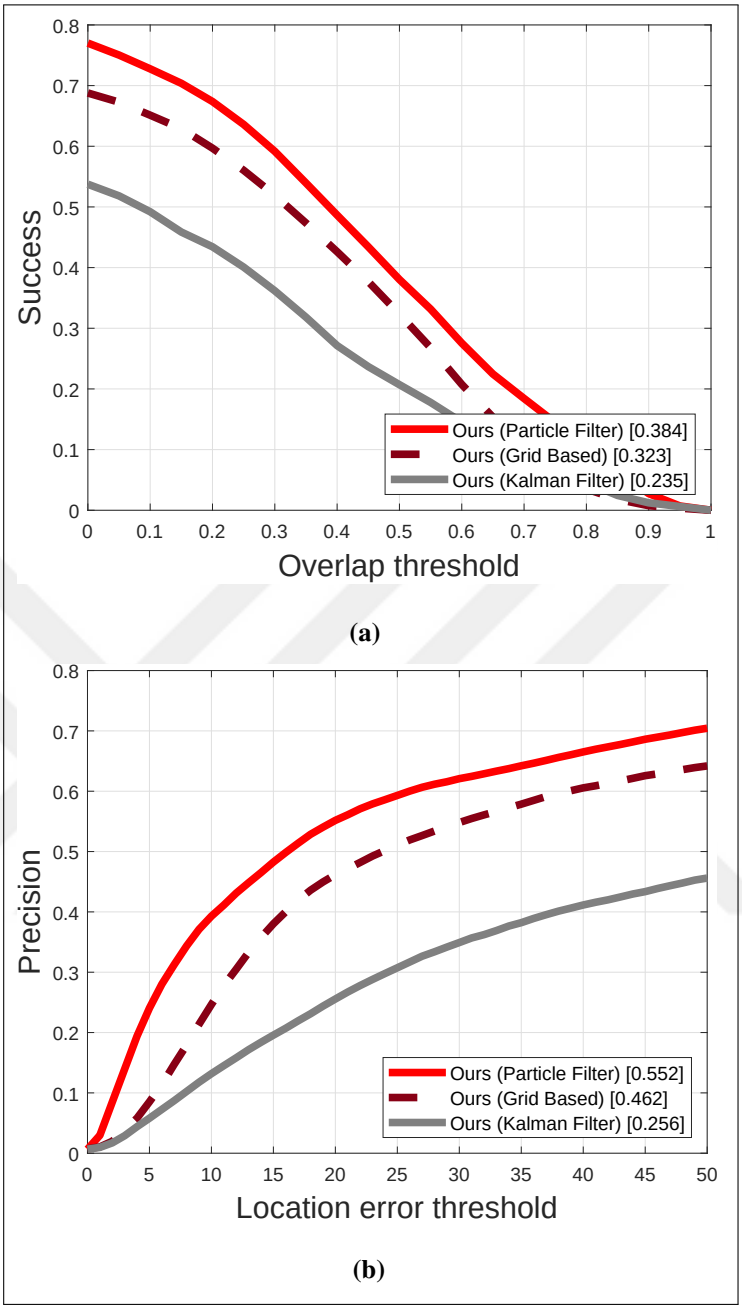

Figure 5.13. Success and precision plot comparison of our trackers with Bayesian filter (a) Success plot (b) Precision plot.

We present effect of number of the particles in particle filter in Figure 5.14. Number of the particles are directly related to time of computation for the particle filter. Increasing or decreasing the number of particles depends on how densely the pdf is sampled. Therefore it is important to select the particle number enough to cover effectively the sampled space.

As can be seen in Figure 5.14, there is a saturation after 20 particles in our case. Which means our prediction and update pdfs can be successfully covered with 20 particles. If we consider the grid based approximation method, generated pdf's size is 120×120 which is 720 times higher than the particle filter. However, we selected the particle number as 200 to be safe during the experiments.

5.6. EMBEDDED IMPLEMENTATION

We provide the result of trackers that are implemented on STM32F746 Discovery board in Figure 5.15. The worst performance belongs to tracking by detection tracker as expected. The best performance in precision and success plots is particle filter implementation of our tracker.

In Table 5.2 timing performances of the trackers on microcontroller are given. We observe that particle filter with our detection method yields the fastest performance since we calculate measurement pdf in few points. Therefore, it can speed up the detection method compared to other trackers. For the grid based and Kalman filter method, calculating the whole pdf is needed for locating the target. KCF tracker can not operate as much fast as in PC due to requirement of additional memory on the microcontroller. Classic Kalman filter method is the slowest one because we used template matching method which is an extremely slow process. Color histogram matching as in classic particle filter is faster than template matching. However, it can not catch up our particle filter implementation.

Performance comparison of the trackers implemented on MATLAB and microcontroller is given in Figure 5.16. As can be seen in this figure, performance of the KCF changed drastically. The main reason for this change is the usage of raw pixel values on the microcontroller. For MATLAB implementation, HOG features are used hence it can perform much better. It is important to say that our particle filter implementation can be exported to microcontroller without loss of performance.

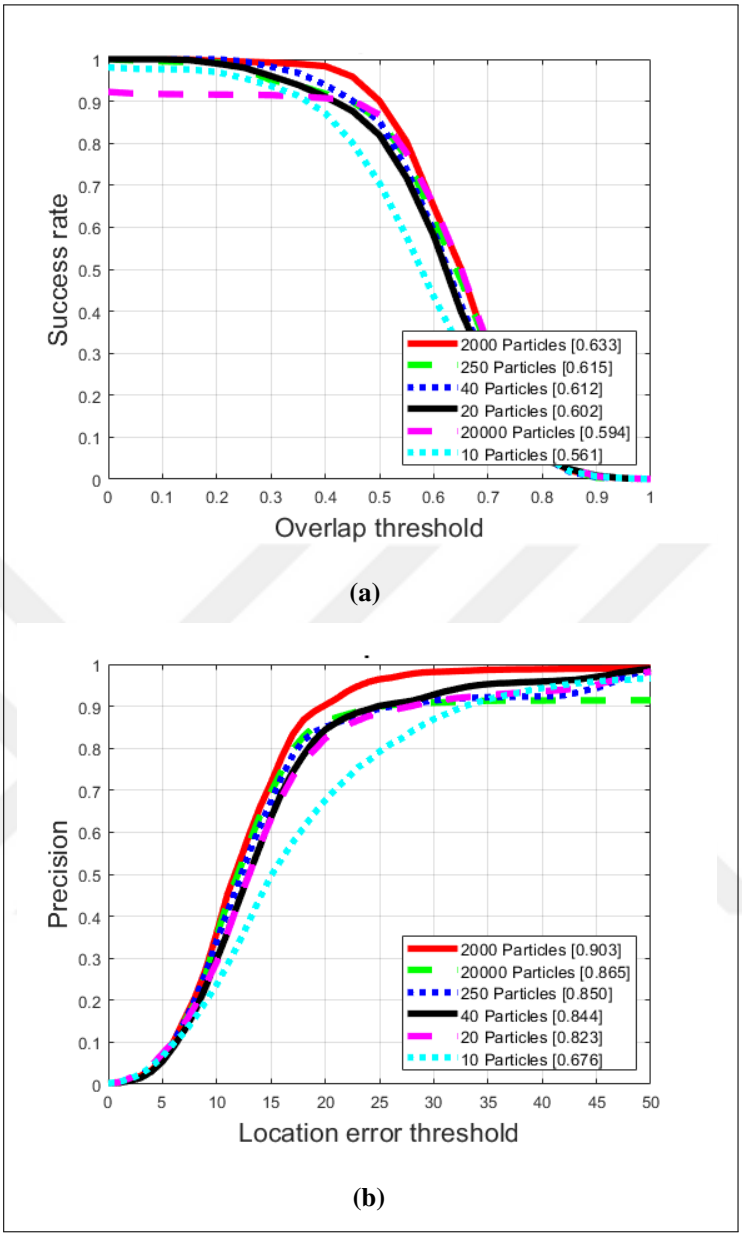

Figure 5.14. Comparison of our particle filter wrt particle number (a) Success plot (b) Precision plot.

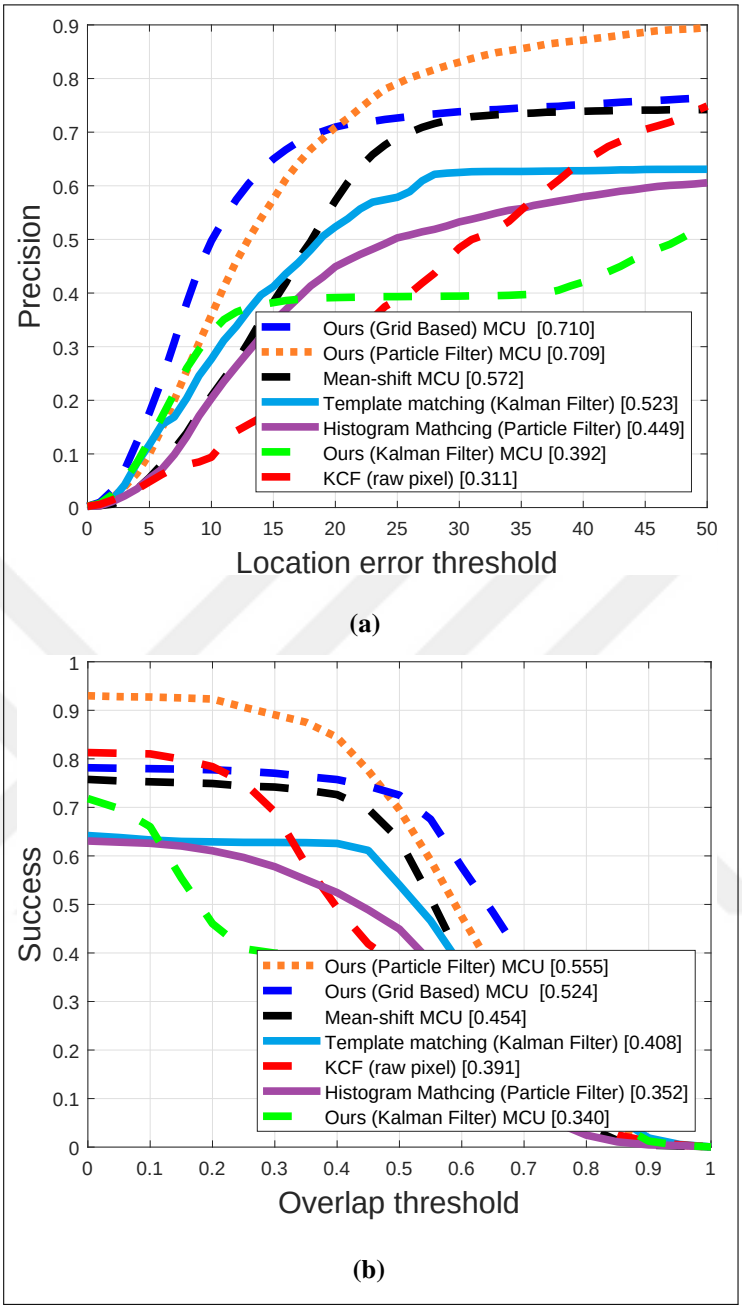

Figure 5.15. Success and precision plot comparison of trackers on microcontroller (a) Success plot (b) Precision plot.

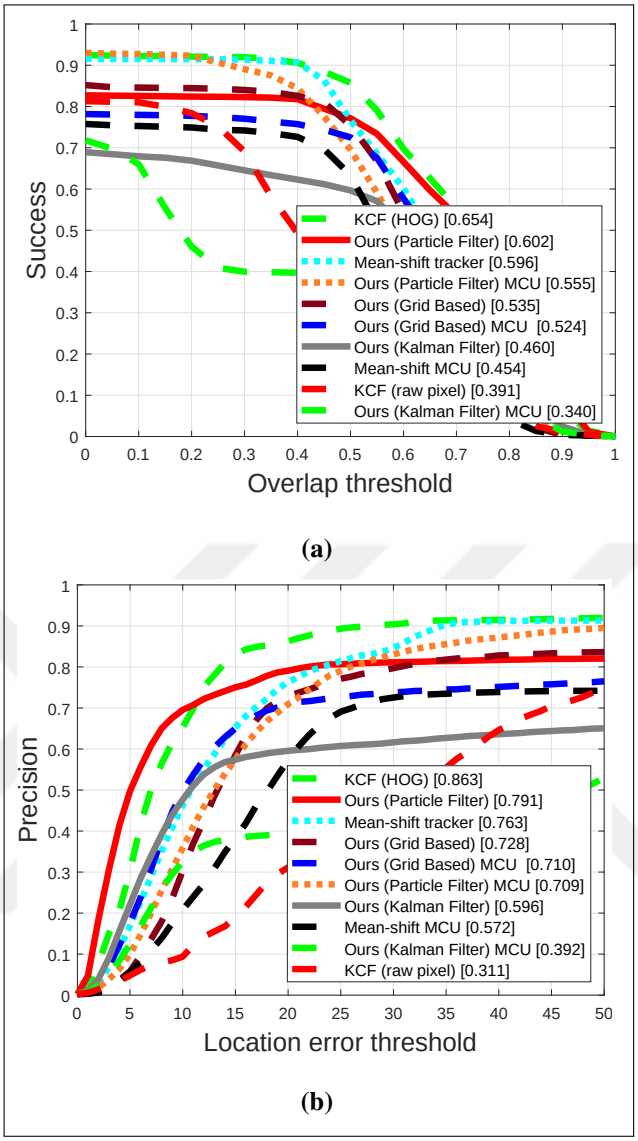

Figure 5.16. Trackers comparison with microcontroller and PC implementations (a) Success plot (b) Precision plot.

Table 5.2. Timing performance of trackers on microcontroller.

Method	**FPS**
Our tracker (particle filter)	**8.6**
Our tracker (Kalman filter)	8.08
Our tracker (grid-based)	7.92
KCF	2.17
Classic Kalman filter	0.54
Classic particle filter	3.38
Mean Shift	1.65

5.7. MEMORY USAGE

Memory usage is crucial for microcontroller implementation due to limited resource of memory. In this section, memory requirements of the tracker implementation are compared. Allocation strategy of RAM and flash memory requirements are presented.

5.7.1. RAM Usage

As mentioned earlier, microcontroller has limited memory source. In our setup, the selected microcontrolller has 320 kilobytes internal RAM. It can access to eight megabytes of external RAM. In our test sequences, target size is around 70×50 pixels. Therefore, region of interest patch size is selected as 120×120 pixels.

For all trackers that are implemented with our object representation method, only internal RAM is used. One patch sized unsigned char type memory space is reserved for input . Three patch sized float type image is allocated for measurement and pdf calculations. Two of them are holding the gradient information and one is used for Harris response and pdf calculations. For Kalman filter and grid based estimation implementations, prediction and voting pdfs are stored in flash memory. Hence, they do not occupy memory on RAM. For the particle filter implementation, these windows are discarded. Instead, a `double` type array with number of particle element is allocated. As a result $13 \times PatchWidth \times PatchHeight$

bytes memory is allocated to image processing operations. Additionally, four kilobytes memory is reserved within stack which is used by function variables, arrays and additional libraries such as FatFS.

KCF implementation contains a lot of FFT operations. Therefore, the patch size is selected as 128×128 pixels for ease of calculation. As aside note, FFT functions in CMSIS-DSP library accept arrays only with a length of power of two. FFT of an image resulted with complex numbers and input of inverse FFT function is also complex number. Therefore, at least two float type patch sized memory should be allocated in order to store and operate these functions. KCF tracker requires FFT and IFFT operations multiple times and it also stores model patch as complex numbers. As a result, it needs $49 \times PatchWidth \times PatchHeight$ bytes memory for transformation functions in total. This amount of memory is not available on the STM32F746. Consecutively external RAM chip is used in KCF implementation. One drawback of this usage is that external RAM is accessed through a module named Flexible Memory Controller (FMC). It can not operate as fast as the internal bus in the microcontroller. FMC is fed by peripheral clock in microcontroller which can be half of the core clock maximum. This causes drop in tracker speed by half. Another issue with the KCF is that it uses histograms of oriented gradients as the main feature [79]. As stated in original paper, it can also work with grayscale pixels. Since implementation of HOG algorithm on the microcontroller is time and memory consuming, grayscale pixel values are used. However, it also caused performance drop for the tracker.

Particle filter with Bhattacharya distance and Kalman filter with template matching implementations are the lightest trackers when memory is concerned. They can both run on smaller memory space compared to other trackers. Therefore, only internal RAM is used in their implementation. The main drawback of these algorithms that is they are both computationally expensive.

We present approximate RAM requirement of implemented trackers in Table 5.3. The values here are approximate since heap and stack memory requirements are not included in this table. Only image processing related memory size is presented because it consists of biggest percentage of the memory usage. As can seen in this table, KCF requires largest RAM size. This memory provided by external RAM chip on STM32F746 Discovery board.

Table 5.3. RAM requirements of implemented trackers.

Tracker	**RAM Size (Kilobytes)**
Our tracker (particle filter)	171
Our tracker (Kalman filter)	169
Our tracker (grid-based)	169
KCF	802
Classic Kalman filter tracker	57
Classic particle filter tracker	44
Mean Shift	259

5.7.2. Flash Memory Usage

We implemented six trackers on the microcontroller and ranked them by their space requirement on ROM. Code size is a measure that gives an idea about how large the program is. Code that generated by the compiler is stored in microcontroller's flash memory. In our case, STM32F746NG has 1 MB flash memory on chip. All implemented trackers are within this memory size.

Grid based approximation method requires the largest flash memory among our trackers because it has to include two precalculated Gaussian pdfs. These pdfs are stored in 2D arrays for voting and prediction. The KCF tracker needs large memory size since there is also a precalculated mask for filtering purpose. FFT of candidate image is filtered by this mask in order to reduce the effect of edges of the search area. Other trackers' flash memory requirements are close to each other because they mostly consist of function implementations, no predefined memory block is included. We present code size which indicates the flash memory size of our implementation in Table 5.4.

Table 5.4. Code sizes of implemented trackers.

Tracker	**Code Size (Kilobytes)**
Our tracker (particle filter)	84
Our tracker (Kalman filter)	105
Our tracker (grid-based)	181
KCF	199
Classic Kalman filter	65
Classic particle filter	70
Mean Shift	73

6. CONCLUSIONS

In this study, we proposed a novel visual tracking method. With our method, we achieved a visual tracker such that it can operate even on very resource limited hardware such as a microcontroller. As a result, we can discard the companion computer for space and power limited platforms such as UAVs.

We build our tracker framework on a probabilistic object representation method. We tested the performance of our tracker with different feature extraction methods. We obtained optimal results with the Harris corner detector. Since this method is able to operate with very limited resources, this would be an advantage when we implement our tracker on the microcontroller.

After the object representation step, we can adopt probabilistic the outcome of the detector in Bayesian filtering. To do so, we proposed three Bayesian filter implementations. First, we used Kalman filter. We observed that we can not use full information that is provided by our detector. Then, we implemented grid based pdf approximation method. We used pdf generated by detector in prediction and update steps of the Bayesian filter. We observed that we can speed up these steps using particle filter since we do not need to calculate the whole pdf. Particle filter speeds up the not only prediction and update steps, but also our object detector as well. Thanks to particle filter, we do not need to calculate whole pdf in measurement method either. This was the second advantage of our tracker for microcontroller implementation.

We compared our tracker with both classical and state of the art methods. On the PC, we obtained a moderate performance with our trackers among others. However, we can export our tracker on a microcontroller platform without performance loss. We obtained superior performance in terms of success and speed for this case. Hence, we achieved a tracker performance with low computational cost. In this study, we selected STM32F746 as our microcontroller implementation platform. It is a ARM Cortex-M7 based microcontroller. However there is no limitation to implement our tracker on microcontrollers with less computational resource such as ARM Cortex-M4 based microcontrollers. For example

STM32F429 has 192 kilobyte ram with similar clock speed. Our tracker would work but smaller image size. On the other hand we can achieve a faster performance with STM32H7 devices which offers two times higher clock than the STM32F746.

In this study we targeted only microcontroller as embedded platform. Our work can be extended to lower level of hardware such as FPGA with multiple object tracking.

REFERENCES

1. Drones A. AERIUS - The NEW World's Smallest Quadcopter; [cited 2020 17 January]. Available from: https://www.indiegogo.com/projects/aerius-the-new-world-s-smallest-quadcopter.

2. Yilmaz A, Javed O, Shah M. Object tracking: A survey. *ACM Computing Surveys.* 2006;38(4):13–es.

3. Harris C, Stephens M. A combined corner and edge detector. *In Proceedings of Fourth Alvey Vision Conference on*; 1988.

4. Chen Z. Bayesian filtering: from Kalman filters to particle filters, and beyond. *Statistics: A Journal of Theoretical and Applied Statistics.* 2003;182(1):1–69.

5. Grewal MS, Andrews AP. *Kalman filtering: Theory and practice with MATLAB. 4th ed.* New Jersey: Wiley-IEEE Press; 2014.

6. Arulampalam MS, Maskell S, Gordon N, Clapp T. A tutorial on particle filters for online nonlinear/non-Gaussian Bayesian tracking. *IEEE Transactions on Signal Processing.* 2002;50(2):174–188.

7. Khanna R, Möller M, Pfeifer J, Liebisch F, Walter A, Siegwart R. Beyond point clouds - 3D mapping and field parameter measurements using UAVs. *Emerging Technologies Factory Automation on*; 2015:IEEE.

8. Faessler M, Fontana F, Forster C, Mueggler E, Pizzoli M, Scaramuzza D. Autonomous, vision-based flight and live dense 3D mapping with a quadrotor micro aerial vehicle. *Journal of Field Robotics.* 2016;33(4):431–450.

9. Delmerico J, Giusti A, Mueggler E, Gambardella LM, Scaramuzza D. "On-the-spot training" for terrain classification in autonomous air-ground collaborative teams. *International Symposium on Experimental Robotics on*; 2016:Springer.

10. Giusti A, Guzzi J, Cireşan DC, He FL, Rodríguez JP, Fontana F, et al. A machine learning approach to visual perception of forest trails for mobile robots. *IEEE Robotics and Automation Letters.* 2015;1(2):661–667.

11. Mueggler E, Faessler M, Fontana F, Scaramuzza D. Aerial-guided navigation of a ground robot among movable obstacles. *International Symposium on Safety, Security, and Rescue Robotics on*; 2014:IEEE.

12. Olson E. AprilTag: A robust and flexible visual fiducial system. *International Conference on Robotics and Automation on*; 2011:IEEE.

13. Minaeian S, Liu J, Son YJ. Vision-based target detection and localization via a team of cooperative UAV and UGVs. *IEEE Transactions on Systems, Man, and Cybernetics Systems.* 2015;46(7):1005–1016.

14. Gohl P, Honegger D, Omari S, Achtelik M, Pollefeys M, Siegwart R. Omnidirectional visual obstacle detection using embedded FPGA. *International Conference on Intelligent Robots and Systems on*; 2015:IEEE.

15. Fraundorfer F, Heng L, Honegger D, Lee GH, Meier L, Tanskanen P, et al. Vision-based autonomous mapping and exploration using a quadrotor MAV. *International Conference on Intelligent Robots and Systems on*; 2012:IEEE.

16. McGuire K, de Croon G, Tuyls K. A comparative study of bug algorithms for robot navigation. *Robotics and Autonomous Systems.* 2019;121:103261.

17. Schauwecker K, Zell A. On-board dual-stereo-vision for the navigation of an autonomous MAV. *Journal of Intelligent & Robotic Systems.* 2014;74(1-2):1–16.

18. Engel J, Sturm J, Cremers D. Camera-based navigation of a low-cost quadrocopter. *International Conference on Intelligent Robots and Systems on*; 2012:IEEE.

19. Lynen S, Achtelik MW, Weiss S, Chli M, Siegwart R. A robust and modular multi-

sensor fusion approach applied to mav navigation. *International Conference on Intelligent Robots and Systems on*; 2013:IEEE.

20. Klein G, Murray D. Parallel tracking and mapping for small AR workspaces. *International Symposium on Mixed and Augmented Reality on*; 2007:IEEE.

21. Smeulders AW, Chu DM, Cucchiara R, Calderara S, Dehghan A, Shah M. Visual tracking: An experimental survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2013;36(7):1442–1468.

22. Li S, Yeung DY. Visual object tracking for unmanned aerial vehicles: A benchmark and new motion models. *Conference on Artificial Intelligence on*; 2017:AAAI.

23. Briechle K, Hanebeck UD. Template matching using fast normalized cross correlation. *Proceedings of SPIE - The International Society for Optical Engineering*. 2001;4387:95–102.

24. Baker S, Matthews I. Lucas-kanade 20 years on: A unifying framework. *International Journal of Computer Vision.* 2004;56(3):221–255.

25. Arandjelović O. Automatic vehicle tracking and recognition from aerial image sequences. *International Conference on Advanced Video and Signal Based Surveillance on*; 2015:IEEE.

26. Nguyen HT, Smeulders AW. Fast occluded object tracking by a robust appearance filter. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2004;26(8):1099–1104.

27. Adam A, Rivlin E, Shimshoni I. Robust fragments-based tracking using the integral histogram. *IEEE Computer Society Conference on Computer Vision and Pattern Recognition on*; 2006:IEEE.

28. Comaniciu D, Ramesh V, Meer P. Real-time tracking of non-rigid objects using mean

shift. *Conference on Computer Vision and Pattern Recognition on*; 2000:IEEE.

29. Oron S, Bar-Hillel A, Levi D, Avidan S. Locally orderless tracking. *International Journal of Computer Vision.* 2015;111(2):213–228.

30. Ross DA, Lim J, Lin RS, Yang MH. Incremental learning for robust visual tracking. *International Journal of Computer Vision.* 2008;77(1-3):125–141.

31. Kwon J, Lee KM, Park FC. Visual tracking via geometric particle filtering on the affine group with optimal importance functions. *Conference on Computer Vision and Pattern Recognition on*; 2009:IEEE.

32. Maggio E, Cavallaro A. Multi-part target representation for color tracking. *International Conference on Image Processing on*; 2005:IEEE.

33. Cao X, Lan J, Yan P, Li X. Vehicle detection and tracking in airborne videos by multi-motion layer analysis. *Machine Vision and Applications.* 2012;23(5):921–935.

34. Shi J, Tomasi C. Good features to track. *Conference on Computer Vision and Pattern Recognition on*; 1994:IEEE.

35. Kim Y, Jung W, Bang H. Visual target tracking and relative navigation for unmanned aerial vehicles in a GPS-denied environment. *International Journal of Aeronautical and Space Sciences.* 2014;15(3):112–121.

36. Yang F, Lu H, Yang MH. Robust superpixel tracking. *IEEE Transactions on Image Processing.* 2014;23(4):1639–1651.

37. Lu H, Wang D, Zhang R, Chen YW. Video object pursuit by tri-tracker with on-line learning from positive and negative candidates. *IET Image Processing.* 2011;5(1):101–111.

38. Xiaoyan J, Shiru Q. A target tracking algorithm based on mean shift with feature fusion. *Chinese Control Conference on*; 2015:IEEE.

39. Szottka I, Butenuth M. Advanced particle filtering for airborne vehicle tracking in urban areas. *IEEE Geoscience and Remote Sensing Letters.* 2013;11(3):686–690.

40. Rodríguez-Canosa GR, Thomas S, Del Cerro J, Barrientos A, MacDonald B. A real-time method to detect and track moving objects (DATMO) from unmanned aerial vehicles (UAVs) using a single camera. *Remote Sensing.* 2012;4(4):1090–1111.

41. Kanade T, Collins R, Lipton A, Burt P, Wixson L. Advances in cooperative multi-sensor video surveillance. *Proceedings of DARPA Image Understanding Workshop on*; 1998.

42. Lowe DG. Distinctive image features from scale-invariant keypoints. *International Journal of Computer Vision.* 2004;60(2):91–110.

43. Zhang H, Wu K. A vehicle detection algorithm based on three-frame differencing and background subtraction. *International Symposium on Computational Intelligence and Design on*; 2012:IEEE.

44. Liu L, Sang N, Huang R. Background subtraction using shape and colour information. *Electronics Letters.* 2010;46(1):41–43.

45. Stauffer C, Grimson WEL. Adaptive background mixture models for real-time tracking. *Computer Society Conference on Computer Vision and Pattern Recognition on* 1999:IEEE.

46. Lee DS. Effective Gaussian mixture learning for video background subtraction. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2005;27(5):827–832.

47. Horprasert T, Harwood D, Davis LS. A statistical approach for real-time robust background subtraction and shadow detection. *International Conference on Computer Vision on*; 1999:IEEE.

48. Cheng G, Han J. A survey on object detection in optical remote sensing images.

ISPRS Journal of Photogrammetry and Remote Sensing. 2016;117:11–28.

49. Rodriguez J, Castiblanco C, Mondragon I, Colorado J. Low-cost quadrotor applied for visual detection of landmine-like objects. *2014 International Conference on Unmanned Aircraft Systems on*; 2014:IEEE.

50. Malek S, Bazi Y, Alajlan N, AlHichri H, Melgani F. Efficient framework for palm tree detection in UAV images. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.* 2014;7(12):4692–4703.

51. Song B, Li X. Power line detection from optical images. *Neurocomputing.* 2014;129:350–361.

52. Katrasnik J, Pernus F, Likar B. A survey of mobile robots for distribution power line inspection. *IEEE Transactions on Power Delivery.* 2009;25(1):485–493.

53. Chen B, Chen Z, Deng L, Duan Y, Zhou J. Building change detection with RGB-D map generated from UAV images. *Neurocomputing.* 2016;208:350–364.

54. Popp M, Granacher R, Trommer G. Automatic detection of complex shaped buildings in aerial images to support the navigation of micro aerial vehicles in urban environment. *Gyroscopy and Navigation.* 2015;6(1):1–8.

55. Flores G, Zhou S, Lozano R, Castillo P. A vision and GPS-based real-time trajectory planning for a MAV in unknown and low-sunlight environments. *Journal of Intelligent & Robotic Systems.* 2014;74(1-2):59–67.

56. Yang S, Scherer SA, Schauwecker K, Zell A. Autonomous landing of MAVs on an arbitrarily textured landing site using onboard monocular vision. *Journal of Intelligent & Robotic Systems.* 2014;74(1-2):27–43.

57. Moranduzzo T, Melgani F. Automatic car counting method for unmanned aerial vehicle images. *IEEE Transactions on Geoscience and Remote Sensing.*

2013;52(3):1635–1647.

58. Tuermer S, Kurz F, Reinartz P, Stilla U. Airborne vehicle detection in dense urban areas using HoG features and disparity maps. *IEEE Journal of Selected Topics in Applied Earth Observations and Remote Sensing.* 2013;6(6):2327–2337.

59. Liu K, Mattyus G. Fast multiclass vehicle detection on aerial images. *IEEE Geoscience and Remote Sensing Letters.* 2015;12(9):1938–1942.

60. Chen X, Xiang S, Liu CL, Pan CH. Vehicle detection in satellite images by hybrid deep convolutional neural networks. *IEEE Geoscience and Remote Sensing Letters.* 2014;11(10):1797–1801.

61. Kluckner S, Pacher G, Grabner H, Bischof H, Bauer J. A 3D teacher for car detection in aerial images. *International Conference on Computer Vision on*; 2007:IEEE.

62. Bradski G, Kaehler A. *Learning OpenCV: Computer vision with the OpenCV library.* Sebastopol: O'Reilly Media, Inc.; 2008.

63. Henriques JF, Caseiro R, Martins P, Batista J. High-speed tracking with kernelized correlation filters. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2014;37(3):583–596.

64. Possegger H, Mauthner T, Bischof H. In defense of color-based model-free tracking. *Conference on Computer Vision and Pattern Recognition on*; 2015:IEEE.

65. Zhang J, Ma S, Sclaroff S. MEEM: robust tracking via multiple experts using entropy minimization. *European Conference on Computer Vision on*; 2014:Springer.

66. Wang N, Shi J, Yeung DY, Jia J. Understanding and diagnosing visual tracking systems. *International Conference on Computer Vision on*; 2015:IEEE.

67. Danelljan M, Häger G, Khan F, Felsberg M. Accurate scale estimation for robust visual tracking. *British Machine Vision Conference on*; 2014:BMVA Press.

68. Sirmacek B, Unsalan C. Urban-area and building detection using SIFT keypoints and graph theory. *IEEE Transactions on Geoscience and Remote Sensing.* 2009;47(4):1156–1167.

69. Özcan AH. 3D Object Detection and Representation in Remote Sensing: Probabilistic Methods and Applications. PhD thesis. Yeditepe University. Istanbul; 2017.

70. Leibe B, Leonardis A, Schiele B. Combined object categorization and segmentation with an implicit shape model. *Toward category-level object recognition*; 2006:508–524.

71. Bar-Shalom Y, Li XR. *Estimation and tracking- Principles, techniques, and software.* Norwood: Artech House, Inc.; 1993

72. STM32F745xx, STM32F746xx Datasheet; [cited 2020 17 January]. Available from: https://www.st.com/resource/en/datasheet/DM00166116.pdf.

73. Chan E. FatFs - Generic FAT File System Module; [cited 2020 17 January]. Available from: http://elm-chan.org/fsw/ff/00index_e.html.

74. OV9655 Color CMOS SXGA (1.3 MegaPixel) CAMERACHIP with OmniPixel Technology; [cited 2020 17 January]. Available from: http://www.arducam.com/downloads/modules/OV9655/ov9655_full.pdf.

75. File Exchange - MATLAB Central; [cited 2020 17 January]. Available from: https://www.mathworks.com/matlabcentral/fileexchange/.

76. CMSIS DSP Software Library; [cited 2020 17 January]. Available from: http://www.keil.com/pack/doc/CMSIS/DSP/html/index.html.

77. Kim Z, Malik J. Fast vehicle detection with probabilistic feature grouping and its application to vehicle tracking. *International Conference on Computer Vision on*; 2003:IEEE.

78. Canny J. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 1986;(6):679–698.

79. Dalal N, Triggs B. Histograms of oriented gradients for human detection. *Conference on Computer Vision and Pattern Recognition on*; 2005:IEEE.

80. Rothe R, Guillaumin M, Van Gool L. Non-maximum suppression for object detection by passing messages between windows. *Asian Conference on Computer Vision on*; 2014:Springer.

81. Press WH, Teukolsky SA, Vetterling WT, Flannery BP. *Numerical recipes in C.* Cambridge: Cambridge University Press; 1988.

82. Box GE. A note on the generation of random normal deviates. *The Annals of Mathematical Statistics* 1958;29:610–611.

83. Marsaglia G, Tsang WW, et al. The ziggurat method for generating random variables. *Journal of Statistical Software.* 2000;5(8):1–7.

84. Schraudolph NN. A fast, compact approximation of the exponential function. *Neural Computation.* 1999;11(4):853–862.

85. Li A, Lin M, Wu Y, Yang MH, Yan S. Nus-pro: A new visual tracking challenge. *IEEE Transactions on Pattern Analysis and Machine Intelligence.* 2015;38(2):335–349.

86. Robicquet A, Sadeghian A, Alahi A, Savarese S. Learning social etiquette: Human trajectory understanding in crowded scenes. *European Conference on Computer Vision on*; 2016:Springer.

87. Collins R, Zhou X, Teh SK. An Open Source Tracking Testbed and Evaluation Web Site. *Workshop on Performance Evaluation of Tracking and Surveillance on*; 2005:IEEE.

88. Zhu P, Wen L, Bian X, Ling H, Hu Q. Vision meets drones: A challenge. *arXiv*

preprint arXiv . 2018;1804:07437

89. Mueller M, Smith N, Ghanem B. A benchmark and simulator for uav tracking. *European Conference on Computer Vision on*; 2016:Springer.

90. Wu Y, Lim J, Yang MH. Online object tracking: A benchmark. *Conference on Computer Vision and Pattern Recognition on*; 2013:IEEE.

APPENDIX A: SEQUENCE LIST

All sequences that are used in this study are listed in tabular form in Tables A.1 to A.4.

Table A.1. Sequence list.

	Occlusion	Low Res.	Similar Objects	Scale /Aspect Change	Source Dataset
01-Light_video00012				✓	Alov300
01-Light_video00014					Alov300
01-Light_video00024					Alov300
09-Confusion_video00002			✓		Alov300
11-Occlusion_video00006	✓				Alov300
11-Occlusion_video00007	✓				Alov300
airplane_002				✓	NusPro
airplane_003				✓	NusPro
airplane_004				✓	NusPro
airplane_005	✓			✓	NusPro
airplane_006				✓	NusPro
airplane_007				✓	NusPro
airplane_008				✓	NusPro
airplane_009				✓	NusPro
airplane_010				✓	NusPro
airplane_011				✓	NusPro

Table A.2. Sequence list cont.

	Occlusion	Low Res.	Similar Objects	Scale /Aspect Change	Source Dataset
airplane_012				✓	NusPro
airplane_013					NusPro
airplane_014				✓	NusPro
airplane_015				✓	NusPro
airplane_017	✓			✓	NusPro
airplane_018				✓	NusPro
airplane_020				✓	UAV123
boat2				✓	UAV123
boat4				✓	UAV123
boat5				✓	UAV123
boat6				✓	UAV123
boat8				✓	UAV123
bookstore					Stanford
bookstore_2			✓		Stanford
bookstore_3					Stanford
bookstore_4			✓		Stanford
bookstore_5			✓		Stanford
bookstore_6	✓				Stanford
car_5					OTB70
car_6	✓			✓	OTB70
car_8				✓	OTB70
car1				✓	UAV123

Table A.3. Sequence list cont.

	Occlusion	Low Res.	Similar Objects	Scale /Aspect Change	Source Dataset
car10	✓				UAV123
car11	✓	✓			UAV123
car14		✓		✓	UAV123
car2	✓				UAV123
car3					UAV123
car4	✓				UAV123
deathcircle_1					Stanford
deathcircle_2					Stanford
deathcircle_3					Stanford
egtest01		✓	✓		Vivid
egtest02		✓	✓		Vivid
egtest03	✓	✓			Vivid
egtest04	✓	✓			Vivid
egtest05			✓		Vivid
ManRunning1	✓			✓	OTB70
person1					UAV123
person11	✓				UAV123
person15			✓		UAV123
person2					UAV123
person22		✓		✓	UAV123
person23			✓	✓	UAV123
person3					UAV123
person6			✓	✓	UAV123
RcCar5				✓	OTB70
RcCar8			✓	✓	OTB71

Table A.4. Sequence list cont.

	Occlusion	Low Res.	Similar Objects	Scale /Aspect Change	Source Dataset
RcCar9				✓	OTB72
soccer_001	✓			✓	UAV123
soccer_002	✓		✓	✓	UAV123
soccer_003	✓				UAV123
soccer_004	✓		✓		UAV123
soccer_005	✓		✓	✓	UAV123
soccer_006	✓		✓		UAV123
soccer_007	✓			✓	UAV123
soccer_008			✓	✓	UAV123
soccer_009			✓	✓	UAV123
soccer_010	✓		✓		UAV123
SpeedCar2		✓			OTB70
SpeedCar4			✓		OTB70
uav0000126_07915_s					VisDrone
uav0000238_01280_s				✓	VisDrone
uav0000252_00001_s	✓				VisDrone
uav0000303_01250_s				✓	VisDrone