Computational Geometric-Based Visual Shape Tracker.

by
Pasan Bandara

A thesis
submitted to the Faculty of Graduate Studies,
in Partial Fulfilment of the Requirements for the degree of

Master of Science
in
Electrical and Computer Engineering

Department of Electrical and Computer Engineering
University of Manitoba
Winnipeg, Manitoba R3T 5V6 Canada

Abstract

The main intent of multiple shape tracking is to allocate distinct track identities for all the salient shapes in a video sequence. The approach used in this problem is 'Track by Detection' scheme, where it needs to first, find shapes in a frame, map shapes to appropriate tracking tracks, and maintain the data continuity model throughout the video sequence. This thesis proposes a process to address shape tracking by maintaining a data continuity model based on shape relative proximity, shape context descriptor, and spatial color variation of a shape. Shape intersection is used to calculate the shape proximity. A spatial color analyzer is used to compare shape's color variations between two consecutive frames, and shape similarity comparison is done by using the shape context descriptor. These values are then used to build and maintain the data continuity model throughout the video sequence. Besides, this also introduces a methodology to reestablish tracking identity when the detection is failed or occluded temporarily between small numbers of frames. The proposed method can be implemented using both online and offline approaches. To assess the proposed method's robustness, we evaluated the online method in both VOT-ST2020 (Visual Object Tracking Challenge - Short Term) and MOT20 (Multiple Object Tracking) benchmark datasets. It was observed that in both benchmarks, the proposed approach can deliver state-of-the-art results.

Keywords: Computer vision, shape intersection, shape descriptor, description, feature vector, shape persistence, object tracking, Delaunay triangulation, similarity measure, Kalman estimation, shape analysis, color variation.

Acknowledgements

First of all, I would like to express my sincere gratitude to my advisor, Prof. James Peters, for allowing me to be part of his research group. I am thankful for his thorough advice, steady inspiration, insightful feedback, solid encouragement, and helpful suggestion. Without him, the completion of this thesis would not have been possible.

I would also like to thank my examination committee members, Prof. Arkady Major, Prof. Sherif Sherif, and Prof. Ji Hyun Ko, for their additional guidance and support.

I wish to thank all my colleagues and friends in the Computational Intelligence Laboratory for their generous help during my research work. I want to thank all professors in our department for their exciting and helpful courses. I also wish to acknowledge the kind help and support from the Department of Electrical and Computer Engineering staff.

Finally, I would like to thank my parent, siblings, and friends for their constant love, care, and support.

Contents

List of Tables

List of Figures

1 Introduction

The focus of this thesis is on shape tracking in video sequences (defined by a mask around an object of interest) with attention to shape structure and color. Shape or object tracking generally involves locating the object and applying a distinct identity across all of the frames in a video sequence. Shape tracking is a dominant research area owing to its many use cases such as motion prediction [11], self driving vehicles [12], advanced video processing systems [13], behavior analysis [14] and military applications [15].

Typically, in a digital video sequence, the object-of-interest visual appearance may change over time primarily due to changes in illumination, object rotation and/or scaling (geometrical orientation), occlusions and color. This is the major obstacle that any of the object trackers need to overcome. There are two major branches within object tracking. An initial approach to object tracking is to use the localization information (shape bounding box coordinates) present in the first frame. A more advanced approach to object tracking is multiple objects tracking that uses prior learned information to detect, identify and track frame objects. Nowadays, object detectors are used by almost all multiple object trackers to locate shapes in each frame (also known as a "track by detection" [16] framework) and then employ a method to manage and track objects over time. Another way to categorize object trackers is based on how they utilize the detection information in a video sequence. Trackers that use all detection information to map and assign unique identities to detections in a video sequence are referred to as offline method-based trackers [16] while online methods rely only on data from shape detection up until to the present frame. In the domain of practicality, for real-time applications, online-based methods are the best to track multiple objects in real-time since it only requires detection information up to the current frame [17]. Since offline methods provide more knowledge about all of the objects in the sequence, they can be more precise than online methods.

1.1 Thesis Objective

This research's main objective is to elevate the overall performance of multi-object visual trackers, both online and offline, by providing solutions to some inherent problems in estimating relative shape locations and feature dissimilarities.

In this research, I propose a novel approach to estimate the region of interest for a given shape within two consecutive video frames. This approach is a combination of shape intersection and shapes collision detection algorithms. In addition to that, I propose a method to combine standard shape context descriptor with SIFT descriptor to compute shape similarities with a minimum number of computations. Finally, I introduce a method to re-establish tracking identity when there is a short term detection failure. For this, I use Scale Invariant Feature Transform (SIFT) [59] descriptor with a Kalman filter.

In summary, the contributions presented in this thesis are:

- Introduces a shape region estimation method that based on computational geometric perspective,
- A method to combine SIFT keypoint descriptor to the shape context descriptor,
- A method to address shape occlusion in stable tracks.

1.2 Organization of the Thesis

This thesis is organized as follows: Chapter 2 provides a detailed background, a literature review of object tracking approaches and the proposed online and offline shape tracking framework. Chapter 3,4 and 5 discuss the computational geometric-based region estimator, shape context descriptor with a spatial color analyzer, and Kalman filter-based shape trajectory estimator, respectively. Chapter 6 discusses the proposed tracker evolutions metrics, results and finally, chapter 7 discusses the conclusions and future directions of the research work presented in this thesis.

2 Background

Multiple Object Tracking (MOT) has become one of the most popular, widely studied subject areas in recent past years. Some MOT algorithms are for tracking only a specific object. A great example of this is multiple human tracking [18], [19], [20] . But some algorithms are general [21], [22] [23]. That means they can be used for any type of object tracking. Most MOT algorithms are based on a tracking-by-detection framework. In this approach, shape detection must be handled by an object detector.

Some approaches that use the tracking-by-detection framework make use of all the detected shapes in the video sequence, while others only use shapes that are detected up to that frame. The first methods are known as global or offline approach, while the latter are classified as online approach [16]. Bayesian filtering-based tracking [21], multiple hypotheses tracking (MHT) [24] , network flow optimization [18] and graph-based clustering [19] , are among the accepted approaches in offline multiple object tracking algorithms. Without directly assigning detections to tracks, some global MOT algorithms [21] first assign detections to tracklets.

Most of the global MOT algorithms apply detected shapes to tracklets before directly assigning them to tracks. (A tracklet is a short track between a few continuous frames). Later these tracklets are assigned to tracks to identify the prolonged dissimilarities in candidate shapes. Object appearance and motion are the two most commonly considered factors when computing the similarity or difference between object detections in various frames. Conventional methods like the distance between deeply learned features [18], [19], [24], the separation between Red, Green and Blue color channel histograms [21] , and machine learning-based person detection and identification [18] are used for appearance-based distance calculation methods. The distance between predicted and actual locations [19], [24] , Spatio-temporal distance between object bounding boxes [18] and point motion matching [21], are used to evaluate matching costs based on motion. Moreover, fixing errors in tracklets to track assignments yield highly accurate results. The point change analyzing framework [21] is used to address this issue. Tracking results can be improved by using multiple detectors. Hence some methods [19] utilize this fact though it increases the processing time.

Online MOT algorithms are the best suit for real-time applications. Because they use information not from all frames but up to the current frame. Most of the time, they match the detections to tracks using some form of pairwise cost calculations. Offline MOT algorithms are the opposite of this. They use matching costs from every frame and use a global optimization approach to assign shapes or tracklets to tracks. However, there is a similarity between the online and global algorithms. Both types calculate the matching cost using the shape features and location data from the detection.

The proposed method can be implemented in both ways - as an algorithm based on online pairwise cost calculation or offline pairwise cost calculation followed by global optimization. Let's discuss several similar approaches in detail. Sarthak Sharma et al. In [25] introduced a new tracking framework based on 2D and 3D localization information, object shape, pose, and machine learning-based keypoints analyzing followed by Hungarian assignment algorithm [26]. Here, detection accuracy is improved using two object detectors [27] , [28]. In [29] to calculate the dissimilarity measure, RGB color histogram Chi-Square similarity measure and Spatio-temporal cosine distance of the shape is used. Sudden changes in the camera may result in false matching when calculating matching cost. Comparative motion variation between shape masks [23] can be used in compensation for this global motion. To reduce the false positives and false negatives, in [23] proposed an event aggregation as a post-processing step. Reinforcement learning-based closeness computation is used in [30]. Presenting an online method, Amir et al. [20] introduced a new way to calculate similarity scores based on appearance and motion. This method uses three Recurrent Neural Networks (RNN) to calculate the similarity score. In [31], pairwise costs are calculated using a Recurrent Autoregressive Network (RAN) that uses machine learning-based person detection and identification results and shape location data. Chen et al. [32] proposed a new approach to minimize the false negatives and calculate the pairwise cost using the bounding boxes from the object detector and the expected shape localization data (from each track's history). Here the machine learning-based person detection and identification results are taken as the basis of pairwise cost. The non-maximum suppression method is used to filter out the detections based on detection and track dependence. Apart from size and motion-based matching, in [33], uses a trained siamese network to minimize identity switches that generally occur in tracking. This network does the matching based on historical appearance. In [34] uses two deep networks to calculate the pairwise cost. Here, the temporal attention network uses the spatial attention network's information to calculate the final pairwise cost between each track and detection. Here, when comparing track history and detections, the attention network follows siamese architecture. This Siamese approach can be used to assign detections to tracks that are not matched by the single shape tracker (a handcrafted version of [35] using Histogram Of gradients (HoG)).

In [36], for each instance of a person, detection confidence and intersection over union are used for cost computation, and in addition to that, the authors also dynamically initiate a subnetwork to estimate the next positioning coordinates. A discriminative appearance method for individual tracks is used in [37]. Here, detection is used as a possible hit while using the area around it as a negative hit. Besides, authors in [37] use Spatio-temporal matching (STM) based on object size and position. Then, in a multiplicative way, they combine these steps.

Apart from online methods that are based on pairwise cost, filter update-based tracking method is also used to design multiple object trackers. In [38], the expected position of a shape is predicted in a Gaussian Mixture Probability Hypothesis Density filter. Zeyu et al. [39] introduced a similar method using the Monte Carlo PHD filter. Dictionary matching is used to describe the appearance attribute, which is defined using color (RGB) histogram and Histogram of Oriented Gradients (HoG) clustering. In [40], the authors use a Poisson multi-Bernoulli mixture filter. Here, they have used predictions of object 3D coordinates from a pre-trained network. Authors in [41], use an Extended Kalman Filter version to track object 2D image coordinates, object size, and the 3D coordinates using stereo matching along with an ego-motion calculation. Authors in [30] suggest an online tracking framework that uses the Markov Decision Process to change the track's status (active, inactive, and lost) for the individual frame using a shape appearance matching. An energy minimization approach is proposed in [22]. According to this model, the appearance model and motion model are used to calculate energy terms. Though most pairwise methods [20], [25], [29], [30], [37] use the Hungarian assignment [26], some [31] use the Greedy assignment to assign the perfect match for tracks and detections. The hierarchical association is used in [32], and it is based on two distinct pairwise matching costs.

MOT methods [33], [34] based on deep learning are better at giving accurate tracking results. However, compared to the traditional methods that are based on handcrafted features, deep learning-based MOT methods consume more time. They require a fairly large amount of dataset as the training data. Even though some deep learning-based methods [32] obtain real-time performance with greater accuracy, they still need specialized hardware with high system requirements to get higher running speed. Appearance learning-based methods [37] and some filter update methods [38], [39] are capable of getting higher accuracy at the expense of high computational cost. So there is a high demand for simpler and efficient methods for real-time applications and post-processing applications that are better at getting higher accuracy with a lower computational cost.

This research attempts to resolve this problem by proposing a new method that combines shape feature analyzing methods that are handcrafted, computationally less expensive to calculate, and easy to implement. Primarily, the proposed method is based on the likelihood of shape location, shape structure, color, and motion. Multiple hand-crafted features are used in car tracking methods [25], [29]. In terms of precision, our method outperforms them in terms of accuracy. To compute appearance similarity, authors in [25] have used deeply learned feature-based matching [29]. In our method, shape color matching and shape keypoint matching is defined when the relative shape location's likelihood is within the expected range to the previous location. We demonstrate that the proposed approach is capable of achieving state-of-the-art results among popular object tracking benchmark datasets.

2.1 Shape Tracking Method

This section briefly describes the online and offline methods for the proposed shape tracker. As for the online method, I use a shape detector to detect and produce salient (shape(s) of interest) binary shape masks in a video frame and associate these shapes to tracks based on a data continuity model that is calculated within the shapes in the current frame against to the previous frames in the memory. This data continuity model can be represented using three sub-modules. They are the shape region estimator, keypoints-based shape context descriptor, and the spatial color analyzer. Shape intersection is the primary decision-making step for the region estimator. In contrast, the spatial color analyzer uses a cluster-based reduction in RGB color space followed by an affine transformation to analyze the shape's color variation. Finally, the shape context descriptor is based on the concepts borrowed from well-known keypoint descriptors. For the offline method, in addition to the above data continuity model, I use a Kalman-based trajectory estimator to address shape occlusion in any stable tracks. Since the goal of the online approach is to process as many frames as possible within a certain period, the computational complexity should stay lower. Because of this reason, the Kalman filter module is only used in the offline model owing to its relatively high computational cost. Both online and offline overall tracking processes are presented as flow charts in Figure 1 and Figure. 2 respectively.

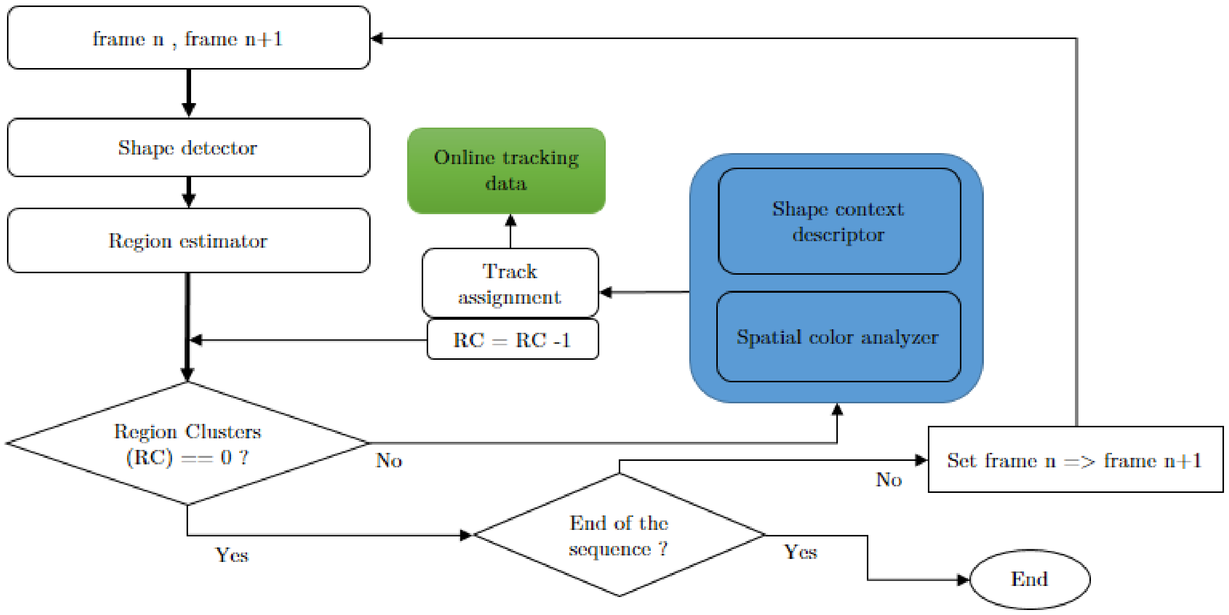


Figure 1: Implementation of the online visual tracker

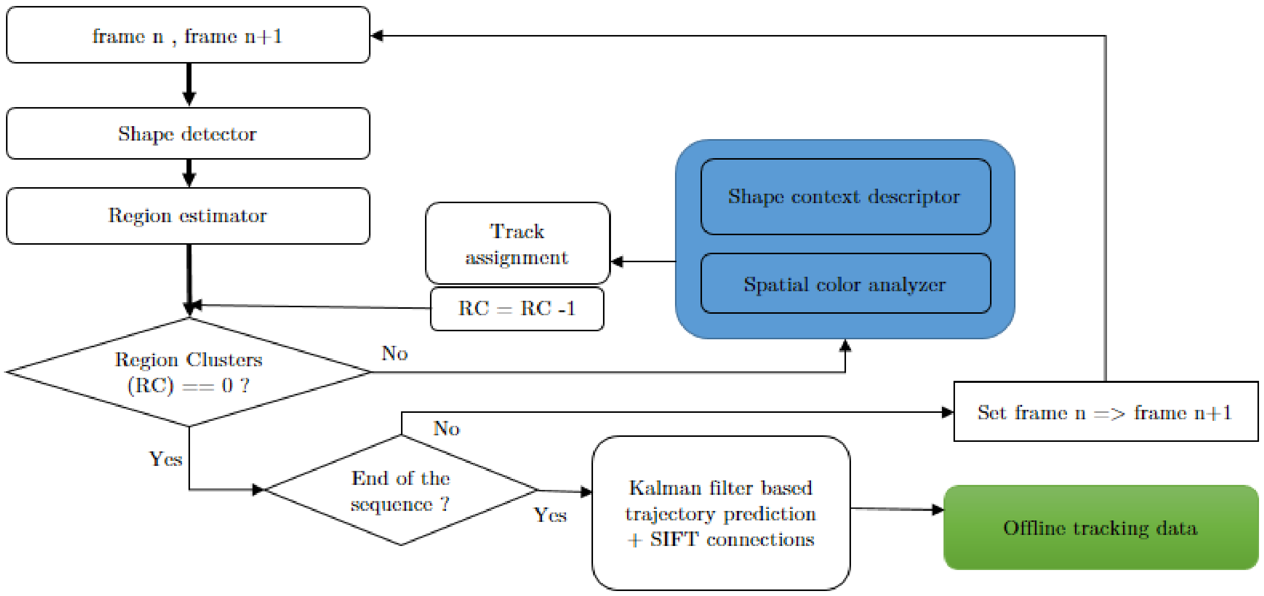


Figure 2: Implementation of the offline visual tracker

3 Shape Region Estimation

3.1 Introduction

In this module, we introduce the proposed region estimator, which is based on computational geometric concepts to estimate the position of a shape in consecutive video frames depending on the shape behavior. For this, we explore methods to isolate shapes in a video sequence, efficient edge detection algorithms, shape contour approximation methods, shape centroid, and Delaunay triangulations. This module is one of the three modules that were used to track shapes in this research. This module's output is the input to the other two modules, which are shape context descriptor and spatial color analyzer. In this chapter, Section 3.2 briefly outlines the fundamental concepts of computational topology and geometry, while Section 3.3 provides the proposed region estimator module for shape tracking.

3.2 Preliminaries

3.2.1 Foreground Shape Detector

There are several methods to isolate foreground shapes from the background in a video sequence. This section explores the two major categories in the domain of computer vision. The first one is a frame subtraction-based background removal method to isolate foreground moving objects, and the second one is to use a pre-trained object detector to isolate shapes. Frame subtraction-based background removal method can be further divided into two major subcategories [42]: native or conventional background subtractor and an adaptive background subtractor (Figure 3). Native background subtraction methods work well for fixed camera viewpoints that have minor light changes throughout the entire video sequence. This method takes a static image representing the background. It then computes the absolute difference between the current frame in the video sequence and the static background frame to detect any pixel changes. From this, it can be argued that given there are negligible light differences between those two frames (in a fixed camera viewpoint), these pixel changes indicate moving objects. The significant advantage of this method is that it is easier to implement. However, due to the nature of the resultant image from this process containing a significant amount of pixel noise, the object detections' accuracy falls dramatically.

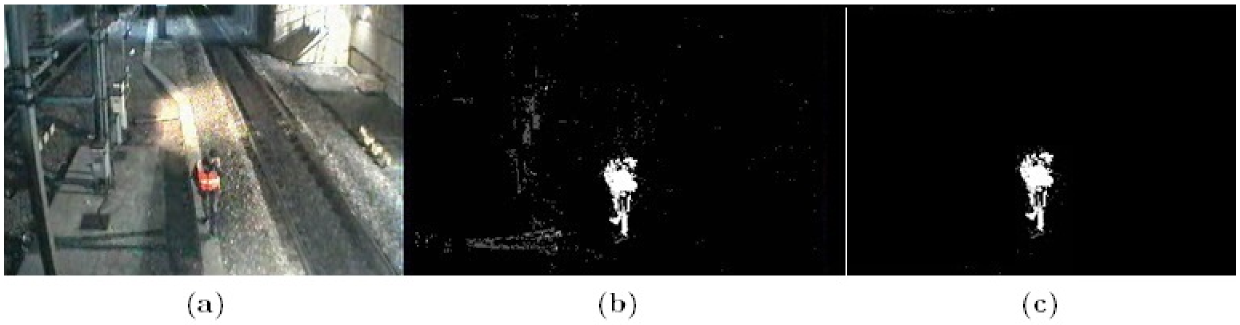


Figure 3: Different background substraction approaches. (a) input frame from a static video sequence [2]. (b) result obtained from first frame substraction (native) method (c) results by using the frame difference algorithm (adaptive background substraction).

Unlike conventional background subtraction methods, adaptive background subtraction methods initialize and maintain a background model throughout the video sequence. One of the most efficient (in terms of average execution time, average CPU usage, and average memory usage) adaptive background subtraction algorithm used in this research as an object detector was the "Frame Difference" algorithm [43]. In this method, the background model is maintained by the arithmetic mean of pixels between previous frames in a video sequence. Hence, given a video sequence with n number of total grayscale frames defined by V = $\{Z_1, ..., Z_n\}$,the background model B can be defined by :

$$B = \frac{1}{n}\sum_{t=1}^{n} Z_t, \tag{1}$$

This can be used to initialize the model. To maintain the background model the following equation was used.

$$B_t = (1 - \alpha)B_{t-1} + \alpha Z_t \tag{2}$$

In here, B_t is the background model at time $t \in \{1, n\} \subset \mathbb{Z}$ and $\alpha \in \{0, 1\} \subset \mathbb{R}$ is the learning rate. Even though adaptive background subtraction methods work well compared to the conventional background subtraction method, this process does not provide any good results for unfix (shaky handheld camera viewpoint) video sequences because of the high rate of background pixel change.

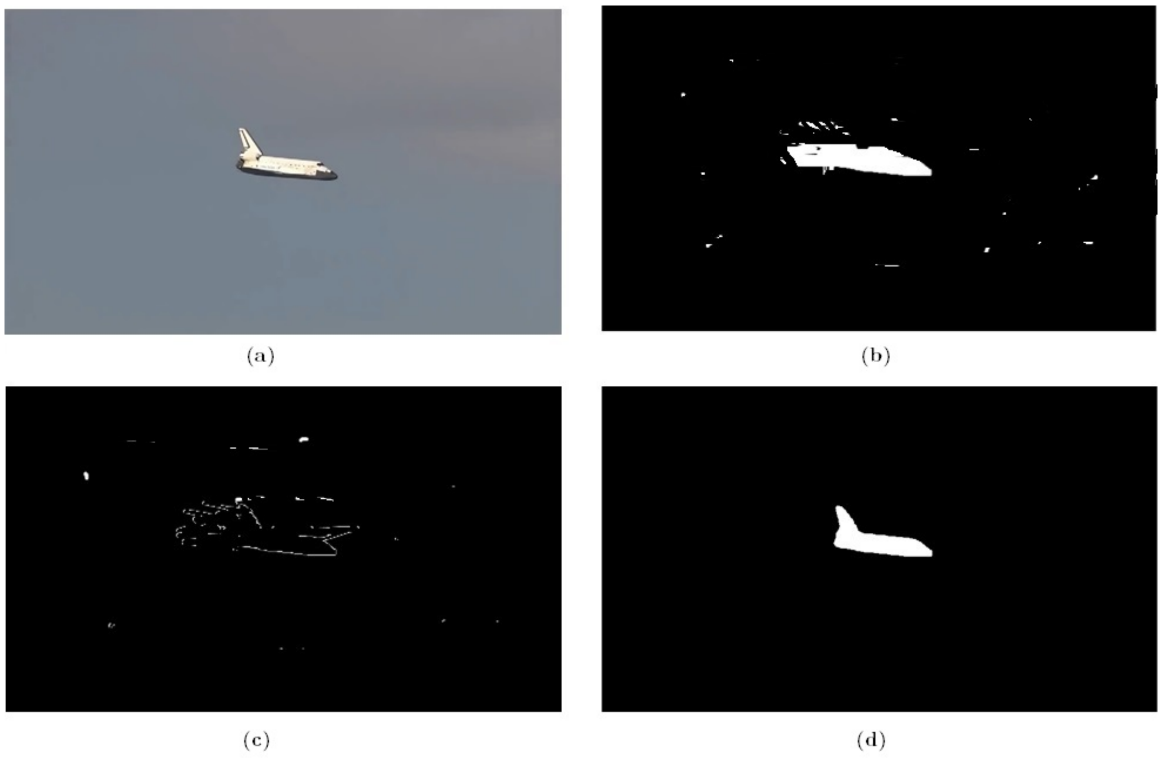


Figure 4: Machine learning vs handcrafted background removal methods. (a) input frame from a dynamic video sequence [3]. (b) native background substraction method (c) adaptive background substraction method. (d) machine learning based "Detectron2" model result.

Compared to which, machine learning-based pre-trained object detectors can handle most of the issues from the methods mentioned above. These object detectors can be used to detect objects regardless of the camera viewpoint or lighting conditions because they are trained on large image dataset(s) that represent real-world scenarios such as different lighting conditions, different scales, and different viewpoints. Machine learning-based object detectors have evolved, and at the time of this research work, the best and reliable open-source object detector was Detectron2 [44] from Facebook AI research. Therefore, in this research, the 'Detectron2' object detection model was used to isolate shapes from their background. For the training dataset, 'Common Objects in Context (COCO)' dataset [45], which contains 300,000 images in 80 object categories, was used.

3.2.2 Edge Detection

Edge detection is an important concept when it comes to shape detection in a video sequence. An edge of an image is a sudden change in intensity. In theory, an edge represents an orthogonal step transition in a set of connected pixels [46]. However, in most practical applications, edges are modeled as having a gradient [46] (figure 5). Thus, rapid changes of luminous intensity can be utilized to detect edges in a digital image. Shapes subjected to unresolved camera focus or surface light refraction can result in invalid shape boundaries that have gradual intensity change. [46].

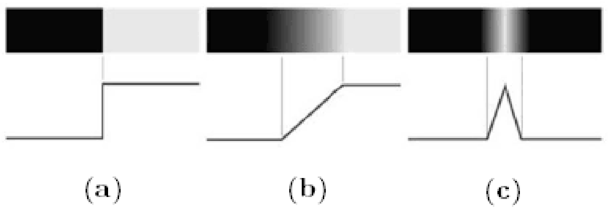


Figure 5: Primary edge types. (a) step edge (b) ramp edge (c) roof edge

The idea behind traditional gradient edge detectors is to match local image segments with specific edge patterns. This can be achieved by observing the minimum and maximum in the first derivative of the image. As regarding to image f(x,y), the gradient of point (x, y) is defined as :

$$\nabla f(x,y) = \left[\frac{\partial f}{\partial x}, \frac{\partial f}{\partial y}\right]^T = [Gx, Gy]^T . \tag{3}$$

The weight of the vector is

$$\nabla f = mag(\nabla f) = \sqrt{(Gx^2 + Gy^2)} \tag{4}$$

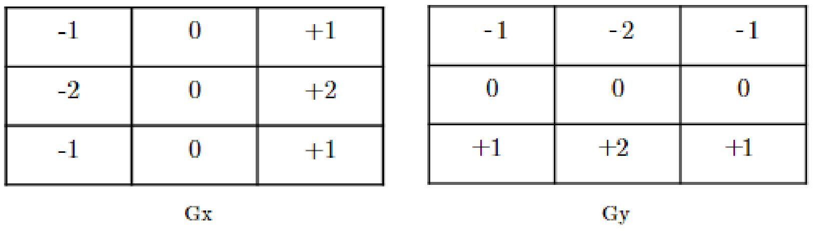
And its direction as:

$$\theta(x, y) = arctan(\frac{Gy}{Gx}) \tag{5}$$

Edge detection operators are represented on a 3x3 grid pattern, making them effective and straightforward to use. There are six major types of edge detection methods: Sobel, Canny, Robert, Laplacian of Gaussian, and Zero crossing.

3.2.2.1 Sobel edge detector

The Sobel operator evaluates a 2-D spatial gradient on an image, highlighting the separation of regions corresponding to edges [47]. On a grayscale image, this will generally evaluate the gradient magnitude per pixel. This can be achieved by performing a 3x3 convolution kernels with the as shown in Figure 6. On the image, this is done in both the x and y directions, with one kernel rotated by 90 degrees.



-1	0	+1
-2	0	+2
-1	0	+1

Gx

-1	-2	-1
0	0	0
+1	+2	+1

Gy

Figure 6: Convolution kernels for Sobel edge detector

3.2.2.2 Canny edge detector

A multi-stage method is used by the Canny operator. For that, firstly, the image is smoothed by Gaussian convolution [48]. The smoothed image is then applied to a simple 2-D first derivative operator to highlight regions of the image with strong first spatial derivatives. In the gradient magnitude image, edges increase ridges (see figure 7). Non-maximal suppression is achieved by making all the pixels to zero that are not actually on the ridge top.

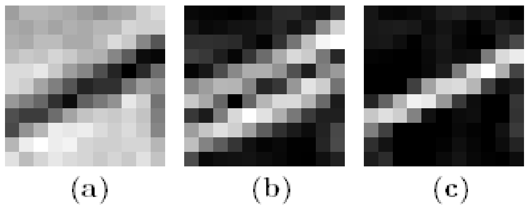


Figure 7: Edge vs ridge detection (at x500 zoom level). (a) input thin line (b) edge detections (c) ridge detections.

3.2.2.3 Robert edge detector

On an image, the Robert Cross operator calculates a 2-D spatial gradient in a simple and fast method. [49]. As an outcome, high spatial frequency regions can be identified, which also lead to edges. The operator's input and output are both grayscale images in its most common usage. The approximate absolute magnitude of the spatial gradient of the input image at that point is expressed by pixel values at each point in the output. As shown in figure 8, this can be performed by a pair of 2x2 convolution kernels. . One kernel is 90 degrees rotated from the other.

+1	0
0	-1

Gx

0	+1
-1	0

Gy

Figure 8: Convolution kernels for Robert edge detector

3.2.2.4 Laplacian of Gaussian

The Laplacian is a 2-D isotropic measure of an image's second spatial derivative [50]. The Laplacian of an image is also used for edge detection as it detects rapid intensity change. The Laplacian is frequently applied to an image that has been smoothed using a Gaussian smoothing filter to reduce sensitivity to noise. In most cases, input and output to this operation take a grayscale image. The Laplacian L(x,y) of an image with pixel intensity values I(x,y) is given by:

$$\nabla^2 L(x,y) = \frac{\partial^2 I}{\partial x^2} + \frac{\partial^2 I}{\partial y^2} \tag{6}$$

A convolution filter can be used to calculate this. Two commonly used kernels are shown in figure 9. The Laplacian can be calculated using standard convolution methods using one of these kernels. These kernels are susceptible to noise since they approximate a second derivative measurement on the image. To reduce the noise, the image first needs to smooth with a Gaussian before applying the Laplacian filter. This will help to reduce the noise component before to the differentiation step.

0	-1	0
-1	4	-1
0	-1	0

-1	-1	-1
-1	8	-1
-1	-1	-1

Figure 9: Convolution kernels to calculate the Laplacian

3.2.2.5 Zero crossing

The zero-crossing detector checks for positions in the Laplacian of an image where the Laplacian value crosses zero [51]. Points where the image's intensity varies rapidly, but they can also happen in positions that are not readily associated with edges. Closed contours often have zero crossings. A zero-crossing detector generates a binary image with single-pixel thickness lines and can be used to locate zero-crossing points. The size of the Gaussian used for this operator's smoothing stage has a significant impact on the zero crossings that follow. Zero-crossing contours are inversely proportional to smoothing. Therefore, when the smoothing is increased, the remaining zero-crossing contours only vaguely correlate to large-scale features in the image.
The analysis result of edge detectors [52] are illustrated in table 1.

Table 1: Proporties of different edge detection algorithms

Operator	Complexity		Noise Sensitivity	False Edges
	Time	Space		
Sobel	Lower	high	Less sensitivity	More
Canny	High	High	Least sensitivity	Least
Robert	High	High	Sensitivity	More
Prewitt	Low	lower	Least sensitivity	More
Laplacian of Gaussian	Low	Least	Least sensitivity	More
Zero crossing	Low	Less	Least sensitivity	More

The following major conclusions can be drawn from the above comparison. Canny method outperforms all the other methods even though its computational complexity is higher. Canny can be used for the detection and extraction of even shapes with weak edges. With any form of edge detection, the computational complexity rises as the spatial resolution increases. Therefore the Canny edge detection method was selected as the preferred edge detection method for this research. Since the selected method has a higher time complexity, it is essential to reduce the spatial resolution to overcome this barrier. Hence, in this research, the previous object detector was used to provide a low noise individual binary mask for each shape to detect edges. Major advantages of Canny edge detection algorithm can be summarized as followings [46]:

- Low error rate.
- The difference between the actual edges and the edge pixels is at minimum.
- Response to a single edge.

Canny edge detection runs in five separate steps:

1. A convolution of the image with a blur kernel,

2. Four convolutions of the image with edge detector kernels,
3. Computation of the gradient direction,
4. Non-maximum suppression, and,
5. Thresholding with hysteresis,

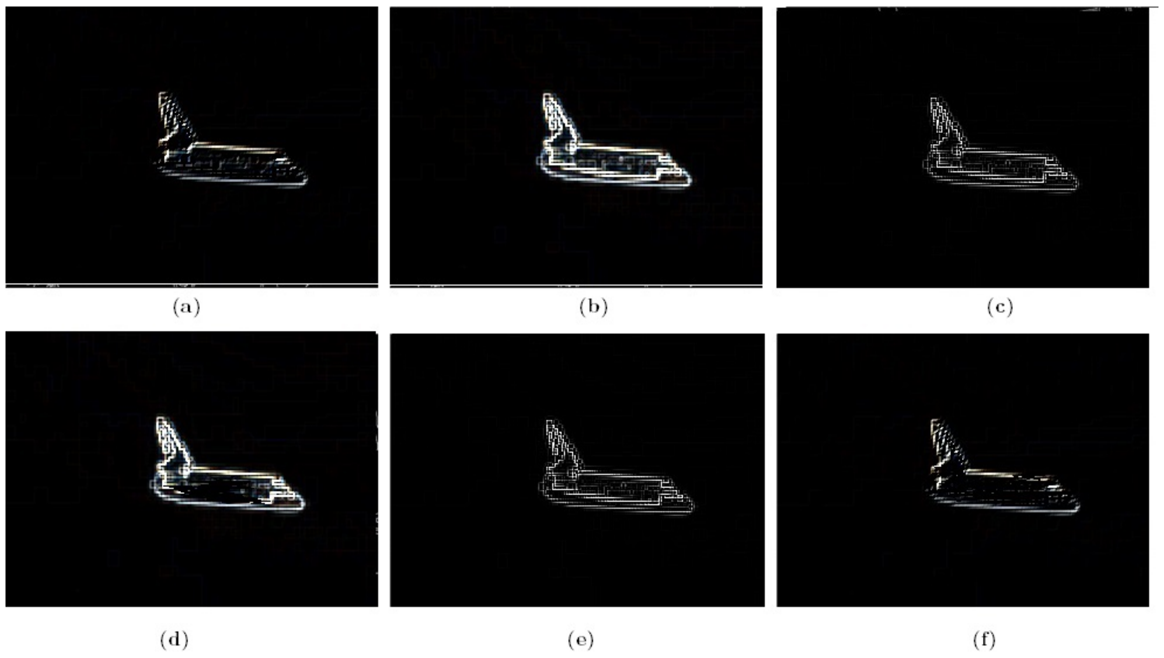


Figure 10: Different edge detection methods. (a) Sobel, (b) Canny, (c) Robert, (d) Prewitt, (e) LoG, (f) zero Crossing

3.2.3 Shape Contour

The contours are a valuable approach for object detection and recognition as well as shape analysis. Contours are essentially a curve that connects all shape boundary continuous points with the same color or intensity. The vertices are abstract elements in discrete mathematics, and the edges are pairs of these elements. Each contour point can be represented as an array of (x,y) positioning coordinates. To find shape contours in a more accurate manner, the following processes were performed.

- Used only binary shape masks to find the shape contour.

- Make sure that the object represents white(255 for 8 bit channel image) and the background should be in black (0).

3.2.3.1 Contour Approximation Method

As mentioned in the previous step, the boundary of a shape with similar intensity values can be represented by a set of (x,y) coordinate points. However, it is not required to store all of the coordinate points in order to emphasize the shape contour in general. Therefore, it is necessary to have a technique to remove the redundant point and compresses the contour to save the memory. In this research, as for the contour approximation method, OpenCV's build in 'cv.CHAIN _APPROX _SIMPLE' method was used to save the memory Figure 11 demonstrates the effect of memory consumption with and without the contour approximation method. In this figure, the first image shows points that returned with 'cv.CHAIN _APPROX _NON' (734 points – without any approximation method) and 'cv.CHAIN _APPROX _SIMPLE' (4 points – with approximation method) is seen in the second image.

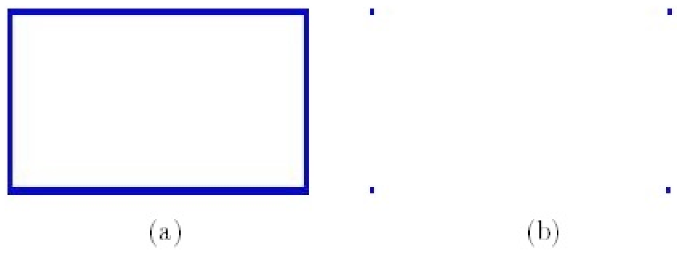


Figure 11: Shape contour approximation demonstration [4]. (a) shape contour with 734 points , (b) shape contour with 4 points

3.2.4 Shape Centroid

In two-dimensional space, a centroid is a point on the shape that corresponds to the mean location of all the points on the shape. As opposed to random shapes, finding the center locations of simple shapes are fairly easy. However, in practice, almost all of the shapes that needed to be tracked can be considered arbitrary shapes. Therefore a reliable method to find the centroids of irregular shapes is essential. In comparison to the real world, computer vision systems view all images as pixels rather than atoms. A blob is a set of related pixels that share a common property (such as grayscale value). In this research, the previously mentioned object detector returns the detected shapes as binary masks; each mask can be considered as blobs. According to the definition of a 2d shape, the centroid of a shape consists of n distinct points $\{x_1, ..., x_n\}$ can be calculated as:

$$C = \frac{1}{n}\sum_{i=1}^{n} x_i \tag{7}$$

Each shape is made up of pixels in computer vision systems, and the centroid is essentially the weighted average of all the pixels in shape. In OpenCV, the easiest way to find the center of a blog is by using moments. The moment is the distribution of matter about a point (image pixel intensities weighted average). The moments can be grouped into two classes. They are spatial moments and central moments. The problem with spatial or regular moments is that they are sensitive to the X and Y position. Therefore central moments are introduced to calculate moments independent of where the shape is in a video frame. The spatial and central moments can be defined as:

$$M_{ij} = \sum_x \sum_y x^i y^j I(x,y) \quad - \quad Spatial\ moments \tag{8}$$

$$\mu_{pq} = \sum_x \sum_y (x-\bar{x})^p (y-\bar{y})^q I(x,y) \quad - \quad Central\ moments \tag{9}$$

$$Where, \quad \bar{x} = \frac{M_{10}}{M_{00}} \bar{y} = \frac{M_{01}}{M_{00}}$$

It can be shown that this only counts points where the image is non-zero. If it is a binary image, then I(x,y) becomes one or zero, and if it is a grayscale image, then we are weighting the point by essentially how heavy it is. In addition to that, if i and j are zero, this would just be counting up the number of non-zeros pixels, which means that $\{M_{00}\}$ is the area of the shape. From this, the position of the centroid can be calculated as:

$$C_x = \frac{M_{10}}{M_{00}} C_y = \frac{M_{01}}{M_{00}} \tag{10}$$

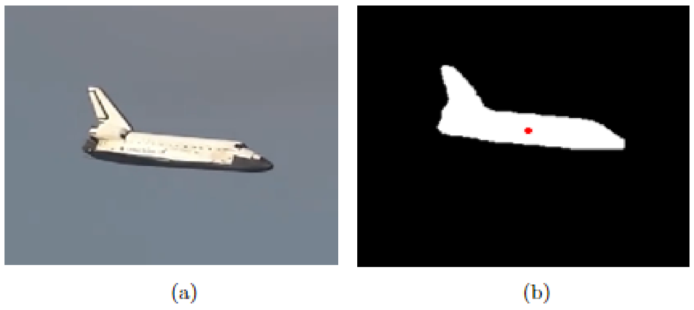


Figure 12: Centroid of a binary mask. (a) input frame (b) shape mask with the centroid (red)

$\{C_x\}$ is the x coordinate, and $\{C_y\}$ is the y coordinate of the centroid, and M denotes the Moment. Hence, throughout this research work, we have used the OpenCV moments method for all centroid calculations, which requires a Raster image (single-channel, 8-bit) as the input image.

3.2.5 Delaunay Triangulation

Delaunay triangulation is an important computational topological concept used in this research to estimate the shape regions within a given video frame, and it has the following properties.

1. In Delaunay triangulation, there are no points inside the each of the triangle's circumscribed circle.

2. The three closest points generate a triangle if each line segment does not intersect.

3. Regardless of the starting point, the final result is steady.

4. Only the adjacent triangle can be affected by adding, removing, or moving a vertex.

5. The triangular mesh's outermost boundary forms a convex polygon shell.

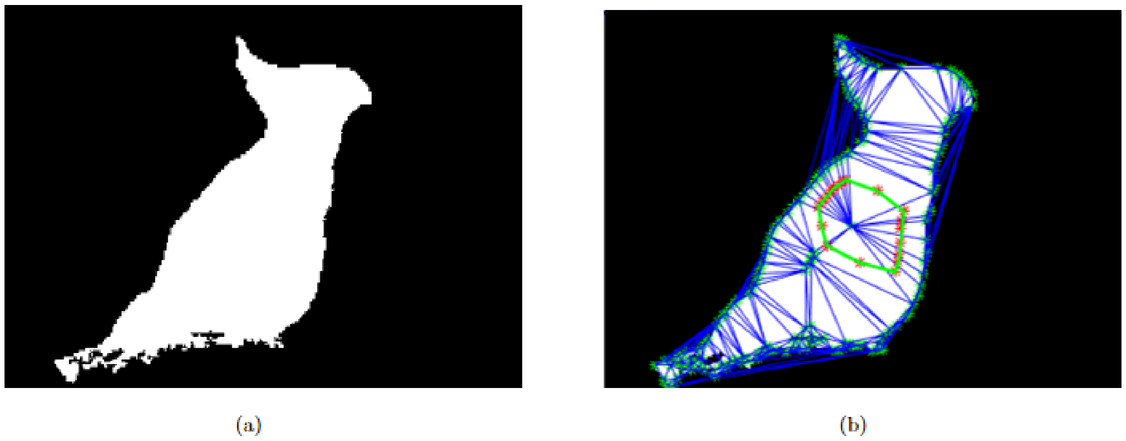


Figure 13: Delaunay triangulation steps. (a) input shape mask, (b) delaunay triangulation with the shape contour points and the shape centroid.

Algorithm 1 introduces an approach to construct Delaunay triangulation for a video frame. In this research, the set of seed points can be considered as the combination of edge vertices, frame corners, and the shape's centroid. Delaunay triangle was constructed between those vertices.

```
Algorithm 1: Construction of Delaunay triangles [53]
1  Select a set of seed points S on a given finite, bounded, planer region of a
     visual scene;
2  Select a seed point p in S;
3  Select 2 seed points q,r that are nearest seed point p;
4  Draw edge $\widehat{pq}$ opposite seed point r;
5  Select closed half plane $\pi_{pq}$ with edge $\widehat{pq}$ so that $r \in \pi_{pq}$;
6  Repeat the edge step for edge $\widehat{pr}$;
7  Repeat the half plane step for edge $\pi_{pr}$;
8  Repeat the edge step for edge $\widehat{qr}$;
9  Repeat the half plane step for edge $\pi_{qr}$;
   Result: Delaunay triangle $\triangle(pqr)$ equals the intersection of the half planes
           $\pi_{pq}$,$\pi_{pr}$,$\pi_{qr}$
10 Repeat the construction for a new Delaunay triangle, starting with the seed
     point step for each seed point p in S;
```

3.3 Region Estimation Method

This section describes the fundamental concepts, theories, and proposed region estimation algorithms used in this research. The deciding intuition to this approach is the shape closeness. If two shapes are close to each other between two consecutive frames, there is a possibility that the first shape in the first frame may represent the same shape in the second frame. More intuitively, given a high frame rate video sequence, the relative shape location difference between two consecutive frames is small or zero if they are close to each other. To express this in another way, if we superimposed two shapes extracted from two consecutive frames into a 2D coordinate plane, these two shapes may intersect if they are close to each other and may not intersect if they are far between these frames. Here, it is assumed that the relative camera viewpoint change is negligible, and the high frame rate is relative to the content in that video sequence (figure 14).

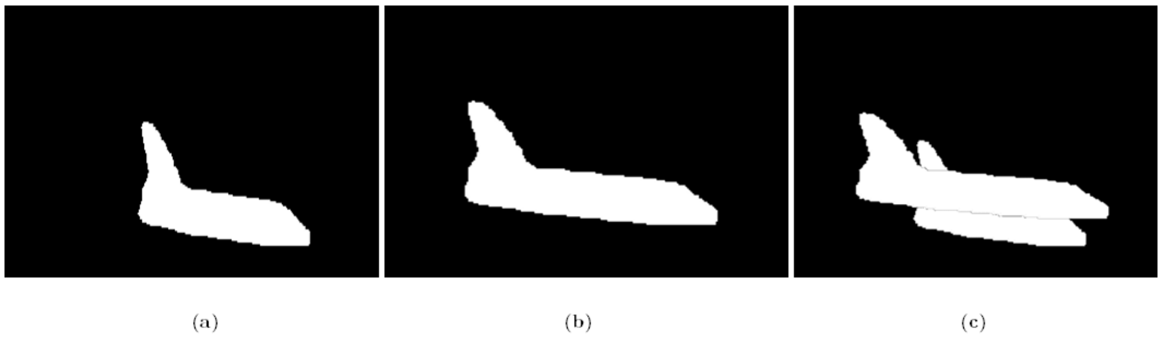


Figure 14: Shape closeness. (a) shape A (shA) in frame n, (b) shape B (shB) in frame n+1 , (c) superimposed frame (shA with shB)

Then the problem can be represented as given two shapes; how to find whether two shapes intersect with each other? This problem is challenging because this method needs to handle all sorts of shapes, sizes, and orientations. One popular algorithm for testing shape intersection is the Gilbert Johnson Keerthi (GJK) algorithm [54]. With that, it is possible to detect the intersection between any two convex polygons. There are other ways to solve this problem that takes perhaps a more straightforward approach but what's particularly satisfying about the GJK algorithm is it solves this problem the most efficient way, with many implementations used for a variety of applications in computer graphics.

All shapes can be split into two distinct classes—convex and concave shapes. Convex shapes have the property for any two points on the shape, the line between them will always be inside that shape, and concave shapes do not guarantee. Convex shapes are significantly more straightforward to work with than concave shapes. Therefore, when there are concave shapes, it is possible to split them into multiple convex shapes. Every concave shape can be broken down into convex shapes. If it can be solved with convex shapes, it is possible to solve it for all shapes. This transforms the original problem into a problem of convex shape intersection. Another less computational expensive operation is to find one point in common between these two shapes, there is an intersection, or similarly, if there are two points whose difference ends up being the origin, there is an intersection.

Minkowski sum is a method where it takes every possible point in one shape and adds it to every possible point in the other shape, and the resulting shape is referred to as Minkowski sum. Mathematically, this is treating every point on each shape as a vector from the origin and then adding every pair of vectors to get a new set of points. A few of these vector sums will define the boundary of the new shape. However, the GJK algorithm uses the Minkowski difference, which is just a Minkowski sum that negates all the points of one shape. For the GJK algorithm's purpose, there are only two properties of the Minkowski difference that need to be explored. The first is that the resultant of the Minkowski difference of two convex shapes will form a convex shape. The second key property is if the two shapes intersect, then the Minkowski difference must contain the origin.

- $sh(A)\ and\ sh(B)\ convex \rightarrow sh(A) \ominus sh(B)\ convex$
- $sh(A)\ and\ sh(B)\ intersect \rightarrow (0,0) \in sh(A) \ominus sh(B)$

As defined, calculating the full Minkowski difference is not a solid solution in terms of computational performance. A simplex is a shape containing n+1 vertices, where n is the number of dimensions. This is the most basic shape that can be used to pick a region in space. In 2D, for example, the easiest shape that can select an area containing a unique point is a triangle. Therefore, if there is a triangle made up of points on the Minkowski difference surrounding the origin, there is an intersection by definition of convexity.

For every point on the convex shape, there is a direction where it is the furthest point. A function that maps a direction vector to the point furthest on the shape is called a support function, and the corresponding point is called the support point. For Minkowski differences, all that need to do is to select a direction, find the support point in that direction for the first shape, then take the opposite direction for the second shape and subtract two points.

$$S_C(\vec{d}) = S_A(\vec{d}) - S_B(\vec{-d}) \tag{11}$$

A support function S_B takes a direction $\vec{-d}$ and returns a point v on the boundary of shape B "furthest" in direction $\vec{-d}$. Mathematically, this is defined as the point on the shape that maximizes the dot product with direction d.

$$S_B(\vec{d}) = v = \operatorname*{argmax}_{v \in sh(B)} v^T \vec{d} \tag{12}$$

To implement this algorithm, start with a random direction and find the support point in that direction. This gives the first point of in the simplex (triangle in two-dimensional space). Since the goal is to surround the origin, the next direction that makes the most sense to select is towards the origin.

Using the support function, get the second point. Now it is required to do a sanity check to verify if this point passes the origin. If the furthest point in this direction does not pass the origin, there is no way the Minkowski difference can contain the origin, and this implies there is no shape intersection. To determine, if a point passes the origin, this implementation uses vector dot product. Treating point A as a vector from the origin if the dot product with the direction vector $\vec{d}$ is less than zero implies that this point A does not pass the origin.

$$A^T \vec{d} < 0 \tag{13}$$

Now the next step is to select the next direction. The direction that GJK selects is the normal vector to the current line segment facing the origin. This can be achieved by using the vector triple product. The triple product is a nice way to find normal vectors with a specific orientation. This completes the third point. Now again, perform the sanity check. If it passes the origin, this gives the first triangle to check if this contains the origin.

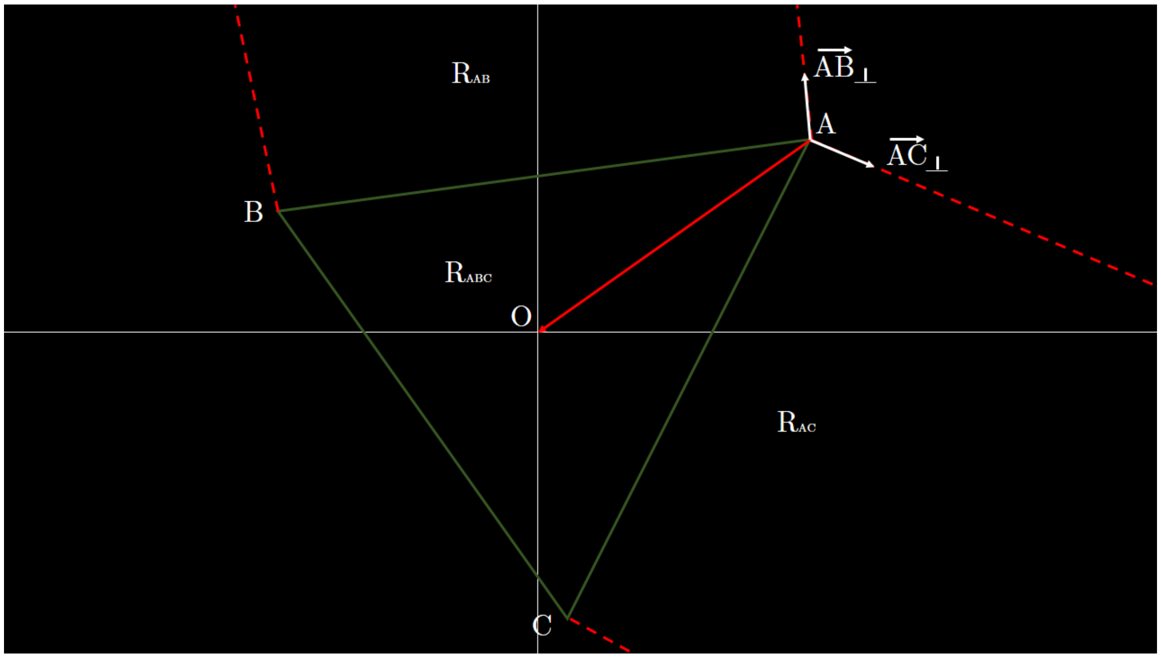


Figure 15: Evaluation of whether a triangle contains the origin. A, B, C points are in the Minkowski difference, O is the origin and R_{AB}, R_{AC}, R_{ABC} define regions as shown in the figure.

Given a triangle, adding perpendicular lines to each side defines a set of regions for space. These regions are known as Voronoi regions, and the origin will be in one of these regions (figure 15). First, check region AB with the dot product. If the origin is not there, similarly check region AC, and if the origin is not in that region, then the origin must be contained within the triangle.

$$\vec{AB}_\perp \cdot \vec{AB} < 0 \rightarrow O \notin R_{AB} \tag{14}$$

$$\vec{AC}_\perp \cdot \vec{AO} < 0 \rightarrow O \notin R_{AC} \tag{15}$$

This implies $O \in R_{ABC}$.

However, if it does not contain, this algorithm iteratively searches a triangle until it covers all the Minkowski difference points. To find the next triangle, the direction GJK selects is the perpendicular vector to the side of the triangle that is closest to the origin. From this, it is necessary to select the vector facing the origin, increasing the chances of getting a new point that will eventually enclose the origin. After that, get a new point that passes the sanity check and now update the triangle. If this triangle now contains the origin, so the two shapes must intersect. In general, this is updating directions and adding points to the simplex until this encloses the origin or determines that it is not possible.

The implementation of the complete region estimator as follows. The GJK function will take two shapes and return true if the shapes intersect. Here, the first step is to pick an initial direction. It is possible to pick a random initial direction, but a common starting direction is a vector between the two centers of the shapes, and also, as a note, all vectors will be normalized. In addition, in this context, a shape is referred to the outermost boundary of the triangular mesh (which results from the Delaunay triangulation) that forms a convex polygon shell, and the Delaunay triangulation is performed on a set of edge vertices that result from the Canny edge operation with the input shape mask for each detected shape. Moreover, the two shapes need to find the intersection extracted from two consecutive frames (figure 16). Here it is assumed that depending on the application, the video sequence's frame rate is comparatively high, the origin of the 2D video frame does not change throughout the video sequence, and the relative viewpoint between two consecutive frames is fixed. Finally, depending on the GJK computation results, these shape masks are then assigned to shape region clusters. In the next sections, these shape region clusters that contain shapes will further compare to each other in terms of shape structure, spatial color, and build/maintain the data continuity model for this shape tracking.

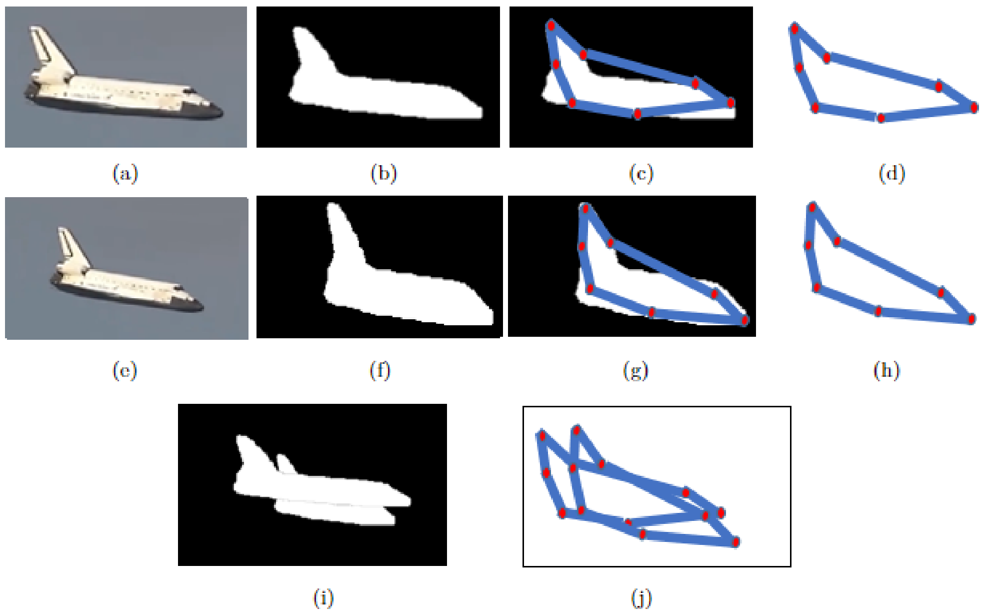

(a) (b) (c) (d) (e) (f) (g) (h) (i) (j)

Figure 16: Overall region estimation process. (a) input frame n, (b) shape masks for frame n $sh(A)$, (c,g) Outermost boundary of the triangular mesh, (d) $sh(A)$ skeleton, (e) input frame $n+1$, (f) shape masks for frame $n+1$ $sh(B)$, (h) $sh(B)$ skeleton, (i) superimposed $sh(A)$ mask with $sh(B)$ mask to a common reference frame, (j) superimposed $sh(A)$ skeleton with $sh(B)$ skeleton to a common reference frame.

Algorithm 2 introduces the full region estimation method used in this research. In Algorithm 2, it is assumed that all the detected shapes in both consecutive frames are assigned to numbers for shape addressing purposes.

Algorithm 2: 2D region estimator

```
Input: $NOS1$ = Number of shapes in the first frame, $NOS2$ = Number of
        shapes in the second frame, $O$ = Reference frame origin point.
Output: Assign shapes to group based on their relative location.
        (groupShapeCluster)
 Function supportPoint($shA$,$shB$,$d$):
    return ($shA$.furthestPoint($d$) − $shB$.furthestPoint($-d$));

Function checkSimplexIntersection(simplex,$d$):
    switch length(simplex) do
        case 2 do
            return lineCase(simplex,d);
        end
        case 3 do
            return triangleCase(simplex,d);
        end
    end

Function lineCase(simplex,$d$,$O$):
    B,A = simplex;
    AB,AO = B-A , O-A;
    ABPerpendicular = tripleProduct(AB,AO,AB);
    d.set(ABPerpendicular);
    return false;

Function triangleCase(simplex,$d$,$O$):
    C,B,A = simplex;
    AB,AC,AO = B-A, C-A, O-A;
    ABPerpendicular = tripleProduct(AC,AB,AB);
    ACPerpendicular = tripleProduct(AB,AC,AC);
    if dotProduct(ABPerpendicular,AO) $>$ 0 then
        Simplex.remove(C);
        d.set(ABPerpendicular);
        return false;
    end
    else if dotProduct(ACPerpendicular,AO) $>$ 0 then
        Simplex.remove(B);
        d.set(ACPerpendicular);
        return false;
    end
    return true;

```

```
Function GJK($shA$, $shB$,$O$):
    D = normalize(shA.centroidPosition – shB.centroidPosition);
    Simplex = [supportPoint(shA,shB,d)];
    D = O – simplex[0];
    while True do
        A = supportPoint(shA,shB,d);
        if dotProduct(A,d) < 0 then
            return false;
        end
        Simplex.append(A);
        if checkSimplexIntersection(simplex,d) then
            return true;
        end
    end

Function Main($NOS1$,$NOS2$,$O$):
    if $NOS1 > NOS2$ then
        while $NOS1 > 0$ do
            shA = getShapeFromFirstFrameWithID(NOS1) ;
            while $NOS2 > 0$ do
                shB = getShapeFromSecondFrameWithID(NOS2) ;
                bool intersect = GJK(shA,shB,O) ;
                if intersect then
                    groupShapeClusters(shA,shB) ;
                end
                $NOS2 = NOS2 - 1$;
            end
            $NOS1 = NOS1 - 1$;
        end
    end
    else
        while $NOS2 > 0$ do
            shB = getShapeFromSecondFrameWithID(NOS2) ;
            while $NOS1 > 0$ do
                shA = getShapeFromFirstFrameWithID(NOS1) ;
                bool intersect = GJK(shA,shB,O) ;
                if intersect then
                    groupShapeClusters(shA,shB) ;
                end
                $NOS1 = NOS1 - 1$;
            end
            $NOS2 = NOS2 - 1$;
        end
    end
```

4 Shape Context Descriptor and Spatial Color Analyzer

4.1 Introduction

This section explores the efficient shape matching technique based on shape key points and the shape color variation. For this, we explore concepts of pixels, distance metrics, image histogram, keypoint descriptors, affine transformation, color spaces, Voronoi regions, and clustering algorithms.

In this chapter, Section 4.2 briefly outlines the fundamental shape analysis techniques. In contrast, Section 4.3 provides the structure and working principle of shape context descriptor, and finally, Section 4.4 illustrates the spatial color analyzer module.

4.2 Preliminaries

4.2.1 Pixels

In a digital image, a pixel may contains a value(s) that determines the color or brightness of that pixel (Figure 17).

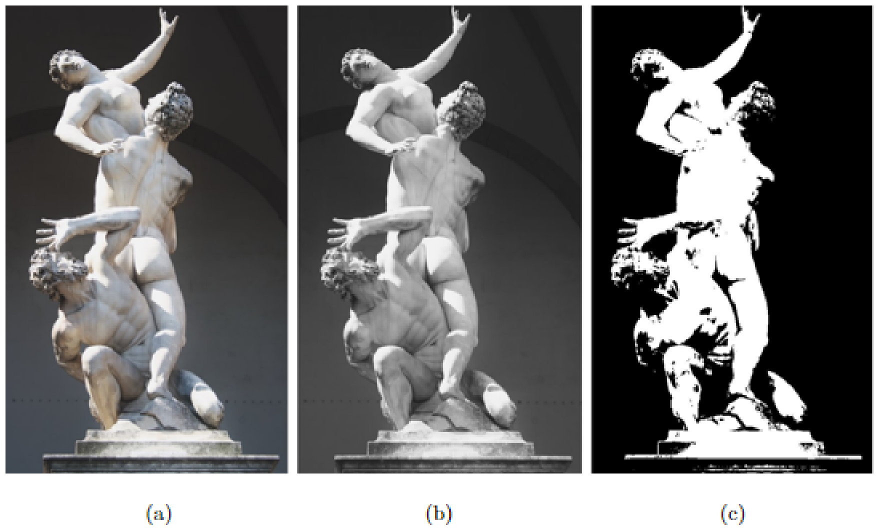
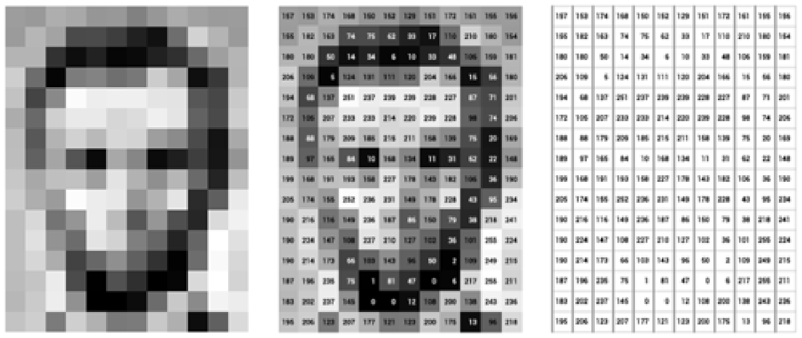


Figure 17: Pixel value representation. (a) RGB color image [5], (b) grayscale version, (c) binary version

The pixel value in a grayscale image reflects its pixel brightness. The most common pixel format is the byte image; here, each pixel value is stored as an 8-bit integer with a value range from 0 to 255. In general, 0 is assigned to black, and 255 is white. The remaining values generate the variations of gray (Figure 18).



157	153	174	168	150	152	129	151	172	161	155	156
155	182	163	74	75	62	33	17	110	210	180	154
180	180	50	14	34	6	10	33	48	106	159	181
206	109	5	124	131	111	120	204	166	15	56	180
194	68	137	251	237	239	239	228	227	87	71	201
172	105	207	233	233	214	220	239	228	98	74	206
188	88	179	209	185	215	211	158	139	75	20	169
189	97	165	84	10	168	134	11	31	62	22	148
199	168	191	193	158	227	178	143	182	106	36	190
205	174	155	252	236	231	149	178	228	43	95	234
190	216	116	149	236	187	86	150	79	38	218	241
190	224	147	108	227	210	127	102	36	101	255	224
190	214	173	66	103	143	96	50	2	109	249	215
187	196	235	75	1	81	47	0	6	217	255	211
183	202	237	145	0	0	12	108	200	138	243	236
195	206	123	207	177	121	123	200	175	13	96	218

(a) (b) (c)

Figure 18: 8-bit grayscale image representation. (a) low resolution input image [6], (b) grayscale pixel values overlapped on the input image, (c) grayscale value matrix representation.

Separate Red, Green, and Blue components are maintained for each pixel in RGB color space (Figure 19). In this color space, an individual pixel value can be represented in a three-number vector. When displaying or processing the image, the three RGB components have to be recombined. Apart from the RGB images, multi-spectral images have more than three components per pixel that need to be stored similarly. It is also likely that each pixel's grayscale or color component intensities are not stored explicitly. It is possible to store them as an index into a colormap which then can be used to extract intensity or colors.

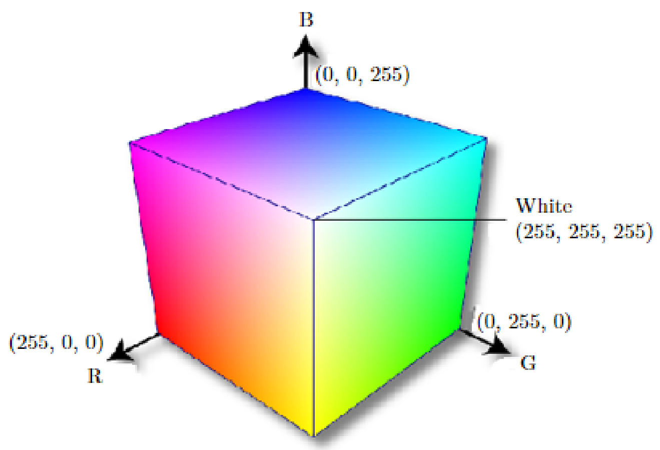


Figure 19: 8-bit RGB color space [7]

4.2.2 Distance Metrics

There are four primary distance metrics in digital image processing: Euclidean, City Block, Chessboard, and Quasi-Euclidean [55]. These metrics are used to estimate the distance of points in an image.

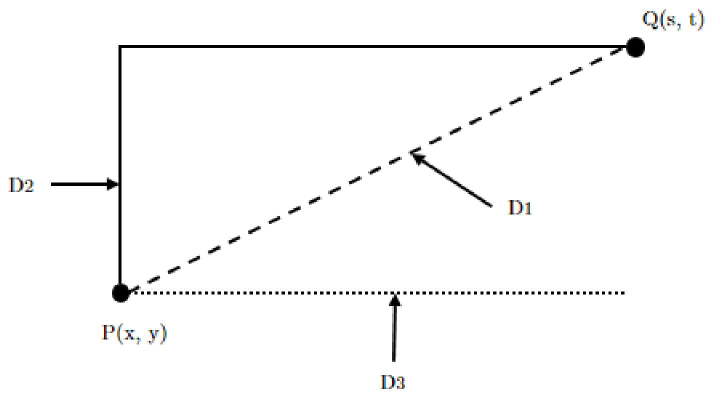


Figure 20: Distance metrices

Figure 20 illustrates the differences between these metrics. All notations are shown according to the figure 20 and $D(p,q) \geq 0$ *and* $D(p,q) = D(q,p)$, hence,

- Straight line distance between two pixels is referred to as the Euclidean distance.

- The city block distance metric is based on a four-connected neighbourhood and calculates the path between those pixels.
- The chessboard distance metric uses an eight connected neighbourhood instead of four to measure the path between pixels.
- The total Euclidean distance along a set of horizontal, vertical, and diagonal line segments is used in the Quasi-Euclidean metric. Since this is a path-based metric, one use case of this is to retrieve the shortest path between two vertices (or pixels in image processing context) on a two dimensional plane.

$$Euclidean\ D_1(p,q) = \sqrt{(x-s)^2 + (y-t)^2} \tag{16}$$

$$City\ block\ D_2(p,q) = |x-s| + |y-t| \tag{17}$$

$$Chess\ board\ D_3(p,q) = max(|x-s|, |y-t|) \tag{18}$$

$$Quasi\ Euclidean\ D_{(p,q)} = \left\{ \begin{array}{ll} |x-s| + (\sqrt{2}-1)|y-t| & ; |x-s| > |y-t| \\ (\sqrt{2}-1)|x-s| + |y-t| & ; otherwise \end{array} \right\} \tag{19}$$

4.2.3 Image Histogram

In image processing, an image's histogram typically refers to a histogram of pixel intensity values. This histogram displays the number of pixels in an image at each of the image's different intensity values. There are 256 different intensities for an 8-bit grayscale image. Therefore, a histogram can be used to represent these different possible intensity values graphically. Color images can also be represented as histograms, either as discrete red, green, and blue channel histograms or as a 3-D histogram. The red, blue, and green channels' brightness is represented by the three axes, with each point representing the pixel count.

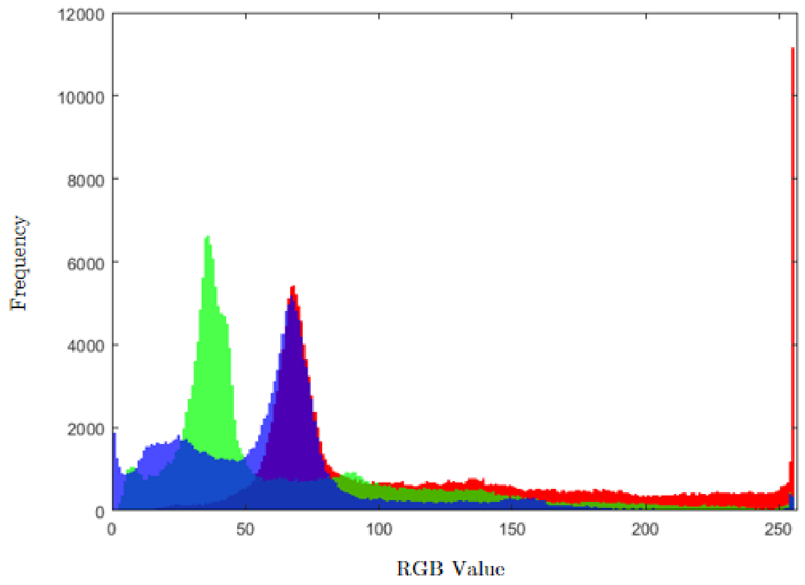


Figure 21: Color histogram in RGB color space. 8-bit per channel.

In digital image processing, histograms are used to identify and modify the following properties. [56].

- Dynamic Range.
- Brightness.
- Contrast.
- Under and over exposure.

In addition to that, a spatially variant image representation in which pixel separation increases linearly with distance from a central point is known as log-polar sampling [57]. This helps to focus on particular areas of interest while holding low-resolution data from a broader perspective. This type of foveal image representation is most effective in active vision systems, where the densely sampled central area can be focused on collecting the most salient information. In general, as shown in Figure 22, conformal projection from points in the cartesian plane (x,y) to points in the log-polar plane is the log-polar transformation (ξ, η):

Where,

$$\xi = \log \sqrt{x^2 + y^2} \tag{20}$$

$$\eta = a \tan(\frac{y}{x}) \tag{21}$$

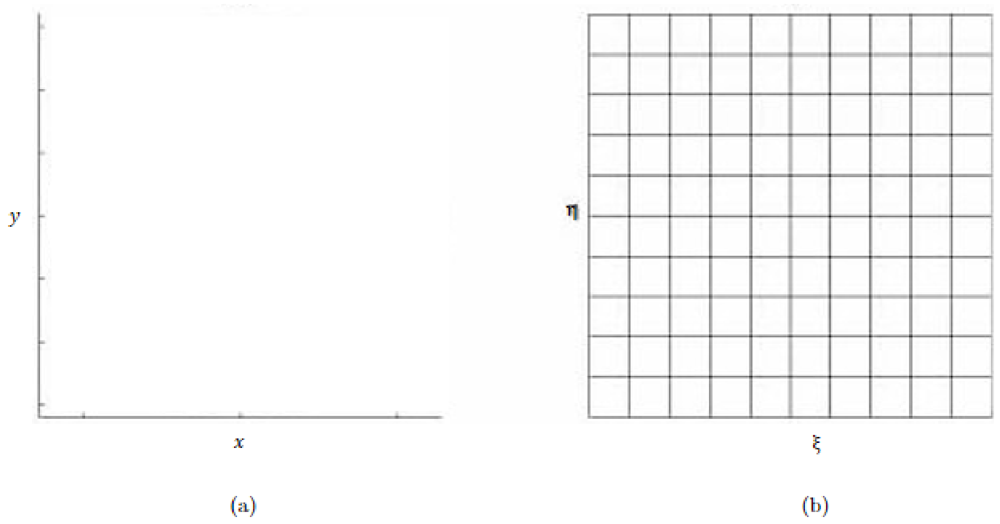


Figure 22: Log Polar histogram [8]. (a) cartesian plane, (b) log-polar plane

4.2.4 Keypoint Descriptors

Keypoint descriptors are primarily used to detect interest-points (feature-points or key-points [58]) in an image. In the context of digital image processing, primary features can be described as corners, edges, and blobs. Based on the unique patterns of their adjacent pixels, the observed features are then interpreted in logically differing ways. This approach is known as feature description since it assigns a specific identity to each feature, allowing for efficient matching [58]. Most feature detectors have their own system for explaining features. In most cases, feature detectors may have a combination of different relevant feature descriptors. Scale Invariant Feature Transformation (SIFT) [59], Speeded Up Robust Features (SURF) [60], and Oriented

FAST and Rotated BRIEF (ORB) [61] have their own feature-descriptors and use scale, rotation and affine invariant feature detectors. Figure 23 illustrates the comparison between SIFT, SURF, BRIEF and ORB.

In terms of distance metrics for feature matching, SIFT, SURF (string-based descriptors) use L1 and L2 norm while ORB (binary descriptors) uses Hamming distance. Threshold-based matching, nearest neighbor, and nearest neighbor distance ratio matching are the primarily used methods when matching features [62]. However, incorrect matches (or outliers) cannot be completely eliminated during the feature-matching period.

4.2.4.1 SIFT

An approximation of Laplacian of Gaussian operator, which is the Difference of Gaussian, is used in SIFT detector. Feature points are calculated using local maxima of Difference of Gaussian at different scales of the input image. Typically, this can be achieved by extracting a 16x16 pixel patch around each detected feature and further segmenting the region into sub-regions. This method is invariant to image rotation, small affine variations, and scale-invariant; nevertheless, the higher computational cost is the key drawback. Equation 22 shows the convolution of difference of two Gaussians (computed at different scales) with image "I(x,y)".

$$D(x, y, \sigma) = (G(x, y, k\sigma) - G(x, y, \sigma)) * I(x, y) \tag{22}$$

In this equation, "G" represents the Gaussian function.

4.2.4.2 SURF

In this method, to increase the feature detection speed, the integral images are used, and SURF is primarily based on the determinant of Hessian [?] matrix. Similar to SIFT, SURF features are also invariant to rotation and scale. ALow computational cost of SURF compared to SIFT is the main advantage of this method. Equation 23 represents the Hessian Matrix in point “x = (x,y)” at scale σ.

$$H(x,\sigma) = \begin{bmatrix} L_{xx}(x,\sigma) & L_{xy}(x,\sigma) \\ L_{xy}(x,\sigma) & L_{yy}(x,\sigma) \end{bmatrix} \tag{23}$$

Where $L_{xx}(x,\sigma)$ is the convolution of Gaussian second order derivative with the image “I” in point “x”, and similarly for “$L_{xy}(x,\sigma)$” and “$L_{yy}(x,\sigma)$”.

4.2.4.3 ORB

ORB algorithm is a combination of modified FAST (Features from Accelerated Segment Test) [63] detection and direction-normalized BRIEF (Binary Robust Independent Elementary Features) [64] description methods. FAST detection is used to detect image corners for each layer in the image pyramid, and then Harris corner detection algorithm is used to evaluate the best corners for each layer. An updated version of the BRIEF descriptor was used since the BRIEF is volatile with rotation [65]. ORB features are invariant to scale, rotation, and slight affine changes.

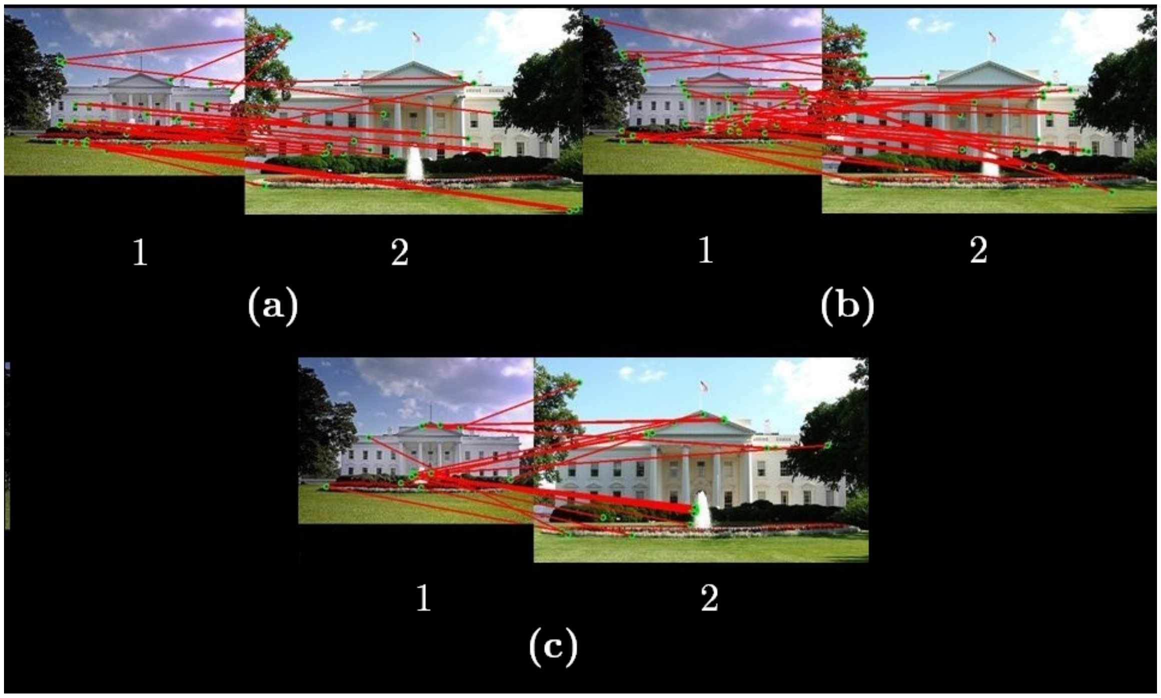


Figure 23: Keypoint matching comparison. (a) SIFT keypoint matching, (b) SURF keypoint matching, (c) ORB keypoint matching. In each method, image (1) and image(2) represent two images of the same object but from different viewpoints. Each red line indicates the matching connection between the different viewpoints.

4.2.5 Affine Transformation

4.2.5.1 Collinear

If three or more points X1, X2, X3,..., lie on a single straight line L, they are considered to be collinear [66] . Since two points define a line, they are trivially collinear [66]. Three points $X_i = (x_i, y_i, z_i)$ for $i = 1, 2, 3$ are collinear iif the ratios of distances satisfy [66],

$$x_2 - x_1 : y_2 - y_1 : z_2 - z_1 = x_3 - x_1 : y_3 - y_1 : z_3 - z_1 \tag{24}$$

Furthermore, the area of a triangle defined by three points would be zero if they are collinear [67] , [68], [69], e.g.,

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ x_3 & y_3 & 1 \end{bmatrix} \tag{25}$$

or, in expanded form,

$$x_1(y_2 - y_3) + x_2(y_3 - y_1) + x_3(y_1 - y_2) = 0 \tag{26}$$

This can also be expressed as a vector:

$$Tr(X \times Y) = 0 \tag{27}$$

where Tr(A) is the sum of components, $X = (x_1, x_2, x_3)$ and $Y = (y_1, y_2, y_3)$

4.2.5.2 Affine Transformation

Any transformation that preserves collinearity (‘All points lying on a line initially still lie on a line after transformation’ [70]) and ratios of distances (e.g., after transformation, the midpoint of a line segment remains as the midpoint. [70]) is called an affine transformation [71] [72] [73]. The phrase affine refers to a type of projective transformation in which no objects are moved from affine space $\mathbb{R}^3$ to the plane at infinity or vice versa.

Affinity is another name for an affine transformation [70]. Translation, dilation, similarity transformations, reflection, expansion, rotation, shear, spiral similarities, and geometric contraction are all affine transformations, as are their combinations [70]. An affine transformation is made up of translations, rotations, shears, and dilations in general [70]. While an affine transformation conserves proportions on lines, angles and lengths are not always preserved [70]. The rotation-enlargement transformation is an example of combining rotation and expansion.

$$\begin{bmatrix} x' \\ y' \end{bmatrix} = s \begin{bmatrix} \cos\alpha & \sin\alpha \\ -\sin\alpha & \cos\alpha \end{bmatrix} \begin{bmatrix} x - x_0 \\ y - y_0 \end{bmatrix} \tag{28}$$

$$= s \begin{bmatrix} \cos\alpha(x - x_0) + \sin\alpha(y - y_0)) \\ -\sin\alpha(x - x_0) + \cos\alpha(y - y_0)) \end{bmatrix} \tag{29}$$

separating the equations,

$$x' = (s\cos\alpha)x + (s\sin\alpha)y - s(x_0\cos\alpha + y_0\sin\alpha) \tag{30}$$

$$y' = (-s\sin\alpha)x + (s\cos\alpha)y + s(x_0\sin\alpha - y_0\cos\alpha) \tag{31}$$

This can also be written:

$$x' = ax - by + c \tag{32}$$

$$y' = bx + ay + d, \tag{33}$$

where,

$$a = s\cos\alpha \tag{34}$$

and

$$b = -s\sin\alpha \tag{35}$$

The scale factor s is then defined by :

$$s = \sqrt{a^2 + b^2} \tag{36}$$

and the rotation angle by:

$$\alpha = \tanh^{-1}(-\frac{b}{a}) \tag{37}$$

4.2.6 Color Spaces

Color spaces are various color modes used for various uses of computer vision and signals and systems applications. The following are some of the most common color spaces:

- RGB - Red, Green, Blue
- CMYK - Cyan, Magenta, Yellow, Black
- YCbCr - Y is the luma component and CB and CR are the blue-difference and red-difference chroma components [74].
- HSV -Hue, Saturation, Value,

The most commonly used color space is RGB. According to the RGB model, each color image is made up of three separate images. A grayscale image is characterized by a single matrix, while a color image is defined by three matrices. R (red), G (green), and B (blue) values on each axis of a three-dimensional coordinate plane can be used to represent the possibilities for combining the three primary colors. This three-dimensional plane illustrates the RGB color space. In this color space, if all the color vector is at position (0,0,0) the resulting color is black while if the position is (255,255,255 - for 8-bit color depth per channel) the color is white. All the other color values are within these limits. This type of color mixing is also called “additive color mixing”:

4.2.7 Voronoi Regions

The separation of a plane with n points into convex polygons, with each polygon containing exactly one generating point and each point in a given polygon being closer to its generating point than any other [1].

Definition: Let,

- Set S of point sites.
- Distance function: d(p,s) = Euclidean distance

partition of space into regions $VR(s)$ such that for all p in $VR(s), d(p, s) < d(p, t)$ for all $t \neq s$. The naming conventions can be different depending on the dimension as described in this Table 2.

Table 2: Naming convention based on the dimensionality [1]

	2-Dimensional	3-Dimensional
Voronoi polygons	Voronoi regions	polytopes (convex polyhedron)
Voronoi edges	equidistant to 2 sites	intersections of faces
Voronoi vertices	equidistant to 3 sites	
Voronoi faces		bisectors, bound polytopes

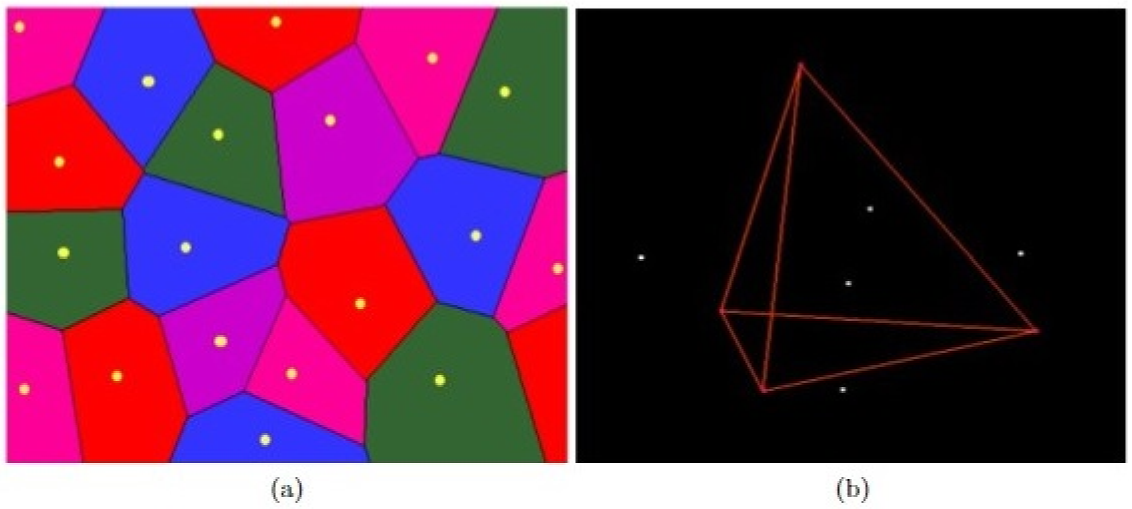


Figure 24: Voronoi regions. (a) 2d Voronoi region (b) tetrahedron represents a 3d Voronoi region [9]

4.2.8 Clustering Algorithms

Clustering is a technique for grouping data points that are similar. A clustering algorithm can be used to assign each data point to a specific category. After clustering is performed, each group should contain data points that have identical features or properties. Clustering is an unsupervised learning approach that is widely used in many areas for statistical data analysis.

$k - means$ is a recursive algorithm that tries to group given data points into non-overlapping clusters, and each data point is guaranteed to assign to only one group. This method tries to reduce the sum of squared distance between the points of the cluster centroid at minimum. The $k - means$ clustering is described in Algorithm 3 :

```
Algorithm 3: $k - means$ Clustering
1 Initialize cluster centroids $\mu_1, \mu_2, ..., \mu_k \in \mathbb{R}^n$ randomly.;
2 while until convergence do
3     For every $i$,set $c^{(i)} := arg\,\underset{j}{min} \left\| x^{(i)} - \mu_j \right\|^2$;
4     For each $j$, set $\mu_j := \frac{\sum_{i=1}^{m} 1(c^{(i)}=j)x^{(i)}}{\sum_{i=1}^{m} 1(c^{(i)}=j)}$;
5 end
```

The approach $k{-}means$ follows to solve the problem is called Expectation-Maximization. The E-step is assigning the data points to the nearest cluster. The M-step is computing the centroid of each cluster. However, the following can be identified as the disadvantages of this method.

To start, it is necessary to determine how many groups/classes there will be. Since $k - means$ does not have a method to calculate that from the input data, it has to be given as an input parameter to the algorithm. Often domain awareness and intuition come in handy, but that is not always the case to determine the number of clusters. One of the popular approaches to use an Elbow method for this problem. The Elbow approach suggests a reasonable k number of clusters. At the point where the sum of squared distance begins to flatten out and form an elbow, that is the reasonable k to pick when deciding the number of clusters. To illustrate this approach, using the geyser dataset [75], we will measure the sum of squared distance for various k values to see where the curve could form an elbow and flatten out.

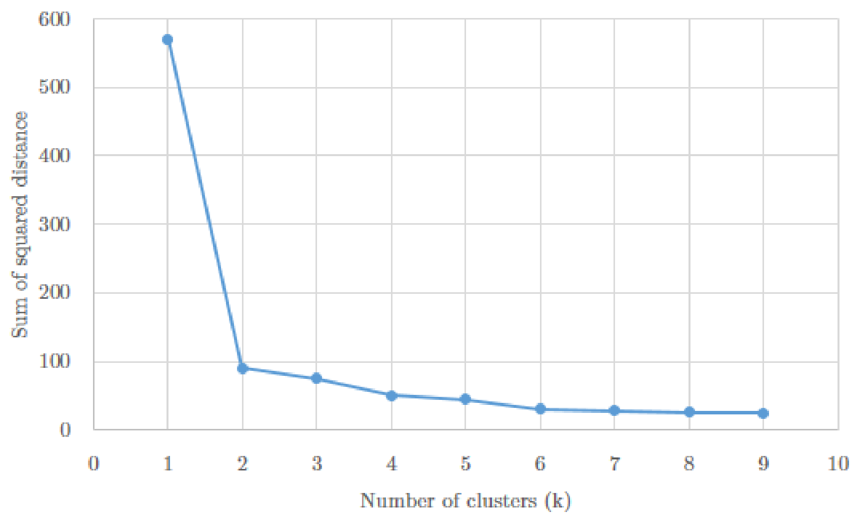


Figure 25: Results obtained from elbow method

Figure 25 demonstrates for this particular data set, $k = 2$ is a reasonable number of clusters. However, the initialization of the centroids points affects this algorithm. As a result, If an initial centroid point is set a greater distance to the data points or a couple of centroid points initialized too close to each other, the clustering would be weak. To overcome this issue, we have used the $k - means++$ algorithm. This addition to the existing $k-means$ algorithm ensures a more logical initialization of the centroids and increases the clustering accuracy. The rest of the algorithm is identical to the existing $k-means$ algorithm, except the initialization section. $k-means++$ combines the standard $k - means$ algorithm with a smarter centroids initialization. Following this initialization protocol, it is possible to initialize centroids that are far apart. This raises the probability of getting centroids in various clusters initially.

Step(1) From the given data points, randomly adopt points as for the initial centroid points.

Step(2) Calculate the distance between each data point and the closest, previously selected centroid.

Step(3) Select the next centroid from the data points such that the probability of choosing a point as centroid is directly proportional to its distance from the nearest, previously chosen centroid (i.e., the point having maximum distance from the nearest centroid is most likely to be selected next as a centroid)

Step(4) Repeat steps 2 and 3 until k centroids have been sampled.

In addition, Different runs of the algorithm will yield different clustering results since $k - means + +$ starts with a random collection of cluster centers. As a result, the findings could not be repeatable or consistent.

In conclusion, when compared to all other major clustering algorithms such as expectation-maximization (EM) clustering using Gaussian mixture models (GMM), mean-shift clustering, Agglomerative hierarchical clustering, and density-based spatial clustering of applications with noise [76], $k - means + +$ has the benefit of being fairly fast [76]. Even though $k - means + +$ algorithm has several drawbacks when compared to other clustering methods, in this research work, this is the selected algorithm for the spatial color analyzer module.

4.3 Shape Context Descriptor

The shape context descriptor compares shapes between two consecutive frames and produces a similarity value that can be used when assigning shapes to tracks. If the measurement value (similarity cost value) is zero or minimum, that can be interpreted as the compared two shapes are identical or similar in terms of shape outline (contour). The algorithm to obtain shape context descriptor [77] between two consecutive frames are given in the Algorithm 4.

```
Algorithm 4: Shape context descriptor
  Input: Shape A in first frame and Shape B in next the frame
  Output: Shape similarity cost value.
1 Let sh(A), sh(B) are shapes that are extracted from two consecutive frames
    in a video and their binary masks are available. Let number of vertices in
    sh(A) = sh(B) = $n$;
2 In Sh(A), start with the 1st vertex. Compute the distance and angle between
    the 1st vertex and the rest of $n-1$ vertices;
3 Normalized the distance based on the median of the distribution.;
4 Construct a log-polar histogram for the 1st vertex of sh(A) based on the
    distance and angle;
5 Similarly, create log—polar histogram for the rest of the $n-1$ vertices in
    Sh(A) and also repeat the process for the $n$ vertices in sh(B);
6 With both $n$ log-polar histograms generated for sh(A) and sh(B), compute
    the cost (eqn.38) between the two sets of log-polar histograms (Here, K =
    Number of bins in the histogram and k = specific position of data bin
    within the histogram);
7 Find the matched pair of vertices between sh(A) and sh(B) whose having the
    minimum cost value;
8 Select at least 3 vertices from sh(B) that are not collinear with each other;
9 Perform affine transformation for sh(B) with the above selected vertices and
    project two shapes onto the same coordinate system;
10 Compute the nearest vertices distances between projected sh(B) and sh(A).
11 Similarity cost value = sum of the nearest distances.
```

The following example demonstrates the above algorithm in action. Firstly, it is necessary to identify the vertices in shape. The vertices are usually the intersection points of the edges. Let sh(A) = sh(B) = 24 vertices and their named as sh(A) vertices = A1,A2,...,A24 and similarly, sh(B) vertices = B1,B2,...B24 (figure 26). Start with the first vertex (A1), compute the euclidean distance and angle to the second vertex(A2), and then repeat the process with the rest of the vertices in sh(A). Likewise, compute the euclidean distance and angle for the sh(B) as well and construct the table 3 and 4.

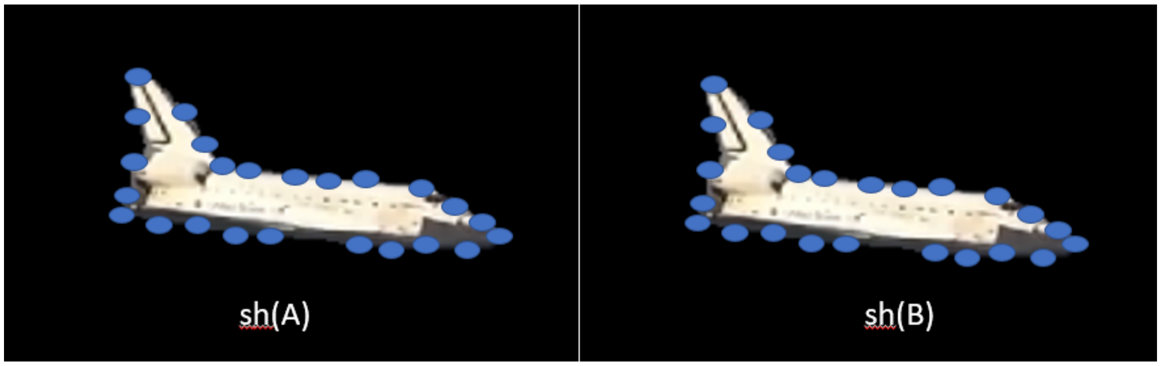


Figure 26: Vertices from Corner detection

Table 3: Shape A (sh(A)) distances and angles

Vertices	Distance	Angle
A1-A2	48	302
A1-A3	82	302
A1-A4	114	322
...	...	...
A1-A24	115	333

Table 4: Shape B (sh(B)) distances and angles

Vertices	Distance	Angle
B1-B2	283	270
B1-B3	250	265
B1-B4	219	258
...	...	...
B1-B24	211	237

Since the two shapes may be in different scales, we usually normalize the distance based on the median value ,and therefore above tables becomes,

Table 5: Shape A (sh(A)) normalized distances and angles

Vertices	Distance	Angle
A1-A2	0.58	302
A1-A3	1.00	302
A1-A4	1.39	322
...	...	...
A1-A24	0.91	333

Table 6: Shape B (sh(B)) normalized distances and angles

Vertices	Distance	Angle
B1-B2	1.67	270
B1-B3	1.47	265
B1-B4	1.29	258
...	...	...
B1-B24	1.24	237

We need to transform the above-normalized tables into log-polar histograms. Let the bin size of a histogram is 12 x 5. Then, the first log-polar histogram for the vertice A1 can be constructed as follows:

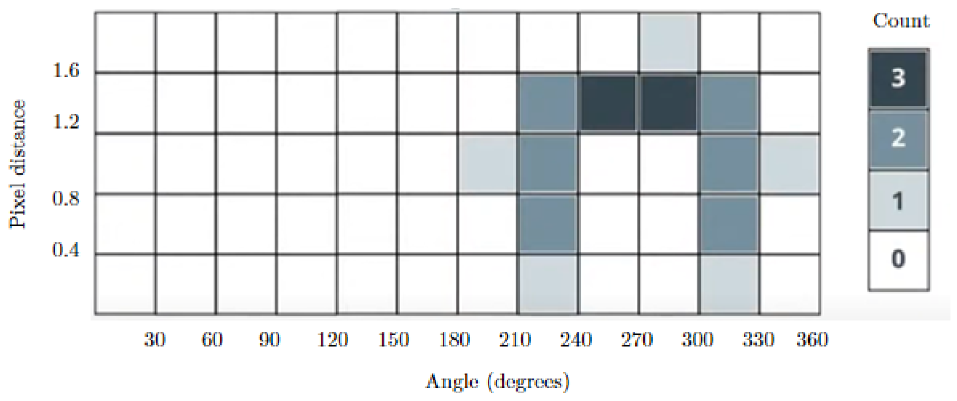


Figure 27: Log-polar histogram for vertice A1

Likewise, we generate log-polar histograms for the remaining vertices in sh(A) and sh(B). According to this example, in total, there are 48 log-polar histograms for sh(A) and sh(B).

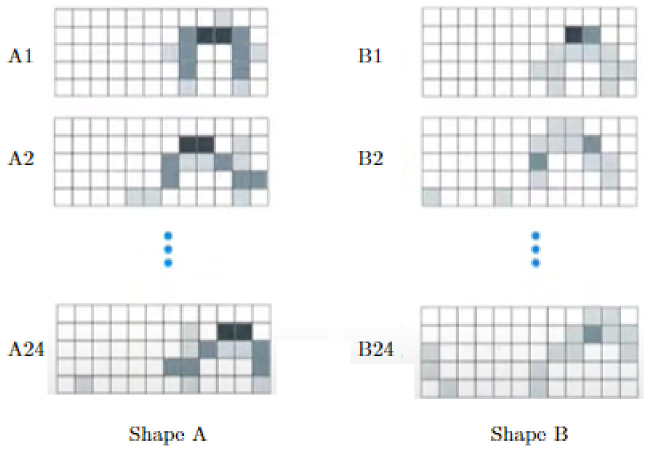


Figure 28: Total Log-polar histogram for sh(A) and sh(B)

Compute the cost matrix of A1 with respect to B1 to B24.

Cost matrix equation:

$$C = \frac{1}{2} \sum_{k=1}^{K} \frac{(A(k) - B(k))^2}{A(k) + B(k)} \tag{38}$$

Here, K is the total number of data bins (12 * 5 = 60) in the histogram, k refers to the data bin's specific position within the histogram. Finally, A and B refer to the log-polar histogram of Shapes A and B.

The next step is to pair up the A1 in sh(A) with any vertice in sh(B) so that C is minimum among all of the combinations. Let A1 and B2 pair have the minimum C value among all combinations; therefore, we can pair up A1 and B2. Similarly, pair up all the other vertices in sh(A) to vertices in sh(B). Note:- All vertices in sh(A) should be matched with a corresponding vertex in sh(B)

Assuming we successfully pair up the vertices, we perform an affine transformation based on the paired up vertices. After that, sh(A) and sh(B) are projected into the same coordinate system. We compute the nearest vertices euclidean distances between the two shapes. The sum of the nearest distance should be zero or very small if the two shapes are identical or similar.

Another improved version for this shape context descriptor is that extracting feature vertices from SIFT algorithm. Throughout this research, instead of selecting vertices from corner detection, we have used SIFT to detect distinct feature connections between two shapes that need to compare (see Figure 29). Since SIFT is invariance to image scale and rotation, it will eliminate the issues that can arise from shape evolution with respect to time. SIFT algorithm extract key points and compute their descriptors. The main four steps involved in the SIFT algorithm are (a) Keypoint localization (b) Scale-space extrema detection (c) Orientation assignment (d) Keypoint matching, and (e) Keypoint descriptor [78].

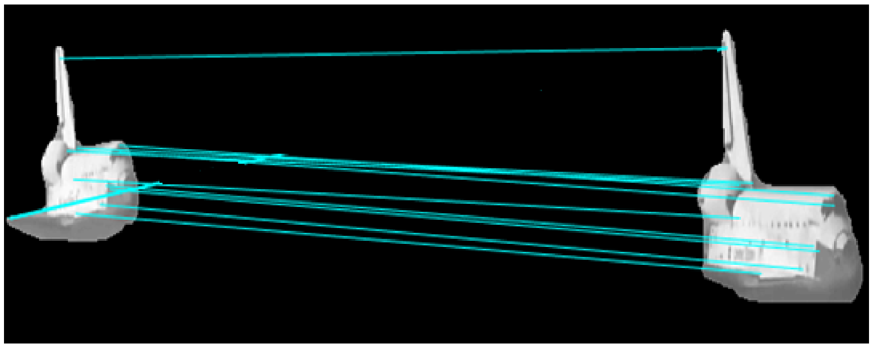


Figure 29: SIFT based keypoint detection

The improved algorithm can be given in Algorithm 5.

```
Algorithm 5: Shape context descriptor (modified)
  Input: Shape A (sh(A)) in frame $n$ and Shape B (sh(B)) in frame $n+1$
  Output: Normalized sum of nearest distance.
1 Apply SIFT keypoint descriptor to sh(A) and sh(B).;
2 Perform an affine transformation for all the vertices in sh(B) (translation
    along x and y axis);
3 Compute the nearest Euclidean vertices distance between the two shapes. ;
4 Normalize the sum of nearest distances for each region cluster.;
```

4.4 Spatial Color Analyzer

The main objective of the spatial color module is to compute and compare shape color variation between two consecutive frames. A typical RGB image can consist of approximately 16.7 million color variations if the bit depth per channel is 8 bit. One way to represent this is as a volume-defined region where we divide each axis into equally spaced M divisions. It will generate M^3 possible boxes like cubes in total, and each cube represents a color. So any pixel in an RGB image can be represented using a vector in RGB color space. The problem with this type of representation is that there are millions of possible color combinations that need to be taken into account when constructing the feature vector. Secondly, many of these cubes will be empty in most practical cases, so there are many wasted features. To overcome this issue, our algorithm places K bins (centroids) using $k - means + +$ clustering. This will allow it to go and find K different bin centers per image. So essentially, these center points will form 3D Voronoi K such regions. This way, it is always guaranteed to have at least some colors in the bins resulting in a fully utilized feature vector. So, in the end, the color space for that image can be represented as a K dimensional vector. Hence it can be mapped back to the original image pixel by pixel resulting in a 2D one-channel matrix whose pixel values are ranging from 1 to K. This is essentially a color variation reduction from 16.7 million (assuming 8bit per channel) to K. Then, it is necessary to bring the two shapes in the first frame and the next frame into the same coordinate system. This can be achieved by performing an affine transformation to the second shape.

Finally, the resultant matrix can be calculated as the absolute elementwise difference between the clustered pixel value matrix for sh(A) and sh(B). Any non-zero pixel values in the resultant matrix represent a significant color change between the compared shapes. Therefore, if the count is zero or low value in the resultant matrix, it is safe to assume that shapes are identical or similar in terms of color variation. The algorithm to obtain spatial color variation is given in Algorithm 6.

```
Algorithm 6: Spatial color analyzer
  Input: Affined transformed shapes from shape context
         descriptor.(sh(A),sh(B))
  Output: Normalized non-zero pixel count.
1 Load RGB sh(A) (WxHx3 containing N pixels).;
2 Load RGB sh(B) (WxHx3 containing N pixels).;
3 Plot all RGB color vectors in sh(A) and sh(B).;
4 Run $k - means++$ clustering to find $K$ centroids.;
5 Reconstruct sh(A) and sh(B) colors using only these $K$ RGB color vectors.;
6 Substract sh(A) new RGB color matrix with sh(B) new RGB color matrix.;
7 Count color vectors that have non-zero magnitude. If the two shapes are
    somehow identical in spatial color variation, the pixel count value should be
    zero or minimal.;
8 Normalized the non-zero pixel count for each region cluster.
```

To demonstrate this, let's take two consecutive frames extracted from a color video sequence. For this example, the selected video sequence is in full HD (1080p) and has 8 bit per channel color depth. Hence, each frame has a resolution of 1920x1080x3. Here, the red, green, and blue pixel values for each pixel position will be defined by three 8-bit integers. The first step is to reduce the number of colors to K (let K=10) and recreate the frame using only those ten colors. Applying a $k-means++$ algorithm on the image shows that it is possible to represent the same image with only K colors. Furthermore, this procedure will significantly decrease the video frame's data size.

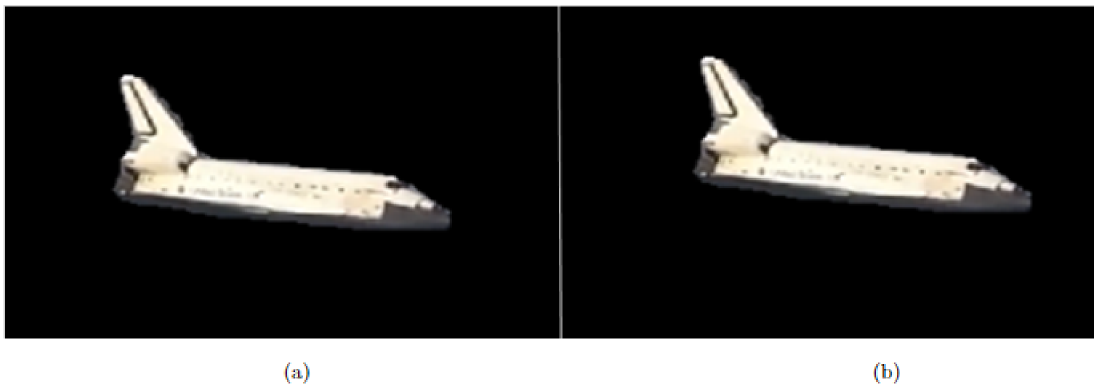
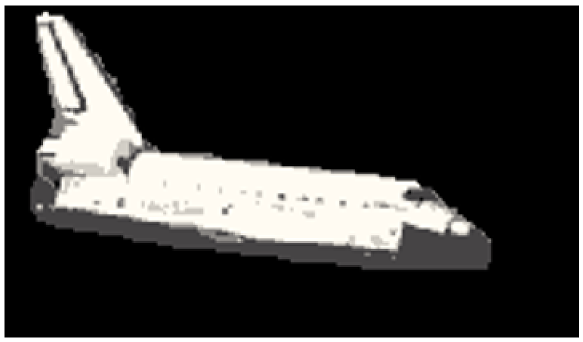


Figure 30: Color reduction in RGB color space. (a) input image , (b) resultant image with only 10 colors.

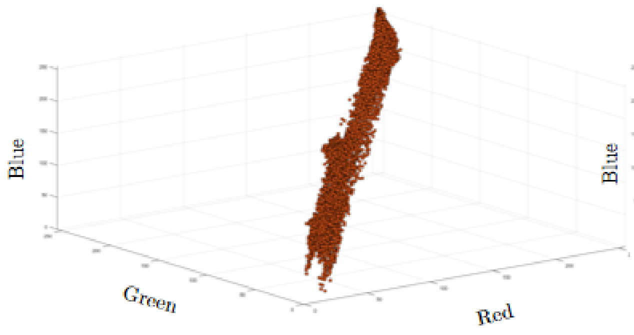
The compressed image resembles the actual image while retaining the primary image characteristics (figure 30).

Since $k-means++$ begins with a random selection of cluster centers from among the available data points, different algorithm runs can induce different clustering results. As a consequence, the results could not be repeatable. To address this non-repeatable issue, we first map two shapes (sh(A) and sh(B)) into a single 3-dimensional color space, and then we run the $k - means + +$ algorithm on those data points. This is guaranteed to have consistent centroid clusters (bin centers) for at least the selected sh(A) and sh(B). Figures 31 and 32 demonstrate the way to obtain resultant matrix in spatial color analyzer module.

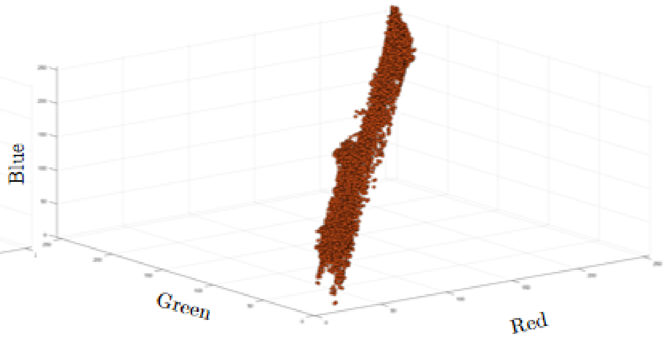
(a)



(b)

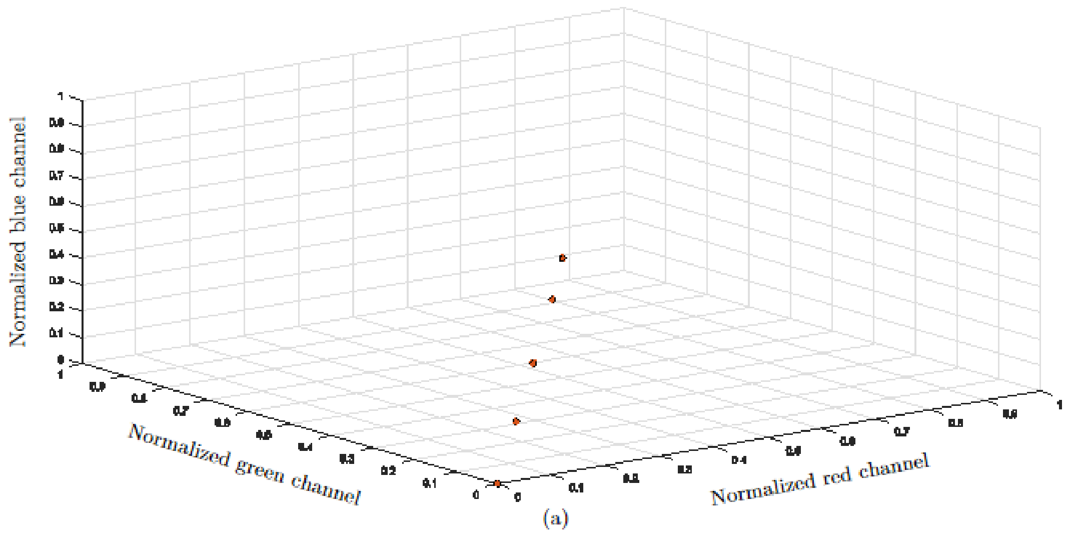


(c)

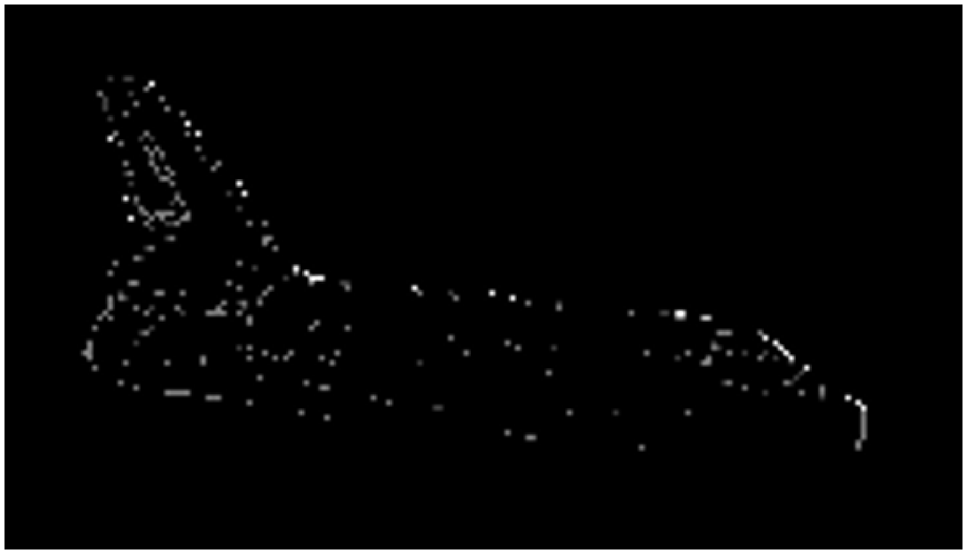


(d)

Figure 31: RGB color distribution for the input shapes. (a) input shape sh(A) in frame n, (b) input shape sh(B) in frame $n+1$, (c) sh(A) RGB color vector points in RGB color space, (d) sh(B) RGB color vector points in RGB color space



(a)



(b)

Figure 32: Output result from $k - mean++$ clustering. (a) new color centroids after applying $k - means++$ clustering, (b) Spatial color difference between two consecutive frames.

4.5 Track Assignment

The main objective of this module is to assign shapes to tracks to minimize the overall cost. Here, the overall cost can be defined as:

$$Overall\ cost\ between\ shape\ A\ and\ B\ (OC_{A,B}) = w_a * C_1 + w_b * C_2 \tag{39}$$

Where, $w_a, w_b = 0.5$, C_1 is the normalized similarity cost value in the shape context descriptor and C_2 is the normalized pixel count in the resultant matrix in spatial color analyzer module.

The following example describes the track assignment in brief. The figure 33 represents proposed region clusters whose are outputted from region estimation module. Here, region estimation module proposes two region clusters namely cluster 1 and cluster 2. In the first cluster, there are two shapes $(sh(x), sh(y))$ in the n^{th} frame that need to assigned to the shapes $(sh(x'), sh(y'))$ in $n + 1^{th}$ frame. To find the respective shape in the next frame and build the data continuity model, spatial color variation property and shape context measurement have been computed for all the shapes in both frames $(sh(x), sh(y), sh(x'), sh(y'))$ and the normalized results for region cluster 1 and 2 can be seen in table 7 and 8 respectively. Using equation 39 , tracks are assign in a way that minimize the overall cost. In this example, in region cluster 1, the minium overall cost value for shape x is 0.125 which maps frame n's $sh(x)$ to frame $n+1$'s $sh(x')$. Likewise $sh(y)$ maps to $sh(y')$. Similarly, for region cluster 2, based on the overall cost value for each shape, $sh(p)$ maps to $sh(p')$ and $sh(q)$ maps to $sh(q')$. This process will continue until the last video frame is computed. Depending on the architecture (online or offline), the shape tracked information will be produced as the final result.

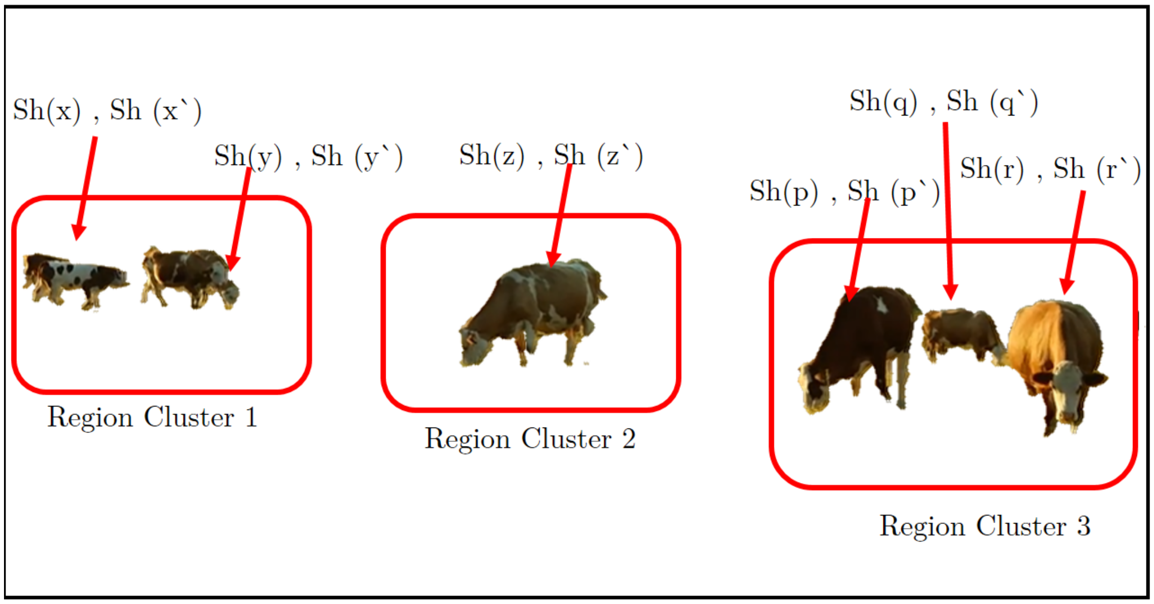


Figure 33: Region cluster representation. $Sh(i), Sh(i')$ $(i \in \{x, y, z, p, q, r\}, i' \in \{x', y', z', p', q', r'\})$ represent shapes extracted from two consecutive frames.

Table 7: Shape similarity measures for region cluster 1

	C_1	C_2	OC
xx`	0.1	0.15	0.125
xy`	0.5	0.8	0.65
yy`	0.1	0.12	0.11

Table 8: Shape similarity measures for region cluster 2

	C_1	C_2	OC
pp`	0.1	0.15	0.125
pq`	0.5	0.8	0.65
qq`	0.1	0.12	0.11

5 Kalman Filter Based Shape Trajectory Estimator

5.1 Introduction

Visual trackers are often subjected to different challenges when extracting features from a given video frame. Shape appearance deformation over time, illumination changes, background clutter, motion blur, occlusion, in-plane rotation, scale change, out-of-the-plane, and out-of-view are the main challenges that have been faced by visual trackers while extracting features. However, the proposed combination of a shape region estimator, a spatial color analyzer, and a shape context descriptor can tackle most of the above challenges. Nevertheless, shape tracking is still a challenging task due to the shape occlusion and/or the object detector's failure that changes the unique track id of the shape that has been tracked. Therefore, embedding a module that can handle shape occlusion and/or detection failure is necessary. The method that has been utilized here is estimating the relative shape trajectory concerning the camera viewpoint for occluded or misdetected shapes. The relative shape trajectory estimation is calculated using a Kalman filter (KF) based trajectory estimation algorithm that uses average acceleration magnitude components before occlusion or a shape's misdetection.

In this chapter, section 5.2 briefly outlines the KF based methods and particle filters that have been used to predict the trajectory of a shape/object. In contrast, section 5.3 explains the KF based trajectory estimation algorithm in detail.

5.2 Preliminaries

Linear models are used to represent the majority of physical systems because of their versatility and the availability of rich analytical techniques. Differential equations or difference equations, several equivalent means, the Laplace or z transform, and state-space representation can all be used to describe a linear dynamical system. The state-space representation alone has several benefits: providing an internal representation of the system, insight into the system's performance, and a straightforward way to create controllers for multiple-input multiple-output (MIMO) systems in general [79]. Therefore, state-space methods play a vital role in system analysis and control problems.

The state of a system at any given time is described in the state-space approach is the minimal information required to describe the system's response to an applied input. One or more variables defined as state variables have been used to describe the state of a system. The state equations (system dynamic model) are a series of equations that connect the current state variables to previous state variables and the most recent input. The output equations (system measurement model) are the equations that connect the output variables to the state variables and inputs. A linear dynamical system's state-space representation is a series of state equations and output equations [79] [80].

The state's estimate at a given time from measurements is an important problem in analyzing and monitoring a system based on a state-space model. State estimation (SE) is usually a two-step method that is repeated at each time step. The current inputs are compared with the previously estimated state to obtain a predicted value for the current state variables in the prediction phase. The available output measurements are then used to correct the predicted in the correction step. Figure 34 depicts this concept in a visual format.

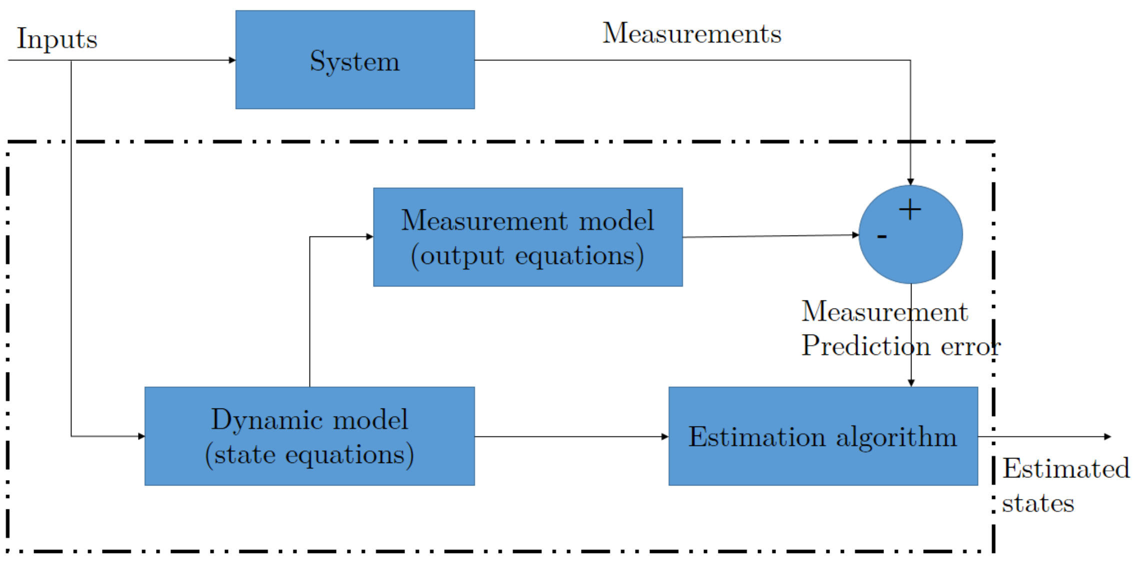


Figure 34: The state estimation process

Owing to noise, the inputs to a physical system and the measurements made by the system usually have a degree of uncertainty. As a result, any system's actual state is a challenge of minimizing or maximizing a cost function depending on specific parameters to find the best value.

5.2.1 Kalman Filter

Kalman filtering [81] is a technique for estimating uncertain variables based on observations collected over time. Kalman filters have shown to be useful in various systems and are simple in design, and require less computational power. Prediction and update are the main two steps of this algorithm. Kalman filters are also used to predict states in a state-space format based on linear dynamical systems.

The process model defines the evolution of the state from time k—1 to time k as:

$$\boldsymbol{x}_{k_{predicted}} = \boldsymbol{F}\boldsymbol{x}_{k-1_{estimated}} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k, \tag{40}$$

where F is the state transition matrix applied to the previous state vector x_{k1} , B is the control-input matrix applied to the control vector u_k , and w_k is the process noise vector that is assumed to be zero-mean Gaussian with the covariance Q , i.e., $w_k \sim \mathscr{N}(0, Q)$.

The process model is used in conjunction with the measurement model, which defines the relationship between the state and the measurement at time step k as follows:

$$\boldsymbol{z}_k = \boldsymbol{H}\boldsymbol{x}_{k_{predicted}} + \boldsymbol{v}_k, \tag{41}$$

where z_k is the measurement vector, H is the measurement matrix, and $_k$ is the measurement noise vector that is assumed to be zero-mean Gaussian with the covariance R , i.e., $v_k \sim \mathscr{N}(0, R)$ .It is worth noting that the word ”measurement” is often referred to as ”observation” in different texts.

It is worth noting that Kalman filters are based on the assumption that the process and measurement models are both linear. As a result, a Kalman filter can only have an optimal estimate if the assumption holds.

The other apparent filters to address this trajectory estimation problem are Extended Kalman filter [82], Unscented Kalman filter [83], and particle filter [84] where each is having its own merits and demerits. The summary of these filters can be listed in table 9.

Table 9: State estimator analysis results

State Estimator	Model	Assumed distribution	Computational Cost
Kalman filter	Linear	Gaussian	Low
Extended Kalman filter	Locally linear	Gaussian	Medium (if the Jacobians can be computed numerically)
Unscented Kalman filter	Nonlinear	Gaussian	Medium
Particle filter	Nonlinear	Non-Gaussian	High

Even though the Kalman filter has several drawbacks compared to other non-linear filters, the method used in this research to estimate the shapes' trajectory was the Kalman filter due to its low computational cost and fairly accurate estimations.

5.3 Shape Trajectory Estimator

This section describes the technique that has been used to minimize the issue of shape trajectory estimation that arises from shape occlusion and/or detection failure. Owing to the lower computational time complexity and simplicity in implementation, the shape trajectory estimation has been carried out by the KF algorithm.

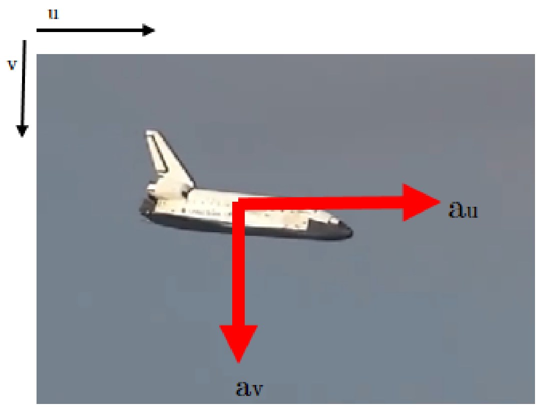


Figure 35: Newton's motion

Assuming that the tracking shape behaves according to equations of Newton's motion [85] during the occlusion period, (42) and (43) have been used to describe the trajectory of the shape in two-dimension space.

$$s = ut + \frac{1}{2}at^2, \tag{42}$$

$$v = u + at, \tag{43}$$

Where s is the displacement, u is the initial velocity, v is the final velocity, t is the time for displacement, and a is the constant acceleration.

To apply the KF algorithm to estimate the shape's trajectory, it is assumed that the shape behaves according to a linear system model, and the measurement noise is assumed to be Gaussian.

Now we can write the shape trajectory state-space model as follows.

$$\boldsymbol{x}_{k_{predicted}} = \boldsymbol{A}\boldsymbol{x}_{k-1_{estimated}} + \boldsymbol{B}\boldsymbol{u}_k + \boldsymbol{w}_k, \tag{44}$$

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_{k_{predicted}} + \boldsymbol{z}_k, \tag{45}$$

where k is the time index, $\boldsymbol{x}_{k_{predicted}} \in \mathbb{R}^4$ is the predicted current state, $\boldsymbol{A} \in \mathbb{R}^{4\times4}$ is the state transition matrix, $\boldsymbol{B} \in \mathbb{R}^{4\times2}$ is the control input matrix, $\boldsymbol{u}_k \in \mathbb{R}^2$ is the control vector , $\boldsymbol{y}_k \in \mathbb{R}^2$ is the measurement vector, $H \in \mathbb{R}^{2\times4}$ is the observation matrix and $\boldsymbol{w}_k \in \mathbb{R}^4$ and $\boldsymbol{z}_k \in \mathbb{R}^2$ are random disturbance vectors.
Assuming random vector $\boldsymbol{w}_k = 0$, we can write (44) as follows.

$$\boldsymbol{x}_{k_{predicted}} = \boldsymbol{A}\boldsymbol{x}_{k-1_{estimated}} + \boldsymbol{B}\boldsymbol{u}_k. \tag{46}$$

Using (46) and (45), for the purpose of shape trajectory estimation, the state equation can be written as follows,

$$\boldsymbol{x}_{k_{predicted}} = \begin{bmatrix} 1 & 0 & \Delta t & 0 \\ 0 & 1 & 0 & \Delta t \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} u \\ v \\ \dot{u} \\ \dot{v} \end{bmatrix} + \begin{bmatrix} \frac{\Delta t^2}{2} & 0 \\ 0 & \frac{\Delta t^2}{2} \\ \Delta t & 0 \\ 0 & \Delta t \end{bmatrix} \begin{bmatrix} a_u \\ a_v \end{bmatrix}, \tag{47}$$

$$\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_{k_{predicted}} + \boldsymbol{z}_k, \tag{48}$$

where $\Delta t = \frac{1}{fps}$, u and v are the coordinate axes in a selected two dimensional space, $\dot{u}$ and $\dot{v}$ are the velocity components in u and v directions accordingly, and a_u and a_v are the acceleration components in the u and v directions.

Assuming the initial state of the system is known ($\boldsymbol{x}_{0_{estimated}}, \boldsymbol{P}_{0_{estimated}}$) and the measurement covariance matrix $\boldsymbol{R} \in \mathbb{R}^{2\times 2}$, we can use Kalman filter to estimate the state of the system at time k that is compatible with observations upto time instance k, the system model, and the assumptions made.

Given $\boldsymbol{x}_{k-1_{estimated}}$ and $\boldsymbol{P}_{k-1_{estimated}}$, the predicted state and the predicted covariance matrix ($\boldsymbol{P}_{k_{predicted}}$) at time step k can be calculated using (47).

$$\boldsymbol{P}_{k_{predicted}} = \boldsymbol{A}\boldsymbol{P}_{k-1}\boldsymbol{A}^T. \tag{49}$$

As and when the measurements are available at the time step k, the predicted state can be corrected. Assuming that $\boldsymbol{x}_{k_{estimated}} = \boldsymbol{x}_{k_{predicted}} + \boldsymbol{K}_k\boldsymbol{\delta}_k$. By minimizing the sum of variances in $\boldsymbol{P}_{k_{predicted}}$ with respect to $\boldsymbol{K}_k$ and taking $\boldsymbol{\delta}_k = \boldsymbol{y}_k - \boldsymbol{H}\boldsymbol{x}_{k_{predicted}}$ we obtain (50).

$$\boldsymbol{K}_k = \frac{\boldsymbol{P}_{k_{predicted}}\boldsymbol{H}^T}{\boldsymbol{H}\boldsymbol{P}_{k_{predicted}}\boldsymbol{H}^T + \boldsymbol{R}}. \tag{50}$$

Therefore the estimated state values are

$$\boldsymbol{x}_{k_{estimated}} = \boldsymbol{x}_{k_{predicted}} + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{H}\boldsymbol{x}_{k_{predicted}}), \tag{51}$$

$$\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_{k_{predicted}}, \tag{52}$$

where I is the identity matrix.

The Algorithm 7 is used to obtain the Kalman filter estimates.

Algorithm 7: Two dimensional Kalman filter based trajectory estimator

```
1 $\boldsymbol{x}_{k_{predicted}} = \boldsymbol{A}\boldsymbol{x}_{k-1_{estimated}} + \boldsymbol{B}\boldsymbol{u}_k.$;
2 $\boldsymbol{y}_k = \boldsymbol{H}\boldsymbol{x}_{k_{predicted}} + \boldsymbol{z}_k$;
3 $\boldsymbol{P}_{k_{predicted}} = \boldsymbol{A}\boldsymbol{P}_{k-1}\boldsymbol{A}^T$;
4 $\boldsymbol{K}_k = \frac{\boldsymbol{P}_{k_{predicted}}\boldsymbol{H}^T}{\boldsymbol{H}\boldsymbol{P}_{k_{predicted}}\boldsymbol{H}^T + \boldsymbol{R}}$;
5 $\boldsymbol{x}_{k_{estimated}} = \boldsymbol{x}_{k_{predicted}} + \boldsymbol{K}_k(\boldsymbol{y}_k - \boldsymbol{H}\boldsymbol{x}_{k_{predicted}})$;
6 $\boldsymbol{P}_k = (\boldsymbol{I} - \boldsymbol{K}_k\boldsymbol{H})\boldsymbol{P}_{k_{predicted}}$;
```

The figure 36 represents two dimensional tracking data with occlusion. The discontinuation in the x-axis represents losses of data continuity of the tracking shape due to its occlusion or a detection failure. This Kalman filter trajectory estimation module aims to find and build the relationships between these data discontinuations. For this, we start from the end of the barcode graph. We select the last global shape ID as the master track and, depending on the availability, select up to 5 tracks (from the end of the barcode graph) as slave tracks that are not overlapped with the master track and show a relative closeness to the master track in terms of the last frame number of each slave tracks. Then we apply SIFT keypoint descriptor to find connections between each of the slave tracks to the master tracks. From that, we select up to two slave tracks that have maximum SIFT keypoint connections to the master track. After that, we extract each of the centroid positions in the proposed two slave tracks and calculate each of the average relative x and y components of the acceleration magnitude denoted as a_x and a_y in equation 47. and we selected $\Delta t = \frac{1}{fps}$, where fps = frames per seconds in the selected video sequence. After that, we use small values (in this case $= 0.5$) for each element in the initial process covariance matrix P_{k-1} and the noise covariance matrix R. We initiate the Kalman filter estimation from the available data from the slave tracks and continue the iterations until it reaches the frame number of the master track. Then, we compare the results (expected centroid position) from the Kalman filter estimation to the master track's shape actual centroid position. Finally, we assign the slave track to the master track, which shows a minimum expected centroid position error. Figure 37 represents SIFT based slave track proposals, figure 38 demonstrates Kalman filter based estimation results, figure 39 depicts the two dimensional tracking data after applying the Kalman filter and figure 40 represents the ground truth tracking data. From this, it can conclude that the shape trajectory estimation module helps to overcome data continuation issues arising from shape occlusion.

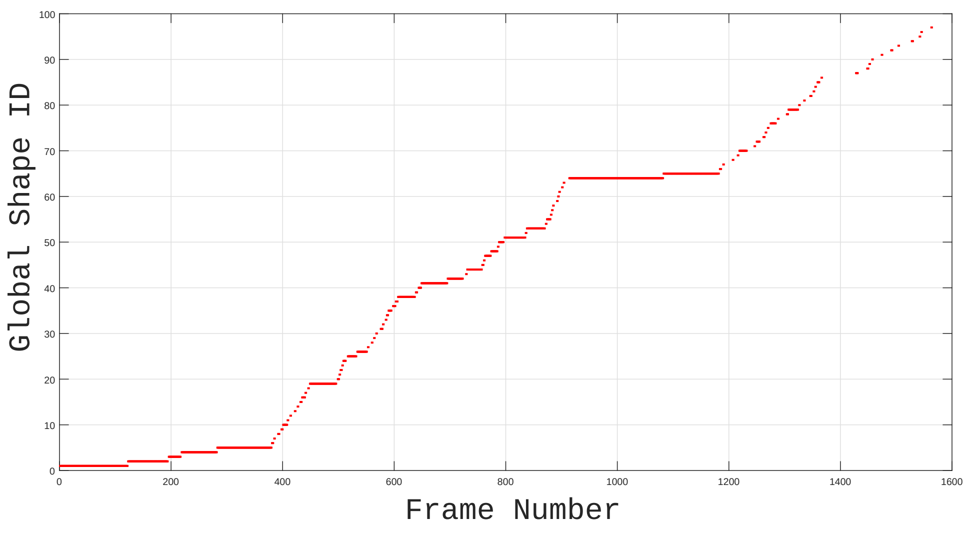


Figure 36: Two dimensional barcode - with occulussion

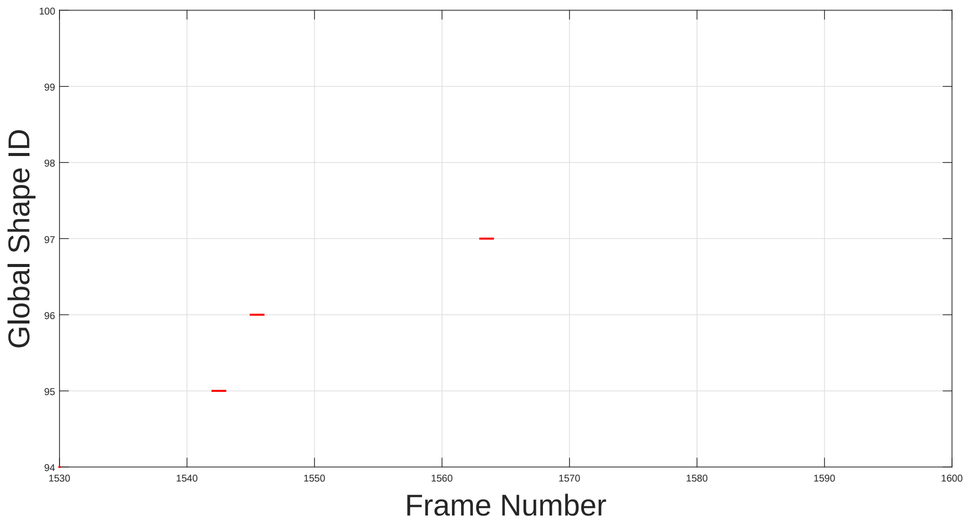
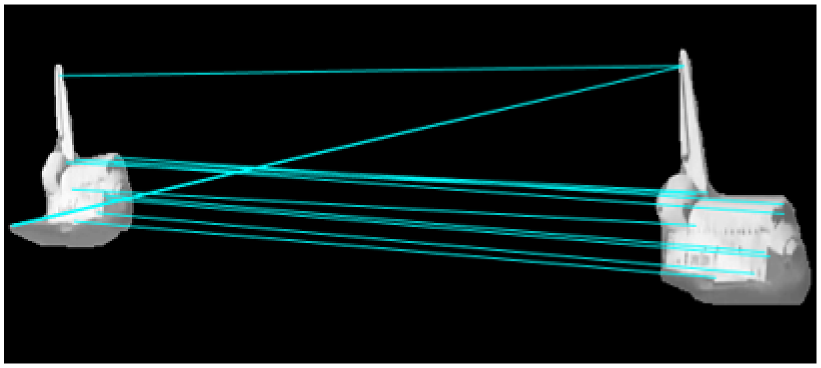
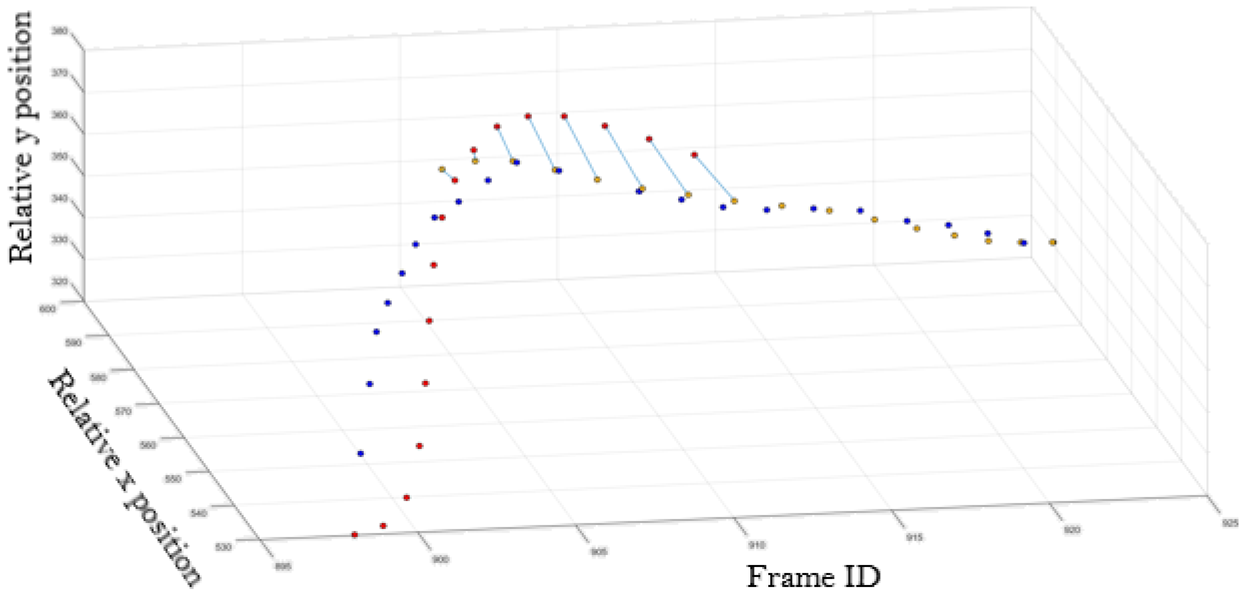


Figure 37: Proposed slave tracks by SIFT keypoint connections. Algorithm chooses global shape ID 95,96 as slave tracks and 97 as the master track.



(a)



(b)

Figure 38: Estimation results from Kalman filter (positions are measured in terms of pixel distances). (a) SIFT connections between track id i and j (b) red - actual relative positions for track id i , blue - actual relative positions for track id j, yellow - Kalman filter based estimations for occluded frames

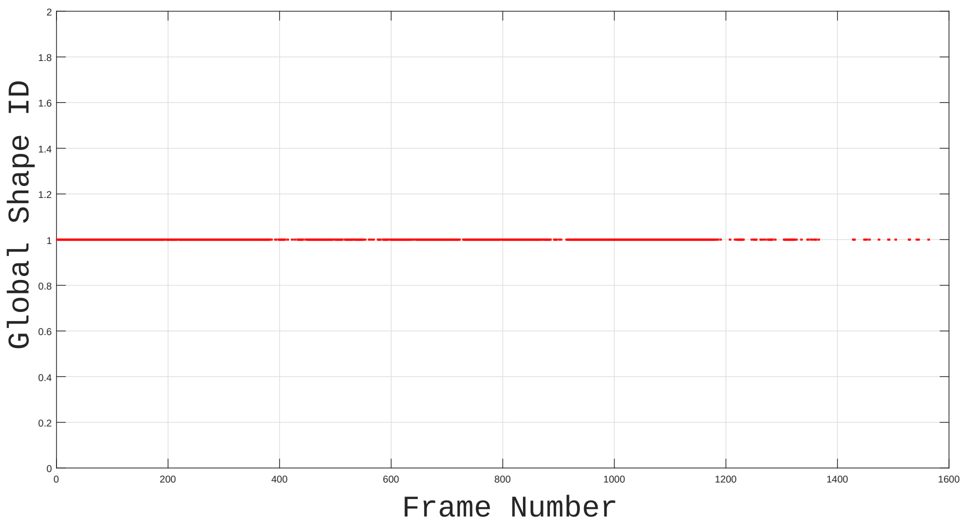


Figure 39: Two dimensional barcode - after Kalman filtering

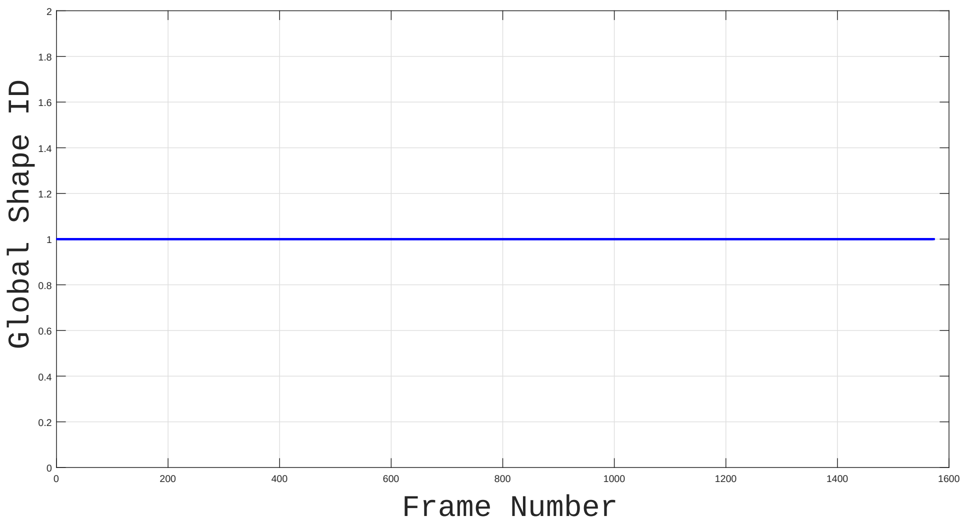


Figure 40: Two dimensional barcode tracking data - Ground truth

6 Results

Python and Matlab are used to implement the proposed tracking method. This section contains all of the test cases that are done on a laptop with Intel(R) Core(TM) CPU I7-6700HQ with 2.60 GHz 8 processors. The code is not designed for parallel processing, despite the fact that there are eight processors. VOT-ST2020 [86] and MOT20 [87] are the popular benchmarks for visual object tracking performance evolution. Both datasets were used to validate the proposed tracking method.

VOT-ST2020 public dataset includes 60 challenging sequences, and all frames are manually segmented (only ground truth segmentation masks are available). Apart from that, each frame is annotated by six attributes (illumination change, occlusion, object size change, unassingment, and object motion camera motion)

On the other hand, MOT20 dataset consists of 4 training sequences and 4 test sequences. MOT20 training set has a total of 8,931 frames, and the test set contains a total of 4,479 frames. We used both testing and training datasets as testbeds in our experiments because the proposed tracking system does not need to train on training data.

6.1 VOT-ST2020 Challenge: (Visual Object Tracking - Short Term)

The VOT challenge contest offers a collection of Octave/Matlab scripts to evaluate the accuracy of trackers written in Python, Matlab, or C++. This toolkit helps to compare the results of various trackers using metrics.

VOT-ST2020 evaluation metrics:

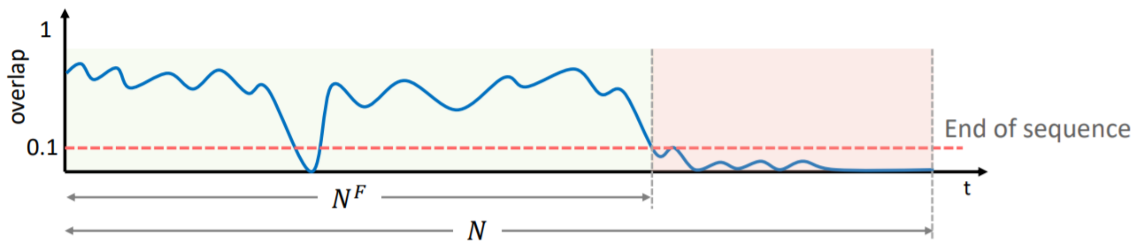


Figure 41: VOT evaluation metric term definitions [10]

Accuracy : On a subsequence starting from an anchor a of the sequence s, the accuracy $A_{s,a}$ is defined as the average overlap between the target predictions[1] and the ground truth[2] calculated from the frames before the tracker fails on that subsequence [10].

$$A_{s,a} = \frac{1}{N^F_{s,a}} \sum_{i=1:N^F_{s,a}} \Omega_{s,a}(i) \tag{53}$$

Where $N^F_{s,a}$ is the number of frames before the tracker failed in the subsequence starting at anchor a in the sequence s and $\Omega_{s,a}(i)$ is the overlap between the prediction and the ground truth at frame i.

Robustness: Robustness measure Rs, a is defined as the extent of the sub-sequence before the tracking failure.

$$R_{s,a} = \frac{N^F_{s,a}}{N_{s,a}} \tag{54}$$

where $N_{s,a}$ is the number of frames of the subsequence.
Expected average overlap (EAO): The value of the EAO curve $\hat{\Phi}_i$ at sequence length i is thus defined as:

$$\hat{\Phi}_i = \frac{1}{|\mathcal{S}(i)|} \sum_{s,a \in \mathcal{S}(i)} \Phi_{s,a}(i) \tag{55}$$

Where $\Phi_{s,a}(i)$ is the average overlap calculated between the first and i-th frame of the extended sub-sequence starting at anchor a of the sequence s, $\mathcal{S}(i)$ is the set of

[1]Proposed tracking model output results
[2]Human evaluation for each frame in a sequence and annotates objects' location manually.

the extended subsequences with length greater or equal to i and $|\mathcal{S}(i)|$ is the number of these sub-sequences.

The EAO measure is then calculated by averaging the EAO curve from N_{lo} to N_{hi},

$$EAO = \frac{1}{N_{hi} - N_{lo}} \sum_{i=N_{lo}:N_{hi}} \hat{\Phi}_i \tag{56}$$

The authors of the VOT [86] dataset has defined the interval bounds : $[N_{lo}, N_{hi}] = [115, 755]$.

In addition to that, for all of the calculations, anchors have placed 50 frames apart. At each anchor, the tracker is initialized. $\theta_\phi = 0.1$ and $\theta_N = 10$ frames.
Table 10 depicts the results obtained from the VOT-ST2020 toolkit for the proposed offline tracking architecture.

Table 10: VOT-ST2020 results for the proposed online tracker

Accuracy	Robustness	EAO
0.750	0.798	0.480

6.2 MOT20 Challenge: (Multi Object Tracking)

The MOT challenge [87] also provides a toolkit to evaluate the performance of trackers. This toolkit helps to compare the results of various trackers using metrics.
Multiple Object Tracking Accuracy (MOTA):

$$MOTA = 1 - \frac{\sum_t (fn_t + fp_t + IDS_t)}{\sum_t g_t} \tag{57}$$

Multiple Object Tracking Precision (MOTP):

$$MOTP = \frac{\sum_{i,t} d_t^i}{\sum_t c_t} \tag{58}$$

where t is the frame t, g is ground truth detections, fp is false positives, fn is false negatives, c_t is correct matches found at frame t and d_t^i is the distance between ground truth detection and predicted detection for each correct detection which is taken as the intersection of union between the two bounding boxes.

Other MOT evaluation metrices [87]:

- Mostly Lost (ML: percentage of ground truth trajectories which are covered by tracker output for less than 20% in length)
- Mostly Tracked (MT: percentage of ground truth trajectories which are covered by tracker output for more than 80% in length)
- Identity Switches (IDS: The total of number of times that a tracked trajectory changes its matched ground truth identity)

The proposed tracking method is compared to other state-of-the-art online/offline trackers that are published in the two benchmarks.Table 11 and 12 show a summary of the findings of the selected trackers. In the comparison, the proposed method is evaluated against best trackers that are published in MOT20 [87] and VOT-ST2020 [86] benchmarks.

In the VOT-ST2020 benchmark for published online tracking methods, our method is ranked fourth (in terms of EAO). However, it should be noted that RPT [88], Ocean-Plus [89], and AlphaRef [90] are based on trainable machine learning architectures, whereas our method can be classified as zero-shot learning. In addition to that, our offline method is ranked first (in terms of Robustness), achieving 0.870 score. Since it is not fair to compare offline results with online results, we did not include that in the table 11. The proposed tracker is ranked fourth in MOT20, out of all the published online and offline tracking methods (in terms of MOTA). Nevertheless, it showed the second-best IDS for the proposed offline tracking method.

Table 11: VOT-ST2020 results overall comparison

	EAO	A	R
VOT-ST2020			
RPT [2]	0.530	0.700	0.869
OceanPlus[3]	0.491	0.685	0.842
AlphaRef 3]	0.482	0.754	0.777
AFOD [25]	0.472	0.713	0.795
LWTL [31]	0.463	0.719	0.798
fastOcean[29]	0.461	0.693	0.803
Ours - online	0.480	0.750	0.798

Table 12: MOT20 results overall comparison

	MOTA	MOTP	MT	ML	IDS
MOT20					
MPNTrack[2]	57.6	79.0	474	279	1,210
TrTest[3]	57.0	79.7	499	243	5,271
center_reid[3]	56.6	77.0	668	141	4,643
ALBOD[25]	56.5	79.4	506	228	3,727
LPC_MOT [31]	56.3	79.7	424	313	1,562
FGRNetIV[29]	55.4	79.4	508	221	2,159
Ours - Offline	56.5	77.2	410	220	1,280
Ours - Online	56.5	77.2	410	220	2,159

Hyperparameters: To obtain the results shown in table 12, the following parameters were chosen for the following modules. In spatial color analyzer module, we use K=10 for the kmeans++ algorithm. For track assignment weights in equation 39, we use $w_a = w_b = 0.5$ giving same attention to both results obtained from spatial color

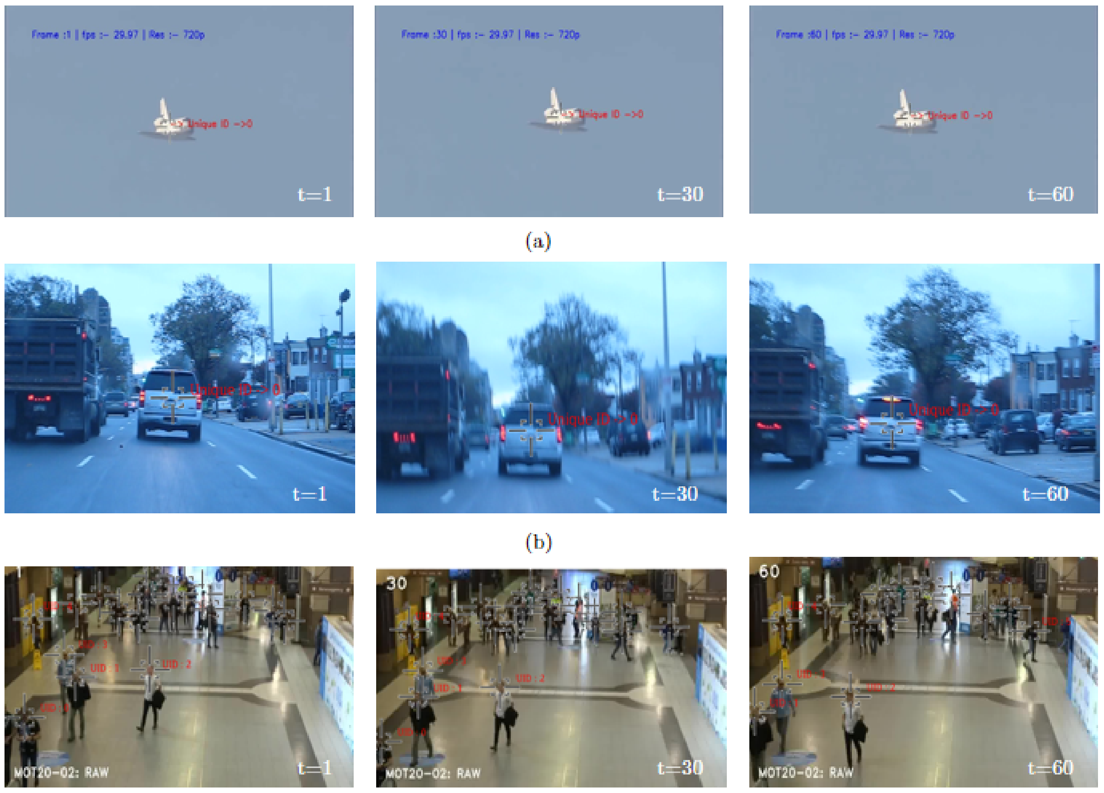

Frame :1 | fps :- 29.97 | Res :- 720p
Unique ID ->0
t=1
Frame :30 | fps :- 29.97 | Res :- 720p
t=30
Frame :60 | fps :- 29.97 | Res :- 720p
t=60
(a)
(b)
MOT20-02: RAW
(c)

Figure 42: Results are being recorded over a few frames. (a) tracking result from a random Youtube video sequence, (b) tracking results from VOT-ST2020 training data, (c) tracking results from MOT20 training data. At each sequence, unique ids displayed maximum up to four objects. For more than 60 frames, these identities are tracked individually.

analyzer module and the shape context descriptor.

Python and C++ were used to implement the proposed tracking method, and it achieved reasonable results, making it suitable to be used in online/offline video processing applications. The tracking results on a couple of random frames are shown in the figure 42, where the track ID is stable across the frames.

In a track-by-detection framework, shape tracking performance depends on the detection accuracy of the shape detector. For this experiment, I used Detectron2 [44] (a Mask R-CNN model) along with pre-trained weights from the COCO dataset [45]. If it is a different detector, the above results may change.

7 Conclusion

In this research, I have introduced a shape tracker based on computational geometric concepts that can achieve state-of-the-art tracking performance for both VOT-ST2020 and MOT20 benchmark datasets. The running time increases with the scene's complexity (e.g., on the number of shapes present in the frame). The proposed shape tracker is based on a combination of region estimator, shape context descriptor, and spatial color variation modules. Furthermore, In this work, I propose a method for reestablishing tracking identity when detection fails for a short period of time between frames. The selection of a shape detector is critical for any trackers that are based on a track-by-detection framework as the tracking accuracy may also depend on the detection performance. Therefore, with the selection of Detectron (pre-trained Mask R-CNN model) as the shape detector, this method shows a competitive performance when there are fewer false positives in shape detection.

7.1 Recommendations for Future Work

The shape tracking approach used in this thesis could be expanded in the following ways. Instead of using a linear filter, a particle filter-based approach with a parallel processing platform can be a potential research direction to address the shape occlusion problem in any stable tracks. In addition, replacing the object detector with an offline trained machine learning model inspired by a Siamese network [91] will open new potential research areas in the domain of visual shape tracking. Finally, there is a possible research extension about weighting similarity values produced by the spatial color analyzer module and the shape context descriptor module and measure how it affects the tracker performance.

References

[1] E. W. Weisstein, “Voronoi diagram,” https://mathworld.wolfram.com/VoronoiDiagram.html.

[2] B. Rezaei, A. Farnoosh, and S. Ostadabbas, “G-lbm: Generative low-dimensional background model estimation from video sequences,” in European Conference on Computer Vision. Springer, 2020, pp. 293–310.

[3] NASA. Space shuttle atlantis sts-129 hd landing, november 27, 2009, runway 33, kennedy space center. NASA. [Online]. Available: https://www.youtube.com/watch?v=5Qj3on0VTSs

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

[5] Thermos, Abduction of a Sabine Woman, Mar 2007. [Online]. Available: https://commons.wikimedia.org/wiki/File:Giambologna_sabine.jpg

[6] G. Levin and B. Dorsey, image processing and computer vision. openframeworks. [Online]. Available: http://openframeworks.kr/ofBook/chapters/image_processing_computer_vision.html

[7] Barnetts, rgb color space. cleanpng. [Online]. Available: https://www.cleanpng.com/png-rgb-color-space-rgb-color-model-light-4170880/

[8] I. S. Técnico, Log-Polar Mapping. Instituto Superior Técnico. [Online]. Available: http://users.isr.ist.utl.pt/~alex/Projects/TemplateTracking/logpolar.htm

[9] M. Teichmann. mit. [Online]. Available: http://groups.csail.mit.edu/graphics/classes/6.838/S98/meetings/m25/m25.html

[10] A. Bartoli and A. Fusiello, Computer Vision-ECCV 2020 Workshops: Glasgow, UK, August 23-28, 2020, Proceedings, Part VI. Springer Nature, 2020, vol. 12540.

[11] Y. Wu, R. Gao, J. Park, and Q. Chen, “Future video synthesis with object motion prediction,” in Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 5539–5548.

[12] R. Sarcinelli, R. Guidolini, V. B. Cardoso, T. M. Paixão, R. F. Berriel, P. Azevedo, A. F. De Souza, C. Badue, and T. Oliveira-Santos, “Handling pedestrians in self-driving cars using image tracking and alternative path generation with frenét frames,” Computers & Graphics, vol. 84, pp. 173–184, 2019.

[13] C. Bregler, K. Bhat, J. Saltzman, and B. Allen, “Ilm’s multitrack: a new visual tracking framework for high-end vfx production,” in SIGGRAPH 2009: Talks, 2009, pp. 1–1.

[14] C.-C. Hsieh and S.-S. Hsu, “A simple and fast surveillance system for human tracking and behavior analysis,” in 2007 Third International IEEE Conference on Signal-Image Technologies and Internet-Based System. IEEE, 2007, pp. 812–818.

[15] A. Berg, J. Ahlberg, and M. Felsberg, “A thermal object tracking benchmark,” in 2015 12th IEEE International Conference on Advanced Video and Signal Based Surveillance (AVSS). IEEE, 2015, pp. 1–6.

[16] H. Karunasekera, H. Wang, and H. Zhang, “Multiple object tracking with attention to appearance, structure, motion and size,” IEEE Access, vol. 7, pp. 104 423–104 434, 2019.

[17] H. Wu, W. Han, C. Wen, X. Li, and C. Wang, “3d multi-object tracking in point clouds based on prediction confidence-guided data association,” IEEE Transactions on Intelligent Transportation Systems, pp. 1–10, 2021.

[18] S. Tang, M. Andriluka, B. Andres, and B. Schiele, “Multiple people tracking by lifted multicut and person re-identification,” in Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition, 2017, pp. 3539–3548.

[19] R. Henschel, L. Leal-Taixé, D. Cremers, and B. Rosenhahn, “Fusion of head and full-body detectors for multi-object tracking,” in Proceedings of the IEEE conference on computer vision and pattern recognition workshops, 2018, pp. 1428–1437.

[20] A. Sadeghian, A. Alahi, and S. Savarese, “Tracking the untrackable: Learning to track multiple cues with long-term dependencies,” in Proceedings of the IEEE International Conference on Computer Vision, 2017, pp. 300–311.

[21] B. Lee, E. Erdenee, S. Jin, M. Y. Nam, Y. G. Jung, and P. K. Rhee, “Multi-class multi-object tracking using changing point detection,” in European Conference on Computer Vision. Springer, 2016, pp. 68–83.

[22] W. Choi, “Near-online multi-target tracking with aggregated local flow descriptor,” in Proceedings of the IEEE international conference on computer vision, 2015, pp. 3029–3037.

[23] J. H. Yoon, C.-R. Lee, M.-H. Yang, and K.-J. Yoon, “Online multi-object tracking via structural constraint event aggregation,” in Proceedings of the IEEE Conference on computer vision and pattern recognition, 2016, pp. 1392–1400.

[24] C. Kim, F. Li, A. Ciptadi, and J. M. Rehg, “Multiple hypothesis tracking revisited,” in Proceedings of the IEEE international conference on computer vision, 2015, pp. 4696–4704.

[25] S. Sharma, J. A. Ansari, J. K. Murthy, and K. M. Krishna, “Beyond pixels: Leveraging geometry and shape cues for online multi-object tracking,” in 2018 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2018, pp. 3508–3515.

[26] H. W. Kuhn, “The hungarian method for the assignment problem,” Naval research logistics quarterly, vol. 2, no. 1-2, pp. 83–97, 1955.

[27] J. Ren, X. Chen, J. Liu, W. Sun, J. Pang, Q. Yan, Y.-W. Tai, and L. Xu, “Accurate single stage detector using recurrent rolling convolution,” in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 5420–5428.

[28] Y. Xiang, W. Choi, Y. Lin, and S. Savarese, “Subcategory-aware convolutional neural networks for object proposals and detection,” in 2017 IEEE winter conference on applications of computer vision (WACV). IEEE, 2017, pp. 924–933.

[29] G. Gündüz and T. Acarman, "A lightweight online multiple object vehicle tracking method," in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 427–432.

[30] Y. Xiang, A. Alahi, and S. Savarese, "Learning to track: Online multi-object tracking by decision making," in Proceedings of the IEEE international conference on computer vision, 2015, pp. 4705–4713.

[31] K. Fang, Y. Xiang, X. Li, and S. Savarese, "Recurrent autoregressive networks for online multi-object tracking," in 2018 IEEE Winter Conference on Applications of Computer Vision (WACV). IEEE, 2018, pp. 466–475.

[32] L. Chen, H. Ai, Z. Zhuang, and C. Shang, "Real-time multiple people tracking with deeply learned candidate selection and person re-identification," in 2018 IEEE International Conference on Multimedia and Expo (ICME). IEEE, 2018, pp. 1–6.

[33] Y.-c. Yoon, A. Boragule, Y.-m. Song, K. Yoon, and M. Jeon, "Online multi-object tracking with historical appearance matching and scene adaptive detection filtering," in 2018 15th IEEE International conference on advanced video and signal based surveillance (AVSS). IEEE, 2018, pp. 1–6.

[34] J. Zhu, H. Yang, N. Liu, M. Kim, W. Zhang, and M.-H. Yang, "Online multi-object tracking with dual matching attention networks," in Proceedings of the European Conference on Computer Vision (ECCV), 2018, pp. 366–382.

[35] M. Danelljan, G. Bhat, F. Shahbaz Khan, and M. Felsberg, "Eco: Efficient convolution operators for tracking," in Proceedings of the IEEE conference on computer vision and pattern recognition, 2017, pp. 6638–6646.

[36] H. Wu, Y. Hu, K. Wang, H. Li, L. Nie, and H. Cheng, "Instance-aware representation learning and association for online multi-person tracking," Pattern Recognition, vol. 94, pp. 25–34, 2019.

[37] S.-H. Lee, M.-Y. Kim, and S.-H. Bae, "Learning discriminative appearance models for online multi-object tracking with appearance discriminability measures," IEEE Access, vol. 6, pp. 67 316–67 328, 2018.

[38] Z. Fu, F. Angelini, J. Chambers, and S. M. Naqvi, "Multi-level cooperative fusion of gm-phd filters for online multiple human tracking," IEEE Transactions on Multimedia, vol. 21, no. 9, pp. 2277–2291, 2019.

[39] Z. Fu, P. Feng, F. Angelini, J. Chambers, and S. M. Naqvi, "Particle phd filter based multiple human tracking using online group-structured dictionary learning," IEEE access, vol. 6, pp. 14 764–14 778, 2018.

[40] S. Scheidegger, J. Benjaminsson, E. Rosenberg, A. Krishnan, and K. Granström, "Mono-camera 3d multi-object tracking using deep learning detections and pmbm filtering," in 2018 IEEE Intelligent Vehicles Symposium (IV). IEEE, 2018, pp. 433–440.

[41] A. Osep, W. Mehner, M. Mathias, and B. Leibe, "Combined image-and world-space tracking in traffic scenes," in 2017 IEEE International Conference on Robotics and Automation (ICRA). IEEE, 2017, pp. 1988–1995.

[42] A. Sobral and A. Vacavant, “A comprehensive review of background subtraction algorithms evaluated with synthetic and real videos,” Computer Vision and Image Understanding, vol. 122, pp. 4–21, 2014.

[43] A. H. Lai and N. H. Yung, “A fast and accurate scoreboard algorithm for estimating stationary backgrounds in an image sequence,” in 1998 IEEE international symposium on circuits and systems (ISCAS), vol. 4. IEEE, 1998, pp. 241–244.

[44] Y. Wu, A. Kirillov, F. Massa, W.-Y. Lo, and R. Girshick, “Detectron2,” https://github.com/facebookresearch/detectron2, 2019.

[45] T.-Y. Lin, M. Maire, S. Belongie, J. Hays, P. Perona, D. Ramanan, P. Dollár, and C. L. Zitnick, “Microsoft coco: Common objects in context,” in European conference on computer vision. Springer, 2014, pp. 740–755.

[46] S. R. Joshi and R. Koju, “Study and comparison of edge detection algorithms,” in 2012 Third Asian Himalayas international conference on internet. IEEE, 2012, pp. 1–5.

[47] R. Gonzalez and R. Woods, “Digital image processing,” 1992.

[48] J. Canny, “A computational approach to edge detection,” IEEE Transactions on pattern analysis and machine intelligence, no. 6, pp. 679–698, 1986.

[49] L. G. Roberts, “Machine perception of three-dimensional solids,” Ph.D. dissertation, Massachusetts Institute of Technology, 1963.

[50] R. Haralick and L. Shapiro, “Computer and robot vision,” 1992.

[51] R. Gonzalez and R. Woods, “Digital image processing,” 1992.

[52] S. K. Katiyar and P. Arun, “Comparative analysis of common edge detection techniques in context of object extraction,” arXiv preprint arXiv:1405.6132, 2014.

[53] F. P. James, Computational Geometry, Topology and Physics of Digital Images with Application: Shape... Complexes, Optical Vortex Nerves and Proximities. Springer Nature, 2019.

[54] E. G. Gilbert, D. W. Johnson, and S. S. Keerthi, “A fast procedure for computing the distance between complex objects in three-dimensional space,” IEEE Journal on Robotics and Automation, vol. 4, no. 2, pp. 193–203, 1988.

[55] “Distance transform of a binary image, year = 2015a note = The MathWorks, Natick, MA, USA.”

[56] E. Agu, “Digital image processing,” 2015a.

[57] C. F. Weiman and G. Chaikin, “Logarithmic spiral grids for image processing and display,” Computer Graphics and Image Processing, vol. 11, no. 3, pp. 197–226, 1979.

[58] M. Hassaballah, A. A. Abdelmgeid, and H. A. Alshazly, “Image features detection, description and matching,” in Image Feature Detectors and Descriptors. Springer, 2016, pp. 11–45.

[59] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," International journal of computer vision, vol. 60, no. 2, pp. 91–110, 2004.

[60] H. Bay, A. Ess, T. Tuytelaars, and L. Van Gool, "Speeded-up robust features (surf)," Computer vision and image understanding, vol. 110, no. 3, pp. 346–359, 2008.

[61] E. Rublee, V. Rabaud, K. Konolige, and G. Bradski, "Orb: An efficient alternative to sift or surf," in 2011 International conference on computer vision. Ieee, 2011, pp. 2564–2571.

[62] K. Mikolajczyk and C. Schmid, "A performance evaluation of local descriptors," IEEE transactions on pattern analysis and machine intelligence, vol. 27, no. 10, pp. 1615–1630, 2005.

[63] E. Rosten and T. Drummond, "Machine learning for high-speed corner detection," in European conference on computer vision. Springer, 2006, pp. 430–443.

[64] M. Calonder, V. Lepetit, C. Strecha, and P. Fua, "Brief: Binary robust independent elementary features," in European conference on computer vision. Springer, 2010, pp. 778–792.

[65] S. A. K. Tareen and Z. Saleem, "A comparative analysis of sift, surf, kaze, akaze, orb, and brisk," in 2018 International Conference on Computing, Mathematics and Engineering Technologies (iCoMET), 2018, pp. 1–10.

[66] Weisstein, Eric W., "Collinear," https://mathworld.wolfram.com/Collinear.html.

[67] H. S. M. Coxeter and S. L. Greitzer, "Collinearity and concurrence," in Ch. 3 in Geometry Revisited. Washington, DC: Math. Assoc. Amer, 1967, pp. 51–79.

[68] R. Honsberger, "Episodes in nineteenth and twentieth century euclidean geometry," in Washington, DC: Math. Assoc. Amerr, 1995, pp. 153–154.

[69] C. Kimberling, "Triangle centers and central triangles," in Congr. Numer. 129, 1998, pp. 1–295.

[70] E. W. Weisstein, "Affine transformation," https://mathworld. wolfram. com/, 2004.

[71] D. Zwillinger, CRC standard mathematical tables and formulas. CRC press, 2018.

[72] A. Gray, E. Abbena, S. Salamon et al., "Modern differential geometry of curves and surfaces with mathematica," 2006.

[73] H. T. Croft, K. Falconer, and R. K. Guy, Unsolved problems in geometry: unsolved problems in intuitive mathematics. Springer Science & Business Media, 2012, vol. 2.

[74] X. Jin, S. Yin, X. Li, G. Zhao, Z. Tian, N. Sun, and S. Zhu, "Color image encryption in ycbcr space," in 2016 8th International Conference on Wireless Communications Signal Processing (WCSP), 2016, pp. 1–5.

[75] GeyserTimes, "Eruptions of Old Faithful Geyser, May 2014 [online database] ," https://geysertimes.org, 2017.

[76] S. Firdaus and M. A. Uddin, “A survey on clustering algorithms and complexity analysis,” International Journal of Computer Science Issues (IJCSI), vol. 12, no. 2, p. 62, 2015.

[77] S. Belongie, J. Malik, and J. Puzicha, “Shape context: A new descriptor for shape matching and object recognition,” Advances in neural information processing systems, vol. 13, pp. 831–837, 2000.

[78] Y. Gan, P. Premaratne, K. Han, and D.-S. Huang, Bio-Inspired Computing and Applications. Springer, 2012.

[79] H. P. Hsu, Schaum’s outlines of theory and problems of signals and systems. McGraw Hill, 1995.

[80] W. Miller and J. Lewis, “Dynamic state estimation in power systems,” IEEE Transactions on automatic control, vol. 16, no. 6, pp. 841–846, 1971.

[81] R. E. Kalman, “A new approach to linear filtering and prediction problems,” 1960.

[82] S. J. Julier and J. K. Uhlmann, “New extension of the kalman filter to nonlinear systems,” in Signal processing, sensor fusion, and target recognition VI, vol. 3068. International Society for Optics and Photonics, 1997, pp. 182–193.

[83] E. A. Wan and R. Van Der Merwe, “The unscented kalman filter for nonlinear estimation,” in Proceedings of the IEEE 2000 Adaptive Systems for Signal Processing, Communications, and Control Symposium (Cat. No. 00EX373). Ieee, 2000, pp. 153–158.

[84] S. Godsill, “Particle filtering: the first 25 years and beyond,” in ICASSP 2019 - 2019 IEEE International Conference on Acoustics, Speech and Signal Processing (ICASSP), 2019, pp. 7760–7764.

[85] B. Bytes. [Online]. Available: http://www.mathsmutt.co.uk/files/newt.htm

[86] M. Kristan, J. Matas, A. Leonardis, T. Vojir, R. Pflugfelder, G. Fernandez, G. Nebehay, F. Porikli, and L. Čehovin, “A novel performance evaluation methodology for single-target trackers,” IEEE Transactions on Pattern Analysis and Machine Intelligence, vol. 38, no. 11, pp. 2137–2155, Nov 2016.

[87] P. Dendorfer, H. Rezatofighi, A. Milan, J. Shi, D. Cremers, I. Reid, S. Roth, K. Schindler, and L. Leal-Taixé, “Mot20: A benchmark for multi object tracking in crowded scenes,” arXiv:2003.09003[cs], Mar. 2020, arXiv: 2003.09003. [Online]. Available: http://arxiv.org/abs/1906.04567

[88] Z. Ma, L. Wang, H. Zhang, W. Lu, and J. Yin, “Rpt: Learning point set representation for siamese visual tracking,” arXiv preprint arXiv:2008.03467, 2020.

[89] Z. Zhang and H. Peng, “Ocean: Object-aware anchor-free tracking,” arXiv preprint arXiv:2006.10721, 2020.

[90] B. Yan, X. Zhang, D. Wang, H. Lu, and X. Yang, “Alpha-refine: Boosting tracking performance by precise bounding box estimation,” arXiv preprint arXiv:2012.06815, 2020.

[91] B. Li, J. Yan, W. Wu, Z. Zhu, and X. Hu, "High performance visual tracking with siamese region proposal network," in 2018 IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2018, pp. 8971–8980.