Institute of Imaging and Computer Vision | RWTH AACHEN UNIVERSITY

Fast multi-object tracking in coded video sequences

Master Thesis

presented by

Mingtao Ou

Supervisor:
M.Sc Tim Claßen, M.Sc Haoming Zhang

Institute of Imaging & Computer Vision
Prof. Dr.-Ing. Johannes Stegmaier
RWTH Aachen University

Erklärung nach §18 Abs. 1 ÜPO

Hiermit versichere ich, dass ich die vorgelegte Master Thesis selbständig angefertigt habe. Es sind keine anderen als die angegebenen Quellen und Hilfsmittel benutzt worden. Zitate wurden kenntlich gemacht.

I hereby confirm that I have written this Master Thesis independently using no sources or aids other than those indicated. I have appropriately declared all citations.

Mingtao Ou
Aachen, 30.09.2023

Contents

List of Figures

List of Tables

Nomenclature

Vectors & Matrices

$\mathbf{K}$	partial derivatives matrix in x and y direction for one pixel window
$\mathbf{b}$	partial derivatives vector for pixel window
$\mathbf{v}$	optical flow vector
α	regularization constant
$\mathbf{p}$	transformation parameters
$\mathbf{W}$	warping matrix
E	error/ loss
d_p	prediction displacement
γ	threshold of Mahalanobis distance
$\boldsymbol{\mu}$	mean vector
$\boldsymbol{\Sigma}$	covariance matrix
$\mathbf{X}$	key points matrix in last frame
$\mathbf{X'}$	key points matrix in current frame
$\mathbf{R}$	rotation matrix
$\mathbf{t}$	translation vector
$\mathbf{X}_a$	affined key points matrix
$\mathbf{A}$	affine matrix
$\mathbf{r}$	residual vector
σ_I	normalization factor
$\mathbf{N}$	normalization matrix
$\mathbf{J}$	Jacobian matrix
$\mathbf{H}$	Hessian matrix
λ	regularization weight parameter
MSE	mean square error
$\mathbf{CM}$	cost matrix
pd	loss of distance
ls	loss of shape
lc	loss of color

Images

I	image
J	image
T	template image in Lucas-Kanade method
K	image
(V_x, V_y)	optical flow of one pixel
$[x, y]$	coordinates of a pixel within image
(m, n)	shape of the image in first 2 dimensions

1 Introduction

With the explosive growth of video usage, video coding is now playing a critical role. Modern video coding standards, e.g., H.264/AVC [3], H.265/HEVC [4], VP9 [5], AV1 [6], etc., all follow the classical block-based hybrid video coding framework. One particular characteristic of block-based coding is the accidental production of visible block structures in reconstructed frames due to the lack of inter-block correlation [7]. Compression artifacts have a dual impact, not only diminishing the visual quality of reconstructed frames but also undermining coding efficiency, as each reconstructed frame assumes the role of a reference for encoding subsequent frames. In each frame of coded video, edges are the regions most noticeably affected by encoding coding blocks. The same edge may be distributed across different coding blocks, resulting in distortion, interruptions, and discontinuities along the edge. To address this issue, an edge tracking algorithm needs to be developed.

1.1 Motivation

The rising demand for video data compression, driven by the surge in video traffic, has led to the development of efficient algorithms that reduce data while maintaining quality. Modern compression systems utilize dependencies withing and between frames in a process called prediction. The prediction signal is usually different from the ground truth signal. To compensate for prediction errors, a so-called residual signal is transmitted. However, this prediction signal is typically a rough approximation and may contain specific artifacts like blocking. To mitigate the impact of these artifacts, in-loop filtering is applied, which may either conceal typical artifacts or utilize additional information to enhance overall quality.

Video coding and video compression are closely related concepts that are often used together to achieve efficient video storage and transmission. Video coding involves the conversion of the original video signal into a digital format and the application of coding algorithms to reduce data size while maintaining image quality. For example, HEVC achieves more efficient video encoding through the flexible combination of Coding Tree Units (CTU) and Coding Units (CU), CTU and CU are commonly referred to as coding blocks. By selecting appropriate CTU and CU sizes, the encoder can adapt to various video content more effectively while reducing data volume without compromising image quality. This facilitates higher compression ratios, thereby reducing storage and transmission costs. The choice of CTU and CU sizes can have an impact on the edge regions of an image. Typically, it is possible to determine the optimal size by balancing considerations of image quality, compression efficiency, and computational complexity. Different video cod-

ing standards and application scenarios may adopt varying strategies for handling edge information. From Figure 1.1 and Figure 1.2, it is evident that smaller CTU

Figure 1.1: Coding $Qp = 27$, reducing the sizes of CTU and CU enhances the preservation of edge details within the image.

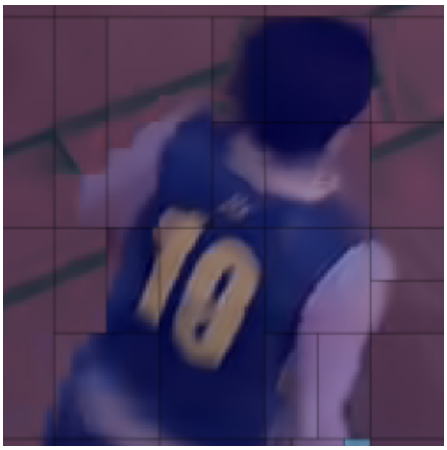

Figure 1.2: Coding $Qp = 42$, larger CTU and CU sizes result in a reduced retention of edge information within the image.

and CU sizes offer better preservation of edge information within images. As edges are typically composed of relatively fine details, smaller coding units provide higher precision in capturing these details, resulting in the retention of more edge information within the image. However, the use of smaller CTU and CU sizes may introduce increased computational complexity in the encoding process, as it involves the handling of a greater number of coding units. This may place certain demands on the performance of hardware encoders or decoders.

Conversely, larger CTU and CU sizes can significantly enhance video compression rates. However, when using larger sizes, the post-encoding images may exhibit more pronounced artifacts in the edge regions, as illustrated in Figure 1.3.

At the boundaries of coding blocks, noticeable blocking effects or artifacts may appear due to sudden changes in pixel averages. The blocking effect causes color at the edges of blocks to exhibit discontinuities, where colors suddenly change from one coding block to another without a smooth transition. This can result in visible boundary lines, making the image appear unnatural, a lot of artifacts are produced near the object edges. These situations can be observed in Figure 1.3. For low bitrate videos, this situation is very obvious and can be seen with the naked eye. There are usually two methods used to solve the problem of low video quality caused by blocking effect. First one is Intra-frame Prediction [8]. In video coding, intra-frame prediction can help reduce artifacts. This method uses pixel values within the current frame to predict pixel values within coding blocks, reducing encoding errors, the task that this paper needs to complete is to track and predict the edges in the video to optimize the video quality. The other method is the solution used in video decoding, namely Filtering Techniques [9]. At the decoding end, deblocking filters can be used to reduce blocky artifacts. These filters help smooth coding

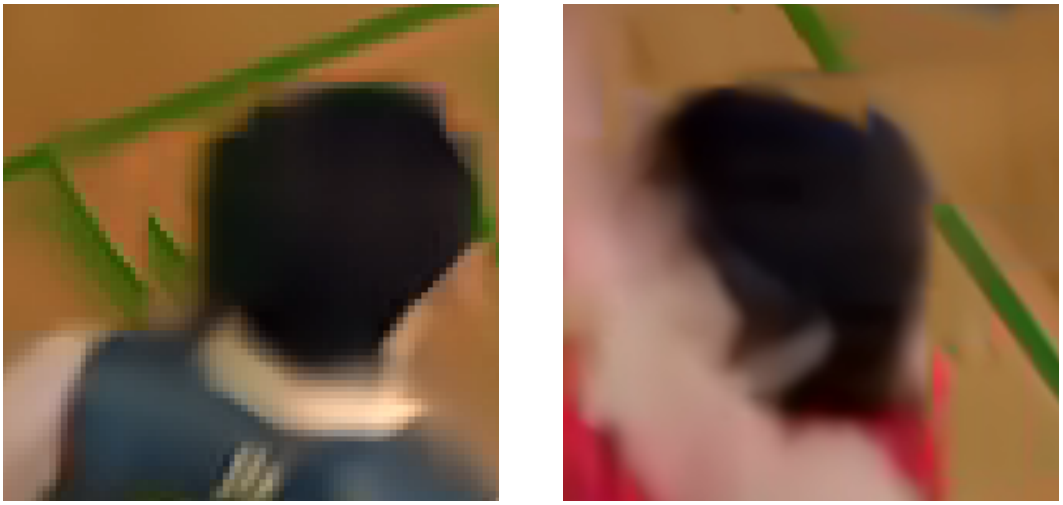

Figure 1.3: Distorted edges during video coding, $Qp = 42$

block boundaries to make them appear more natural. Video coding aims to strike a balance between compression efficiency and video quality, as such, it balances between maintaining a high compression ratio and providing a sufficiently high video quality.

In summary, during video encoding, the blocking effect has a great impact on the edges of objects in the video, and video distortion is often caused by irregular movement of the edges. The main focus of this thesis is the development of a robust edge tracking algorithm in the presence of coding artifacts. In previous video encoding systems, artifacts caused by the motion of edges in the video were not previously noticed. However, by tracking the motion of edges, some areas in the video that are distorted may appear smoother and less cluttered.

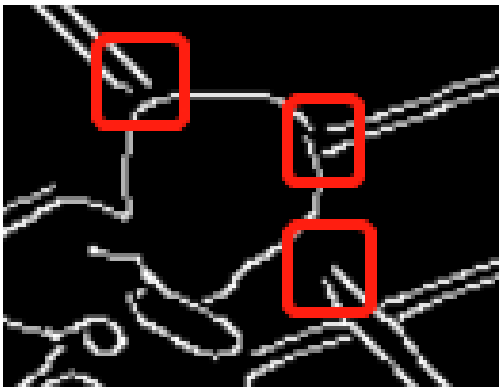

Figure 1.4: Original edges

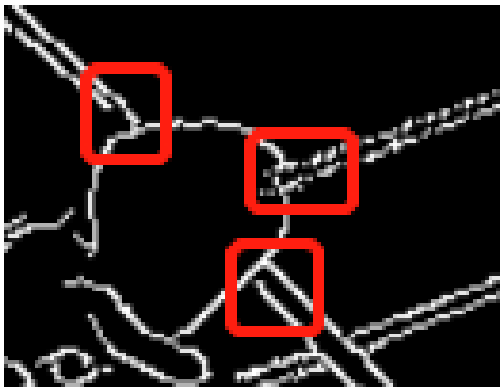

Figure 1.5: Tracked edges

Figures 1.4 through 1.7 display various results obtained in this paper. It can be seen that if the video quality is low, the edges detected in the original frame are missing and inaccurate, but because of the edge tracking algorithm, the edges of the current frame can be corrected based on the edge information in the previous frame image. Compared with previous research, the purpose of this paper is to

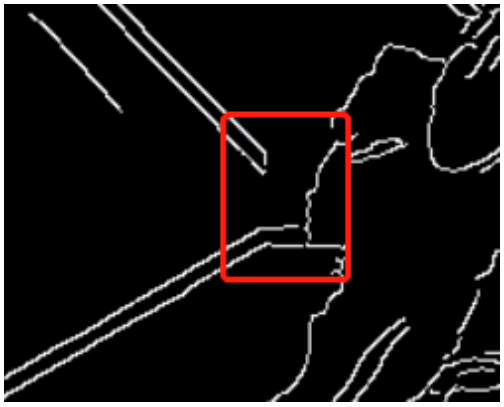

Figure 1.6: Original edges

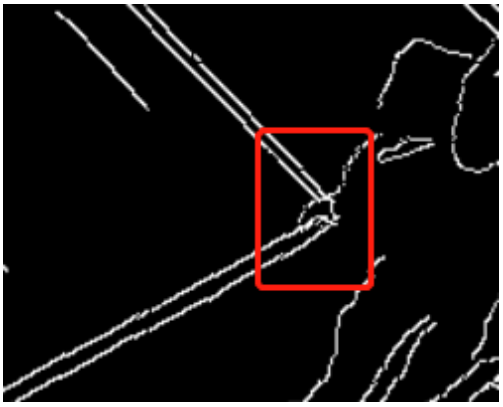

Figure 1.7: Tracked edges

develop a tracking algorithm suitable for different video qualities and evaluate the accuracy. At the same time, time complexity should be considered, in the context of video transmission, such as video conferencing or live streaming, there is a critical need for rapid video encoding and decoding. This scenario places a high demand on the time complexity of algorithms. One should contemplate employing a sparse tracking algorithm for object tracking instead of processing all pixels within the entire image. Tracking simple features that are suitable for video coding, such as edges, contours, textures, and key points. This task is challenging because coding artifacts may distort edges, and artifacts might be erroneously detected as edges. An edge tracking algorithm suitable for such scenarios is desired to provide information for other video processing steps, especially for the steps of Motion Estimation, Motion Compensation, Intra-frame Coding, and Inter-frame Coding [3] [4] [5] [6]. Consequently, tracking algorithms are evaluated and implemented to assess their suitability in this context.

1.2 Introduction to Paper Structure

This paper begins by tracking key points in the coded video, as key point tracking is a prerequisite for edge tracking. By analyzing the positional changes of key points between two frames, an approximation of the motion of edges around the key points, including rotation and translation, can be derived. Subsequently, an evaluation of the effectiveness of key point tracking is conducted, investigating the tracking errors of sparse key point tracking under different video quality conditions. This evaluation allows for the analysis of potential issues that may arise when using key point tracking in subsequent edge tracking. Furthermore, it provides a rough estimate of the impact of key point tracking on the accuracy of the subsequent edge tracking.

Key point tracking algorithms are utilized as the foundation for motion estimation of edges extracted from the previous frame in videos encoded at different quality levels. The motion of edges in the current frame is inferred from those in the previous frame. This inferred information is then compared with edges extrac-

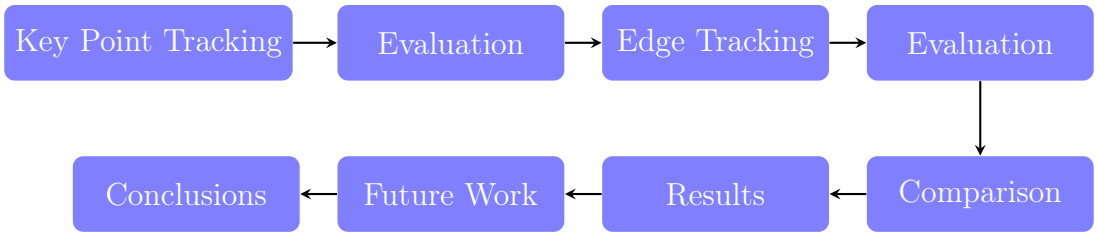

Figure 1.8: Overview of the paper's structure

ted from the uncoded video of the current frame. The disparities between these two sets of edges are computed to assess the algorithm's performance.

Furthermore, a comparative analysis is conducted between the obtained results and those produced by existing optical flow algorithms. This comparison serves to highlight the strengths and limitations of the proposed algorithm. Finally, the practical application and performance of this algorithm are demonstrated. Figure 1.8 serves as a structural abstract for the entire paper, providing a summarized overview of the paper's organization and content.

2 Theoretical Background

To facilitate edge tracking, this paper employs key point tracking as a prerequisite technique. Key point tracking is a vital task in the field of computer vision, commonly used for monitoring distinctive points in images or videos. Key points are points in images with unique characteristics and are typically employed to track the motion of objects or scenes across different frames. Theoretically, by obtaining the motion trajectories of key points distributed around edges, it becomes possible to approximate the motion of edges between two frames. Hence, this chapter's objective is to introduce the theoretical foundation of this method and evaluate its performance across videos of varying quality. For key point tracking, optical flow method is a very common usage [10].

2.1 Optical Flow

Sequences of ordered images allow the estimation of motion as either instantaneous image velocities or discrete image displacements. The optical flow methods try to calculate the motion between two image frames which are taken at times t and $t + \Delta t$ at every voxel position. These methods are called differential since they are based on local Taylor series approximations of the image signal; that is, they use partial derivatives with respect to the spatial and temporal coordinates. a voxel at location (x, y, t) with intensity $I(x, y, t)$ will moved by Δx, Δy and Δt between the two image frames, and the following brightness constancy constraint can be given:

$$I(x, y, t) = I(x + \Delta x, y + \Delta y, t + \Delta t). \tag{2.1}$$

Assuming the movement to be small, the image constraint at $I(x, y, t)$ with Taylor series can be developed to get:

$$I(x + \Delta x, y + \Delta y, t + \Delta t) = I(x, y, t) + \frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t. \tag{2.2}$$

By truncating the higher order terms (which performs a linearization) it follows that:

$$\frac{\partial I}{\partial x}\Delta x + \frac{\partial I}{\partial y}\Delta y + \frac{\partial I}{\partial t}\Delta t = 0, \tag{2.3}$$

or, dividing by Δt,

$$\frac{\partial I}{\partial x}\frac{\Delta x}{\Delta t} + \frac{\partial I}{\partial y}\frac{\Delta y}{\Delta t} + \frac{\partial I}{\partial t}\frac{\Delta t}{\Delta t} = 0, \tag{2.4}$$

which results in

$$\frac{\partial I}{\partial x}V_x + \frac{\partial I}{\partial y}V_y + \frac{\partial I}{\partial t} = 0. \tag{2.5}$$

Where V_x and V_y are the x and y components of the velocity or optical flow of $I(x, y, t)$ and $\frac{\partial I}{\partial x}$, $\frac{\partial I}{\partial y}$ and $\frac{\partial I}{\partial t}$ are the derivatives of the image at $(x, y, t$ in the corresponding directions. I_x, I_y and I_t can be written for the derivatives in the following. Thus:

$$I_x V_x + I_y V_y = -I_t, \tag{2.6}$$

This is an equation in two unknowns and cannot be solved as such. To find the optical flow another set of equations is needed, given by some additional constraint. All optical flow methods introduce additional conditions for estimating the actual flow. Nowadays, there exist numerous well-established optical flow algorithms [11]. The Lucas-Kanade [1] optical flow method is a classical sparse optical flow technique. It is based on the assumption of local brightness constancy and involves selecting a set of feature points in the image to calculate their displacements between adjacent frames. This method is suitable for tracking a small number of feature points. The Lucas–Kanade method assumes that the displacement of the image contents between two nearby instants (frames) is small and approximately constant within a neighborhood of the point p under consideration. Thus the optical flow equation can be assumed to hold for all pixels within a window centered at p. Namely, the local image flow (velocity) vector (V_x, V_y) must satisfy

$$\mathbf{K}\mathbf{v} = \mathbf{b} \tag{2.7}$$

, where

$$\mathbf{K} = \begin{bmatrix} I_x(q_1) & I_y(q_1) \\ I_x(q_2) & I_y(q_2) \\ \vdots & \vdots \\ I_x(q_n) & I_y(q_n) \end{bmatrix}, \qquad \mathbf{v} = \begin{bmatrix} V_x \\ V_y \end{bmatrix}, \qquad \mathbf{b} = \begin{bmatrix} -I_t(q_1) \\ -I_t(q_2) \\ \vdots \\ -I_t(q_n) \end{bmatrix},$$

$q_1, q_2, \cdots, q_n$ are the pixels inside the window, and $I_x(q_i), I_y(q_i), I_t(q_i)$ are the partial derivatives of the image I with respect to position x, y and time t, evaluated at the point q_i and at the current time. This system has more equations than unknowns and thus it is usually over-determined. The Lucas–Kanade method obtains a compromise solution by the least squares principle. Namely, it solves the 2×2 system

$$\mathbf{v} = (\mathbf{K}^T\mathbf{K})^{-1}\mathbf{K}^T\mathbf{b}. \tag{2.8}$$

The Horn-Schunck [12] optical flow method, on the other hand, is a global optical flow approach that not only considers feature points but also takes into account the entire optical flow field of the image. It relies on the assumption of optical flow smoothness and estimates the motion of all pixels in the image by optimizing a global optical flow field. This method is well-suited for smooth image regions but may not handle areas with weak texture. The Horn-Schunck algorithm assumes smoothness in the flow over the whole image. Thus, it tries to minimize distortions in flow and prefers solutions which show more smoothness. The flow is formulated as a global energy functional which is then sought to be minimized. This function

Table 2.1: Performance Comparison of Optical Flow Estimation Methods

Feature	**Farneback**	**Lucas-Kanade**	**Horn-Schunck**	**FlowNet2**
Algorithm Type	Optical Flow Estimation	Optical Flow Estimation	Optical Flow Estimation	Deep Learning Optical Flow Estimation
Speed	Suitable for real-time applications	Suitable for real-time applications	Suitable for real-time applications	Slower, GPU acceleration required
Applicability	Medium to small displacements and velocity changes	Small displacements and velocity changes	Small displacements and velocity changes	Large displacements and velocity changes
Robustness	Relatively robust to some noise and brightness changes	Relatively robust to some noise	Relatively robust to some noise	Sensitive to noise and changes
Accuracy	Suitable for general accuracy requirements	Suitable for general accuracy requirements	Suitable for general accuracy requirements	High accuracy, particularly for large displacements
Handling Non-Rigid Objects	Limited	Limited	Limited	Good
Computational Complexity	Low	Low	Low	High
Application Areas	Real-time motion detection, simple tracking	Real-time motion detection, simple tracking	Real-time motion detection, simple tracking	Complex optical flow estimation and computer vision tasks

is given for two-dimensional image streams as:

$$E = \iint [(I_x u + I_y v + I_t)^2 + \alpha^2(||\nabla u||^2 + ||\nabla v||^2)]d_x d_y. \tag{2.9}$$

Where I_x, I_y and I_t are the derivatives of the image intensity values along the x, y and time dimensions respectively, and the parameter α is a regularization constant. Larger values of α lead to a smoother flow.

Farneback [13] optical flow is a dense optical flow method designed to estimate the motion of every pixel in the entire image. It is based on polynomial expansion

and sparse feature point matching, allowing it to simultaneously estimate local and global motion. Farneback optical flow excels in handling images with rich textures and small to moderate motions. Table 2.1 is a summary table outlining the various optical flow methods mentioned earlier in the text. In recent years, deep learning techniques have made significant advancements in optical flow estimation. These methods utilize Convolutional Neural Networks (CNNs) such as FlowNet2 [2] to learn the task of optical flow estimation and can tackle various optical flow challenges, including occlusions, texture variations, and more. FlowNet2 is the second generation of the FlowNet series, a deep learning model designed for precise optical flow estimation, which calculates the motion of pixels between frames in images or videos. It leverages a deep convolutional neural network architecture and multi-scale processing to handle motion information at various scales. FlowNet2 is trained end-to-end, automatically learning the relationship between image features and motion without manual feature extraction.

Considering the tracking algorithm, the feature tracking algorithm with sparsity and low time complexity is the KLT(Kanade-Lucas-Tomasi) algorithm [14]. Compared with other optical flow algorithms (e.g., Farneback, and deep-learning based methods such as FlowNet2), The KLT algorithm offers several advantages. Firstly, it is highly computationally efficient, making it well-suited for real-time applications. It achieves this efficiency by estimating optical flow within local pixel neighborhoods, thereby reducing computational overhead. Secondly, KLT optical flow is known for its simplicity and stability compared to more complex optical flow algorithms, such as global optical flow. This simplicity facilitates easy implementation and deployment while also demonstrating robustness to noise and variations in brightness.KLT optical flow can be used to track sparse feature points in two frames, such as interest points or corners, this makes it particularly useful for applications that require tracking specific objects or points of interest. KLT optical flow can also be combined with other computer vision tracking techniques to enhance its performance. For instance, combining KLT optical flow with Kalman filters [15] [16] can better handle nonlinear dynamics and estimate target states.

Algorithm 1: KLT Algorithm

Input: $GrayImg1$, $GrayImg2$
Output: Optical flow: $d_1, \ldots, d_N$

$I \leftarrow GrayImg1$;
$J \leftarrow GrayImg2$;
$KeyPoints1 \leftarrow I$;
$KeyPoints2 \leftarrow J$;
for $Keypoint1$ *in* $Keypoints1$ **do**
    get $u = I(x, y)$;
    for $Keypoint2$ *in* $Keypoints2$ **do**
        get $v = J(x', y')$;
        find $d = \begin{bmatrix} dx & dy \end{bmatrix}^T$ for $v = u + d = \begin{bmatrix} u_x + d_x & u_y + d_y \end{bmatrix}^T$;
    end
end

The KLT algorithm can be summarized as Algorithm 1.
In Figure 2.1, there are two pixel blocks with center points $I(x, y)$ and $J(x', y')$,

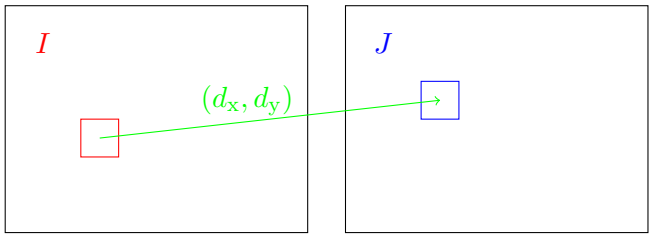

Figure 2.1: Template tracking, the red and blue windows represent the relevant pixel blocks in the two frames

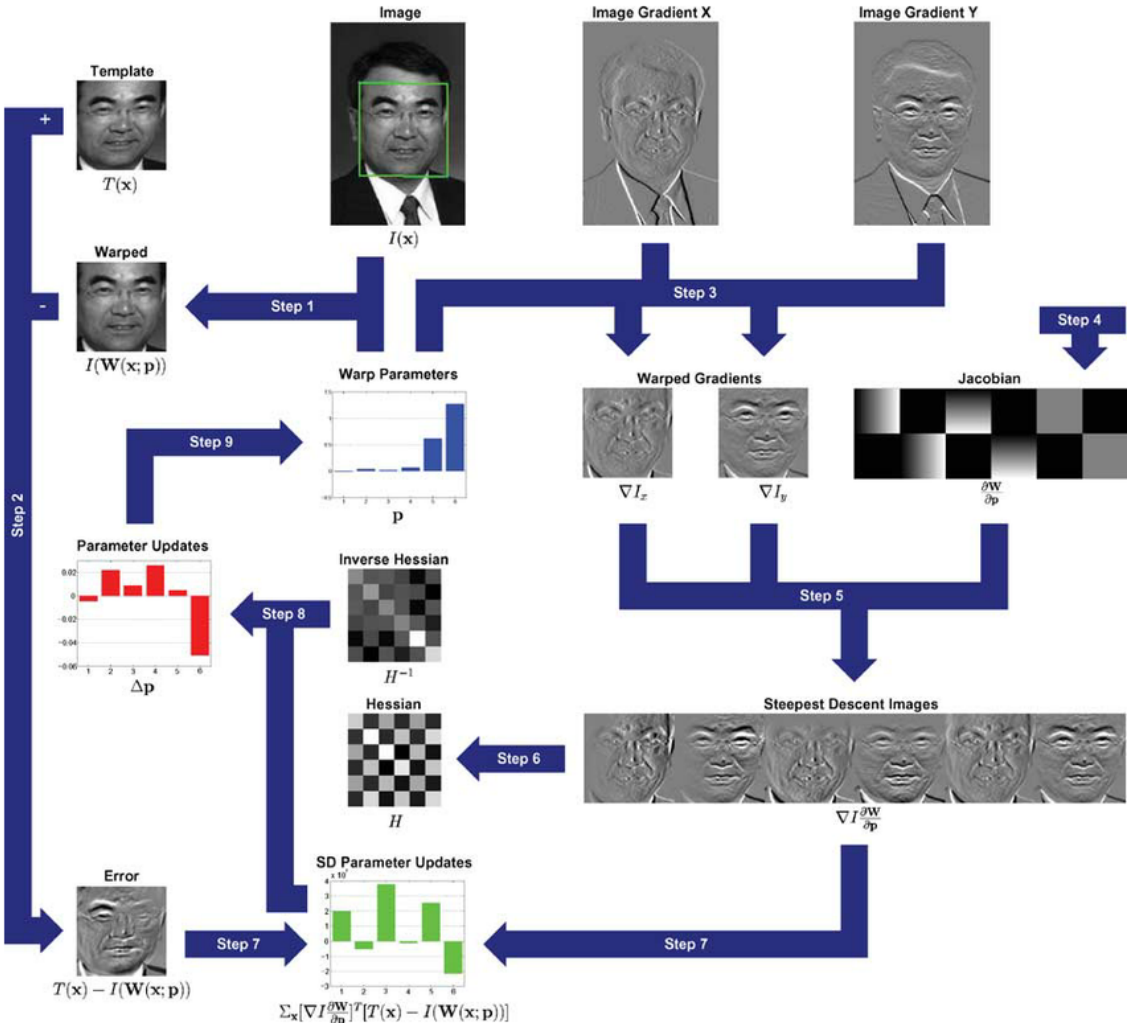

Figure 2.2: Lukas-Kanade template tracker [1], this is an iterative algorithm. Each iteration requires the calculation of 9 related steps to finally obtain the optimal warping parameters.

given two subsequent frames for the template tracking, use the pixel block in frame

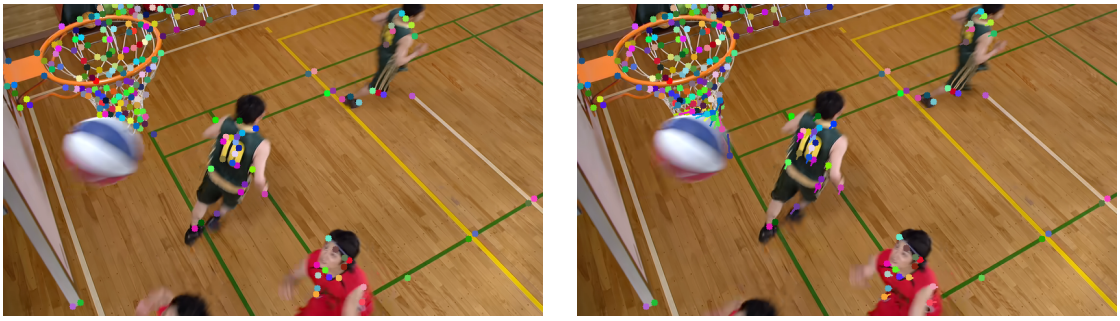

Figure 2.3: KLT tracking results, the key points of the same color on the two frames represent that they belong to the same point in the scene.

I as the template T, estimate the apparent motion field Δx and Δy between $I(x, y)$ and $J(x', y')$. Basic Lucas-Kanade [1] Derivation for Templates(T) can be calculated as

$$\sum_x E(u, v) = [I(x + d_x, y + d_y) - T(x, y)]^2. \tag{2.10}$$

The problem of calculating the displacement is typically transformed into a warp problem as shown in Figure 2.2. To estimate the warping parameters by LK algorithm. Through iterative calculation, once the warp parameter Δp was obtained after the loss falls below a certain threshold, Formula 2.10 will appear as follows:

$$\arg\min_{\Delta \mathbf{p}} \sum_x [I(\mathbf{W}(x; \mathbf{p} + \Delta \mathbf{p})) - T(x)]^2. \tag{2.11}$$

The KLT tracking algorithm can quickly and accurately obtain sparse optical flow, which are formed by the position changes of key points in two frames. The result of KLT tracker can be seen in figure 2.3. The purpose of performing object tracking is to obtain the object's position, displacement, speed, and even rotation. This enables the prediction of the object's future location in subsequent frames or the inference of its intermediate position between two frames, effectively interpolating frames. Since predicting the motion of the key points is desired, consideration was given to using the Kalman filter for optimizing key point trajectory tracking. For example, Bukey and his group [17] successfully applied Kalman filter to their multi-target tracking tasks.

2.2 Kalman Filter

Theoretically, Kalman filter plays a pivotal role in multi-object tracking [18]. It efficiently fuses information from multiple sensors or sources, providing precise estimations of object positions and velocities, thereby enhancing the robustness and accuracy of multi-object tracking systems. The state estimation and covariance matrix updating processes of Kalman filter enable the system to handle interactions and uncertainties among objects, making multi-object tracking more reliable and stable. Therefore, Kalman filter stands as a core technology in the field of

multi-object tracking, providing robust support for real-time objects tracking [19]. Since there are a lot of artifacts and noise in the coded video, the consideration was made to use Kalman filter in order to improve tracking accuracy. Its adaptability to dynamic and noisy environments makes it a valuable tool in tracking multiple objects with varying speeds and directions. Furthermore, Kalman filter can handle missing or noisy measurements gracefully, making it particularly suitable for scenarios where sensor data may be intermittent or corrupted. This may increase the robustness of the entire key point tracking algorithm to noise, and simultaneously obtain the prediction of the next frame position and the corrected position of the current frame. Here is the result of introducing the Kalman filter into KLT tracker. Figure 2.4 represents the tracking result of KLT combined with the Kalman filter

Figure 2.4: Key points tracking combine KLT with kalman filter, the points of different colors represent prediction, measurement, correction

.

after three frames. The red points are predictions for the next frame, the blue points are points corrected based on the predictions of the previous frame and the measurements of the current frame, and the green points are the measurements of current frame.

2.3 Key Point Tracking Evaluation

To perform key points tracking, the tracking effect is evaluated for different video qualities. To understand the effect of optical flow tracking under different video quality conditions, the original uncoded video needs to be used for one-by-one comparison, that is, key points are extracted from the first frame of the uncoded video

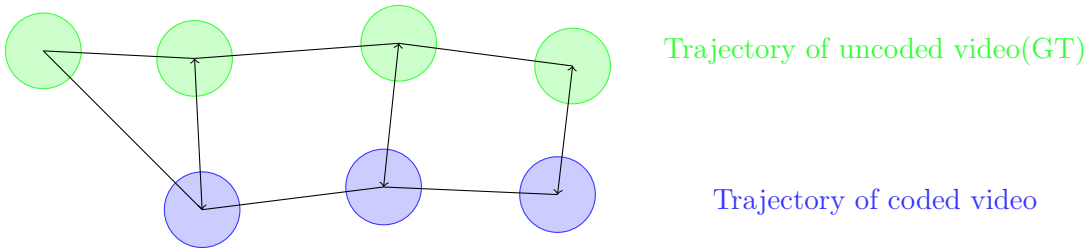

Figure 2.5: Measurement comparison method

and are then used for tracking in coded videos of varying qualities. The tracking results (key point positions) of each frame of each quality video are compared with the key point positions of the original video. The comparison standard is their Euclidean distance on the image.

In Figure 2.5, the key points are extracted from the coded and uncoded (ground truth) videos, and key points at the same position are tracked to compare the key point tracking errors under different qualities. This key point tracking algorithm was tested on multiple datasets and the loss was averaged for them. Different colors represent videos of different quality. The larger the number, the worse the video quality.

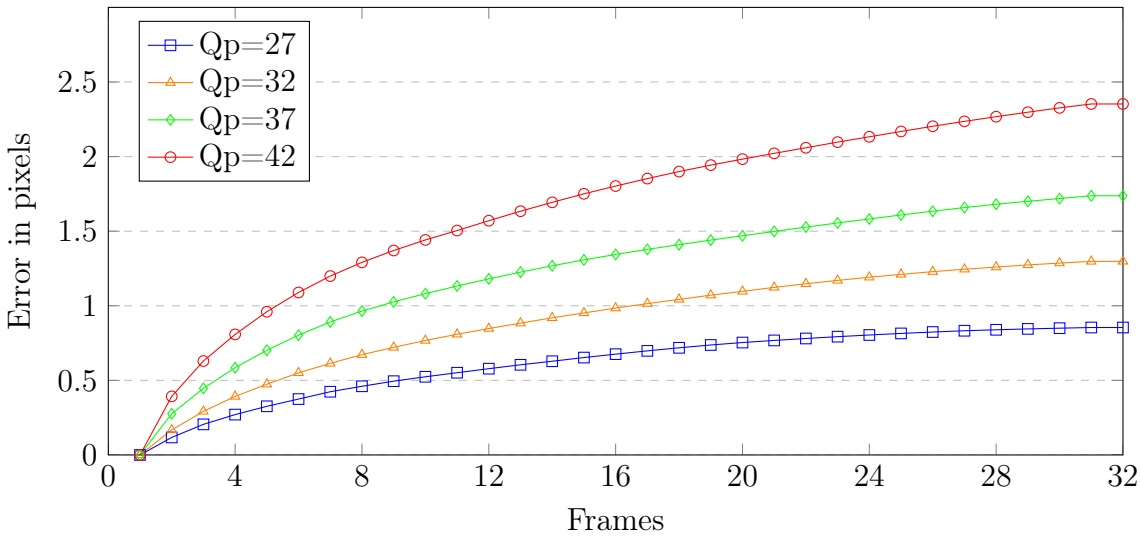

Figure 2.6: Key points distance between uncoded videos and coded videos

It can be seen from the curves in Figure 2.6 that in the first frame, the error is zero because the key points are extracted from the ground truth of the first frame. In videos with poorer quality, the error between coded video and the ground truth is larger, and this error propagates as the tracking step increases. This observation underscores the inherent challenge posed by poorer video quality in maintaining the accuracy and consistency of key point tracking over time. Consequently, it underscores the critical importance of addressing video quality issues in multi-frame

tracking applications to ensure reliable and precise tracking outcomes.In the worst-quality videos, the error reaches more than two pixels apart. In high-resolution videos, it may not have much impact, but for low-resolution videos, it may be an error visible to the naked eye. The results of tracking in the worst quality video and tracking in the uncoded video can be seen in Figure 2.7.

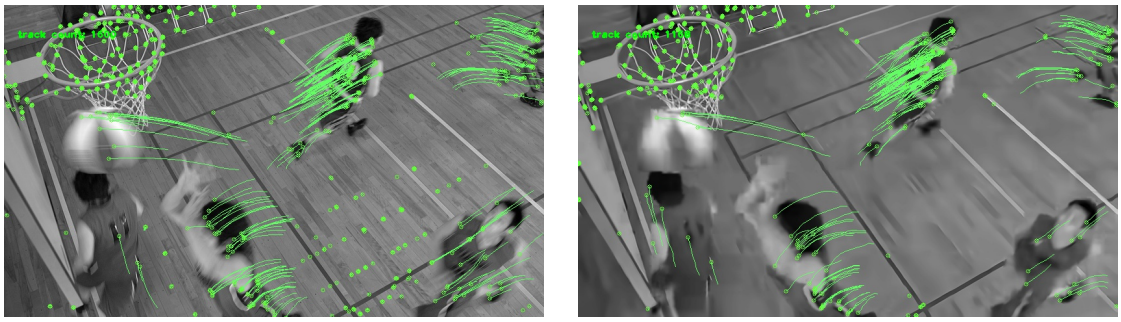

Figure 2.7: Tracking results for uncoded and coded videos

It shows that in the uncoded video, many edges, contours and corners are not distorted, except for some motion blurriness, the effect of key point tracking using KLT is very good, and the tracking trajectories are also smoother. However, for videos with reduced quality and distortion after encoding, it can be clearly seen during tracking that due to the influence of artifacts and macroblocks, fewer key points are detected than in uncoded videos, and during the tracking process, the trajectories of key points are chaotic, noisy, zigzag, and even merge together. After evaluating the measurement accuracy, the next step involves predicting the location of key points in the subsequent video frame. Following this prediction, the distances between these predicted key points and the ground truth will be compared. In object tracking, predicting the displacement of an object typically involves estimating the position at the third frame based on observations from the previous two frames as shown in Figure 2.8.

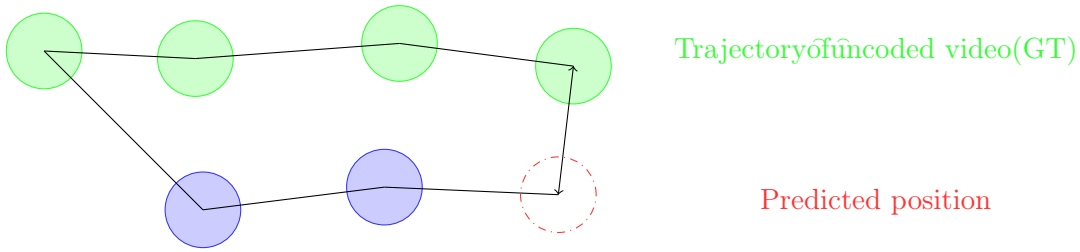

Figure 2.8: Prediction comparison method

This can be achieved through two common methods: the first order motion approximation method and the second order motion approximation method.

The first order motion approximation method assumes that the object moves at a constant speed between two consecutive time steps. It is a simplified model suitable for certain scenarios, especially when the object's speed remains relatively

constant over a short duration. The prediction of displacement can be calculated using the following formula:

$$d_{\mathrm{p}} = v_{\mathrm{cur}} \cdot \Delta t \tag{2.12}$$

Where: d_{p} is the predicted displacement for next frame, x_{cur} is the known position at the current time step. v_{cur} is the velocity estimate from the current frame and last frame. Δt is the time interval between two time steps.

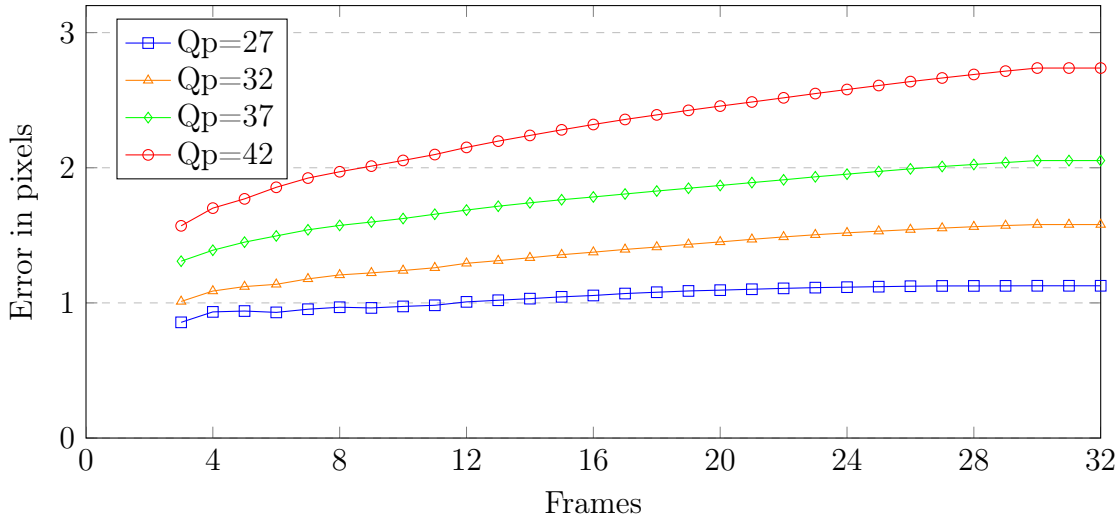

Figure 2.9: The average errors predicted in each frame using the first-order approximation method under various video qualities across multiple videos.

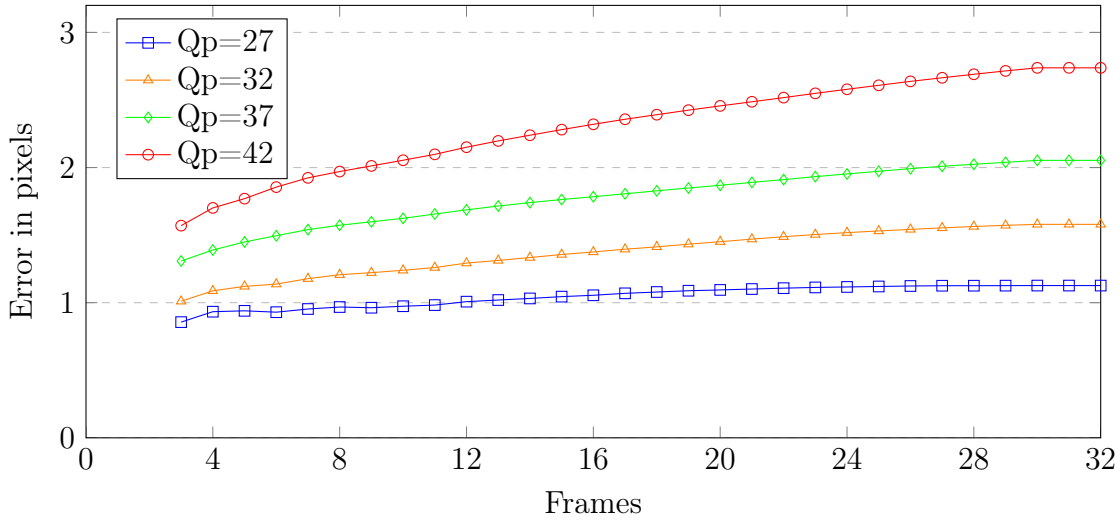

Figure 2.10: The average errors predicted in each frame using the second order approximation method under various video qualities across multiple videos.

The second order motion approximation method is more complex and takes into account changes in velocity between two time steps. This model is more suitable

for scenarios where the object experiences acceleration or variable speeds. The prediction of displacement can be calculated using the following formula:

$$d_{\mathrm{p}} = v_{\mathrm{cur}} \cdot \Delta t + \frac{1}{2}(a_{\mathrm{cur}} \cdot \Delta t)^2 \tag{2.13}$$

a_{cur} is the acceleration estimate from the current frame and last frame.

The results in Figure 2.9 and Figure 2.10 are shown in the above graphs. It can be seen that there is no significant difference in the results of the two methods. And the accuracy of prediction is about one pixel in the best video quality. In the worst video quality, the average error reaches 2.5 pixels as the time step increases. The worst tracking average error does not exceed 3 pixels.

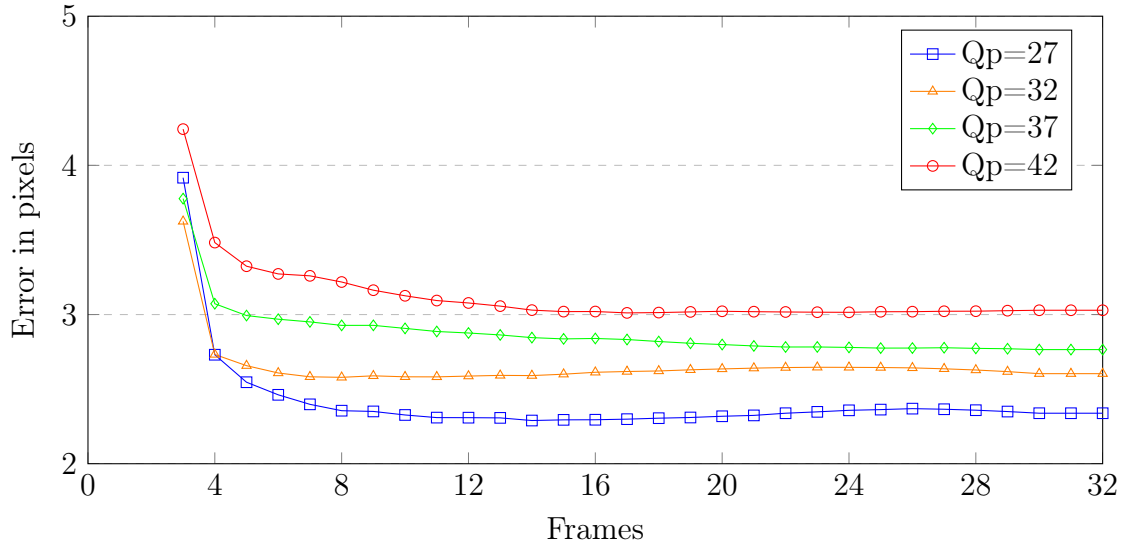

Figure 2.11: Error of predictions by Kalman filter in Basketball drill

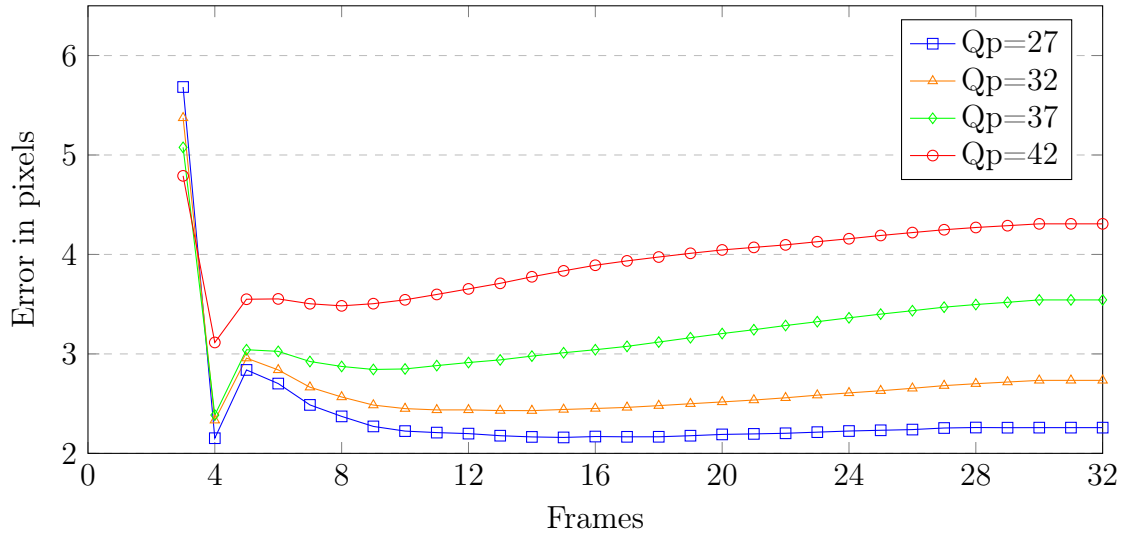

Figure 2.12: Error of predictions by Kalman filter in Daylight road 2(4K)

Let's take a look at the tracking results of the Kalman filter. It can be seen from Figure 2.11 and Figure 2.12 that the prediction of first 2 frames, the prediction has a large uncertainty, thus the error is very large. Although the error has decreased after that, compared with the second order motion approximation method and linear first order motion approximation method mentioned above, the effect not significantly improved, or even worse. Because measurement uncertainty and prediction uncertainty are difficult to set, it turns out that the Kalman filter performs mediocrely at this task. Considering other properties of the Kalman filter, for example, we have to apply the Kalman filter once to each tracked key point, resulting in relatively high computational complexity. Therefore, this method is not planned to be continued in subsequent work.

On the other hand, the problem of trajectory disappearance also needs to be considered. This problem can be attributed to various reasons, such as:

1. Object Occlusion: When an object is occluded by other objects or obstacles, the tracker may lose sight of the object temporarily, leading to a trajectory interruption.

2. Object Disappearance: Objects may leave the camera's field of view or disappear from view for various reasons. In such cases, the tracker cannot reacquire the target, resulting in trajectory interruption.

3. Variations in Lighting Conditions: Changes in lighting conditions, such as strong shadows, low light, or intense illumination, can challenge the tracker's ability to maintain tracking accuracy.

Figure 2.13 and Figure 2.14 are statistics of lost trajectories in 2 videos. Many reasons similar to those mentioned above are inevitable when striving to achieve multi-object tracking. Statistical research on missing trajectories has also been conducted. In the basketball drill video, numerous moving objects are present, including running individuals and basketballs, Due to their high speed motion, there will also be some motion blur, tracking them using the KLT optical flow method becomes particularly challenging.

Moreover, as video quality degrades, key points are increasingly affected by artifacts, making them harder to track. Therefore, in this video, the KLT algorithm caused many missing trajectories during tracking. For video Cactus, the camera is stationary, the movement is relatively slow, so there are not many missing tracks. Therefore, the three major assumptions of the optical flow algorithm are reviewed:

1. Brightness constancy: projection of the same pixel looks the same in every frame.

2. Small motion: pixels do not move very far.

3. Spatial coherence: pixels move like their neighbors.

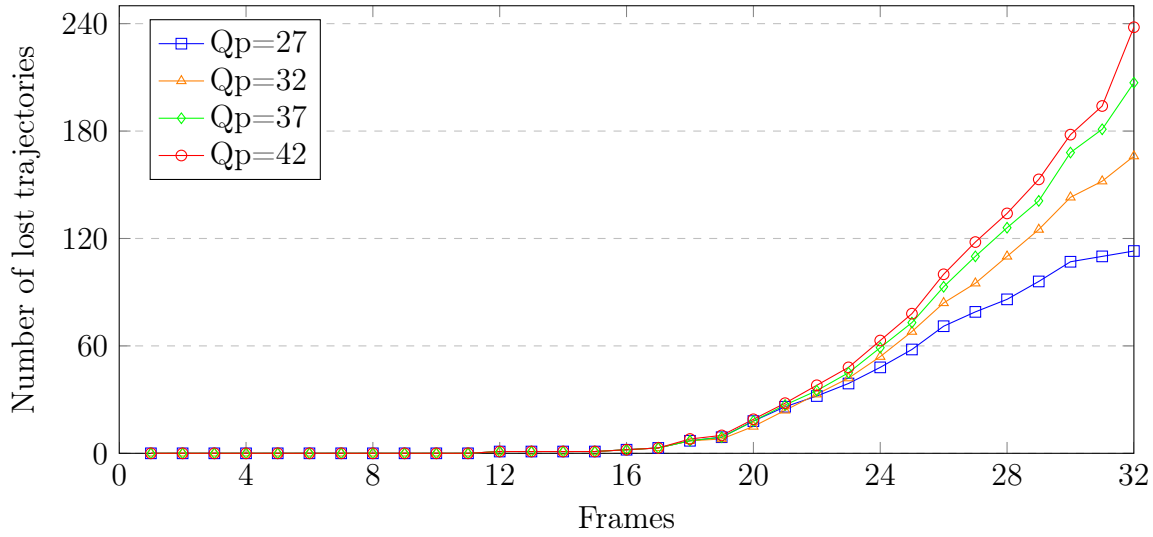

Figure 2.13: Lost trajectories in Basketball drill video, total trajectories=350

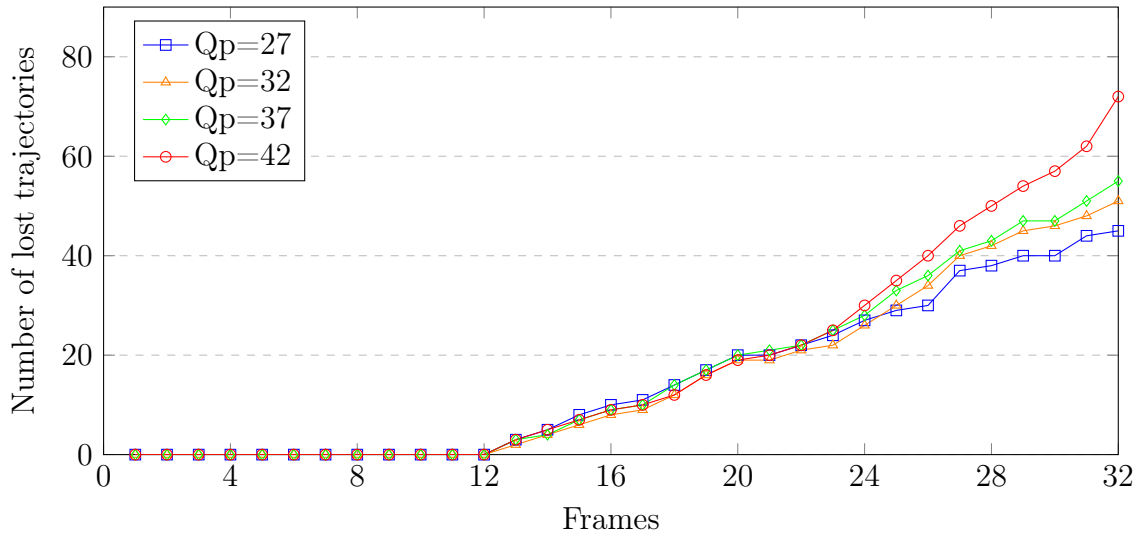

Figure 2.14: Lost trajectories in Cactus video, total trajectories=350

These assumptions cannot be totally met for all videos, so the loss of some trajectories is inevitable. Therefore, new key points are detected every few frames or even every frame to enhance the robustness of subsequent work.
After summarizing, the KLT tracking method mainly has the following advantages and disadvantages:
Pros:

1. High Precision: The KLT algorithm is a pixel-level tracking method, offering high tracking accuracy, especially if objects have good texture and stable lighting conditions. For low-quality videos, the maximum average distance tracking error is only about 2.5 pixel errors, which is acceptable.

2. Relatively Fast: Compared to some complex tracking algorithms, the KLT

algorithm typically operates at a faster speed, suitable for real-time or high frame rate applications.

3. KLT can be used for various types of objects, including natural scenes, computer vision feature points, and medical images, among others. This also works for videos of different scenes we want to track.

Cons:

1. Sensitivity to Lighting and Occlusion: The KLT algorithm is sensitive to changes in lighting conditions and occlusion of the target, which can lead to tracking failures in unstable lighting or when the target is occluded.

2. Failure Propagation: If the KLT algorithm encounters a tracking failure in one frame, this error may propagate to subsequent frames, causing prolonged tracking interruptions.

3 Edge Tracking Development

In Chapter 2, the key point tracking algorithm was introduced and evaluated. Next, the results of this key point tracking will be used to track edges because it is known that key points are concentrated on edges [20]. If the motion of key points is known, then theoretically, the motion of the edges, including translation and rotation, can be inferred. Edges are areas of the largest contrast, typically experiencing the most significant coding errors. Consequently, an edge tracking method might be employed to enhance coding through post-processing or in the video content prediction process.

There are many ways to predict the motion of pixels in an image. Translation is a basic image transformation that involves horizontal translation (dx) and vertical translation (dy). It relocates the edge without altering its shape or angle.

Affine transformation [21] is a linear transformation encompassing translation, rotation, scaling, and shearing. It's controlled by six parameters: horizontal and vertical translation (dx, dy), rotation angle (θ), horizontal and vertical scaling factors (sx, sy), and horizontal and vertical shear factors (shx, shy). Affine transformation preserves the parallelism of lines but doesn't maintain all distance and angle relationships.

Perspective transformation [22] is a non-linear transformation used for handling images under perspective projection. Typically, it's governed by eight parameters, involving coordinates of four pairs of points (four points on the source image and four points on the target image). Perspective transformation corrects image distortions caused by the camera's perspective, making distant objects appear smaller and closer objects appear larger.

3D rotation [23] is primarily employed in three-dimensional scenes, encompassing object rotation, translation, and scaling. This transformation relies on multiple parameters, including the rotation axis, rotation angle, and translation vector.

Edge tracking requires knowledge of the rotation and translation parameters for each edge between two frames. These parameters are encapsulated in the affine and perspective methods, as defined by Formula 3 and Formula 3.2. To determine these parameters, a minimum of 3 or 4 corresponding key points is required. For edge transformation, this study utilizes affine transformations.

$$\mathbf{W}([x,y];\mathbf{p}) = \begin{bmatrix} x + p_1 x + p_3 y + p_5 \\ y + p_2 x + p_4 y + p_6 \end{bmatrix} = \begin{bmatrix} 1+p_1 & p_3 & p_5 \\ p_2 & 1+p_4 & p_6 \end{bmatrix} \begin{bmatrix} x \\ y \\ 1 \end{bmatrix}, \tag{3.1}$$

$$\mathbf{W}([x,y];\mathbf{p}) = \frac{1}{p_7 x + p_8 y + 1} \begin{bmatrix} x + p_1 x + p_3 y + p_5 \\ y + p_2 x + p_4 y + p_6 \end{bmatrix}. \tag{3.2}$$

Where $p_1, p_2, \cdots, p_8$ are the transformation parameters of affine and perspective methods, $[x, y]$ are the pixel coordinates and $\mathbf{W}([x, y]; \mathbf{p})$ represent the transformation of pixel coordinates $[x, y]$ using warping parameters denoted as $\mathbf{p}$.

3.1 Extract the Edges

In order to track edges, Canny Edge method could be used to detect edges in images. The Canny Edge algorithm [24] can be summarized into Algorithm 2

Algorithm 2: Canny Edge Detection

Function `CannyEdgeDetection(`I`):`
  Data: Input image I
  Result: Binary edge map E
  $K_g \leftarrow$ CreateGaussianKernel(GaussianSize, GaussianSigma);
  $I_g \leftarrow$ Convolve(I, K_g) ; `// Apply Gaussian smoothing`
  ;
  $G_m, G_o \leftarrow$ CalculateGradient(I_g) ; `// Calculate gradient magnitude and orientation`
  ;
  $N_m \leftarrow$ NonMaximumSuppression(G_m, G_o) ; `// Non-maximum suppression`
  ;
  $E \leftarrow$ HysteresisThresholding$(N_m,$ LowThreshold, HighThreshold$)$; `// Edge tracing by hysteresis`
  ;
  return E ; `// Binary edge map`
  ;

A detailed explanation can be divided into the following four steps:

1. Filter image with derivative of Gaussian.

2. Find magnitude and orientation of gradient.

3. Non-maximum suppression:
   Thin multi-pixel wide "ridges" down to single pixel width.

4. Linking and thresholding (hysteresis):
   Define two thresholds: low and high, Use the high threshold to start edge curves and the low threshold to continue them, two thresholds sre k_{high} and k_{low} in cv2 library, we will set it manually, Typical ratio of thresholds is roughly $k_{high}/k_{low} = 2$.

In order to detect more edges and reduce noise at the same time, before using Canny edge method, a 3×3 Gaussian kernel is proposed to be used to blur the image., and then set $k_{low} = 50$, $k_{high} = 100$. The result of the Canny edge method used in 2 different videos can be seen in figure 3.1.

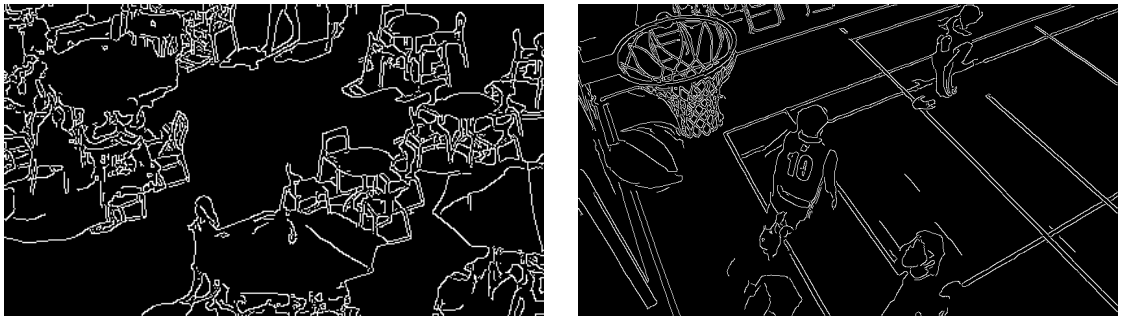

Figure 3.1: Extracted edges

3.2 Build Edge Set

After getting the edges in the image, because each edge moves differently in the image, each individual edge needs to be numbered, and these edges are added to an edge set, denoted as $\Omega = \left\{E_1^{(t)}, E_2^{(t)}, ..., E_n^{(t)}\right\}$. To do this, the function[1] from OpenCV library will be used to distinguish and number each edge in the image. This function divides the pixels in the input image into different connected components and assigns a unique label to each component. This helps us identify and segment different objects or regions in images. Often, the returned labeled image can be used for further analysis and processing of different components in the image. Then, the results like figure 3.2 can be obtained. From figure 3.2, the different edges can

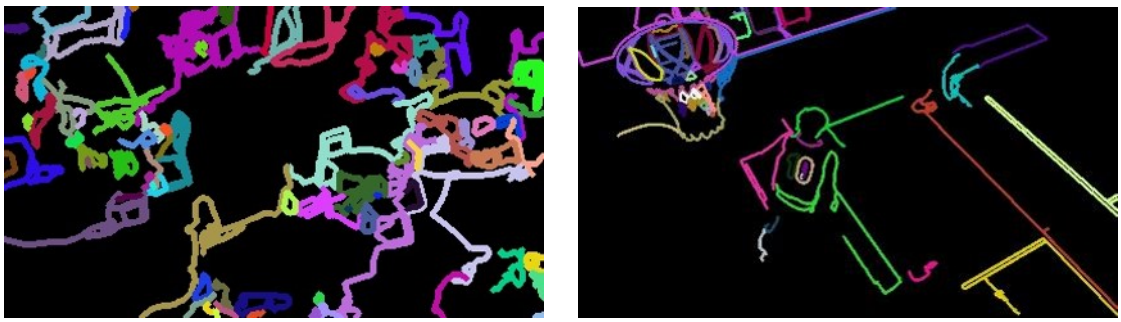

Figure 3.2: Visualization of different edges

be observed. The edges of different colors are independent edges, so that we can estimate the motion of the edge based on the motion of the key points around each edge.

3.3 Key Point Tracking

Now that the set of edges has been obtained, the KLT algorithm is then used to obtain the corresponding key points in the two frames of the video. As shown in Figure

[1] cv2.connectedComponents()

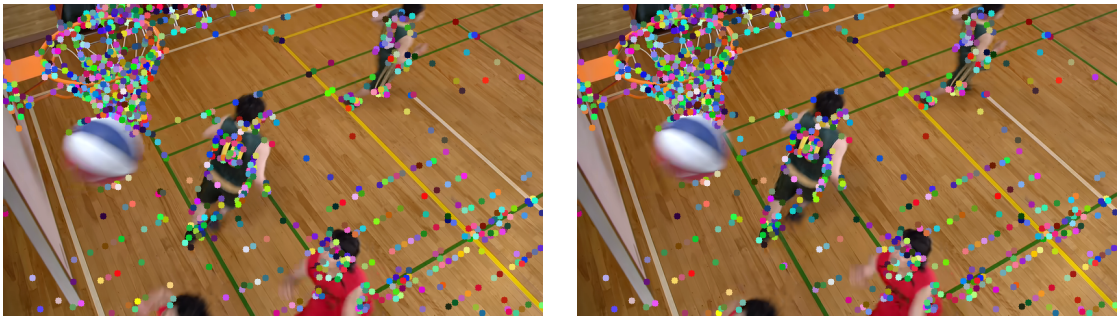

Figure 3.3: Corresponding key points between 2 frames

3.3, the corresponding key points in two grayscale image frames $\left\{x_1^{(t)}, x_2^{(t)}, \cdots, x_m^{(t)}\right\}$ and $\left\{x_1^{(t+1)}, x_2^{(t+1)}, \cdots, x_m^{(t+1)}\right\}$ are detected by KLT.

3.4 Build Edge Set

Theoretically, key points are concentrated around these edges. To achieve this goal, two methods were considered for assigning these key points to each edge. The first method is to set a validation volume for each edge, if the key point falls into this validation volume, it proves that the key point belongs to this edge. The second method is to directly extract key points from the edges image and then use the KLT algorithm to find the key points on the two frames of edges images, and then determine which edge a certain key point falls on.

3.4.1 Key Point Assignment by Validation Volume

Inspired by common multi-object tracking algorithms [25], to establish a Gaussian distribution for each edge, and the corresponding Mahalanobis distance is obtained as a validation volume. Each edge is represented by a two-dimensional array. The points contained in the array are the coordinates of each point on the edge on the image. With this two-dimensional array, the mean and variance of the Gaussian distribution can be calculated. The x-coordinates and y-coordinates of the two-dimensional points are separately averaged to obtain the mean of the Gaussian distribution. The square of the distance from all points to the mean is calculated, and then the squared distances of the x and y coordinates are averaged separately to obtain the variance of the Gaussian distribution. After getting the Gaussian of every edge, a validation volume by Mahalanobis distance could be obtained from

$$V(\gamma) = \left\{x_j^{(t)} | (\boldsymbol{\mu}_i^{(t)} - x_j^{(t)})^T \boldsymbol{\Sigma}_i^{-1} (\boldsymbol{\mu}_i^{(t)} - x_j^{(t)}) \leq \gamma\right\}. \tag{3.3}$$

Where x_i is the key point, μ_j and Σ_j are the mean, $V(\sigma)$ is the inside key points set for each edge, and covariance of the edge. If the Mahalanobis distance between the edge and the key point is smaller than a certain threshold, denoted as γ, then it is determined that this key point belongs to this edge. The threshold is determined

using the χ^2 table. The Mahalanobis distance is χ^2 distributed with the number of degrees of freedom n_z equal to the dimension of input data x. For a given probability bound, the corresponding threshold on the Mahalanobis distance can be obtained from χ^2 distribution tables. This threshold determines how many key points will be filtered. If the threshold is set smaller, only a few key points will be assigned to each edge, resulting in more accurate tracking of the edge, but due to the small number of key points, tracking an outlier among them may result in a significant error, which will reduce the robustness. Moreover, the affine transformation is used in the edge tracking method, and at least three points are required, so that edges are assigned with less than three key points are filtered, which results in the tracked edges being sparse. However, if the threshold is set larger, edges tracking will be easily affected by the movement of key points far away from the edge, resulting in inaccurate tracking results, however, since there are more key points assigned to the edges, the robustness of the results can be increased when calculating the affine matrix. A better trade off parameter γ should be found. After many attempts, it was found that setting this threshold to 0.95 yields better results.

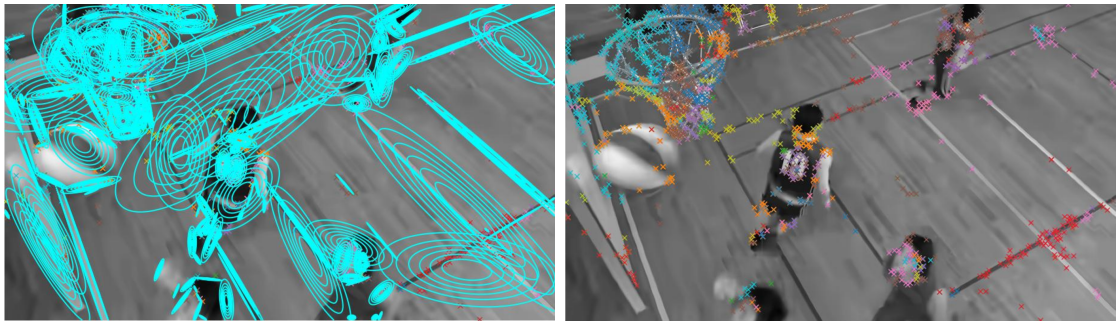

Figure 3.4: Validation volume of key points, in the left image, the blue contours represent Gaussian distribution for each edge in the previous frame. On the right, the image displays the result of key point matching between the current frame and the previous frame. Key points of the same color are assigned to the corresponding edge, ensuring that key points with similar colors are allocated to the same edge.

The results of this method are shown in Figure 3.4. The left picture shows the Gaussian distribution of each edge obtained in the first frame. The second picture shows the corresponding key points in the second frame. Key points of different colors belong to different edges. Of course, there will be repetitions, key points assigned to one edge may be reused on multiple edges.

3.4.2 Key Point Assignment by Distance From Point to Edge

In this method, key points are not actively sought and tracked from the input grayscale images. Instead, key points are identified from the edge images obtained through the Canny edge algorithm and tracked within the input image. After obtaining the key points, although key points are detected on the edges image, it cannot be completely guaranteed that the center of the key point is on the edge.

The distance from the key point x to all edges should be measured. If the distance is less than 1∼2 pixels, it is determined that the key point belongs to this edge. This method seems simpler, but the calculation complexity is higher than that of method 1, because each coordinate point p on an edge will be compared with the global key point by Euclidean distance $d = \sqrt{(x_i - p_j)^2}$. As shown in Figure 3.5,

Figure 3.5: Key points assignment, in the diagram, the origin represents a key point. If the color of the key point matches the color of an edge, it indicates that the key point has been assigned to that particular edge.

the dots are key points. Points of different colors are located on different edges. If the color of the key point is the same as the edge, it means that the key point belongs to this edge. For both methods, the number of Shi-Thomasi corner key points can be controlled by adjusting the parameters of Tomasi key point detection function[2] and the threshold of validation volume or Euclidean distance. To track more shorter edges or achieve better results, more key points are preferred, but this will also lead to increase computations.

3.5 Warp Edges

In order to infer the motion of each edge, the motion of the points on each edge in two frames needs to be used. The affine transformation matrix can be obtained based on the motion of the key points on each edge using the least squares method. Now, key point sets $\mathbf{X}$ and $\mathbf{X}'$ are available. The number of points contained in the $\mathbf{X}$ and $\mathbf{X}'$ matrices must be greater than or equal to 3, the rotation parameters $\mathbf{R}$

[2] cv2.goodFeaturesToTrack()

and translation parameters **b** will be estimated, $\mathbf{X}' = \mathbf{R}\mathbf{X} + \mathbf{s}$ and this equals to:

$$\begin{bmatrix} x_1 & y_1 & 1 \\ x_2 & y_2 & 1 \\ \vdots & \vdots & \vdots \\ x_i & y_i & 1 \end{bmatrix} \begin{bmatrix} r_{11} & r_{12} \\ r_{21} & r_{22} \\ s_1 & s_2 \end{bmatrix} = \begin{bmatrix} x'_1 & y'_1 \\ x'_2 & y'_2 \\ \vdots & \vdots \\ x'_i & y'_i \end{bmatrix} \tag{3.4}$$

Methods for evaluating affine transformation matrices have been provided in the Open-CV library, including partial affine function[3], it used to estimate partial affine transformations, usually transformations that include translation and rotation. Suitable for two-dimensional points that include translation, rotation, and scaling transformation estimation between key points pairs. And total affine function[4] is used to estimate the complete affine transformation, including translation, rotation, scaling and shearing. Suitable for between pairs of 2D points including translation, rotation, scaling and shearing transformation estimate. For two consecutive frames in the video, the objects and edges in the image will not experience much deformation, displacement, or rotation, so the first method is chosen to calculate the affine transformation parameters. After experiments, it is indeed The parameters obtained by the first method are more accurate.

When predicting the parameters of affine transformation, due to the impact of low video quality and artifacts, some key point matching is not accurate, resulting in the predicted optical flow vector being wrong. Therefore, if there are more elements in the matrix **X** or **X'**, which are the key points used to calculate the affine transformation, theoretically, the impact of outliers can be reduced and the robustness of the algorithm can be improved. After affine transformation matrix is obtained, the edges could be warped, that is, project the edges in the edges image of the first frame to the second frame with affine transformation, the results in Figure 3.6 and Figure 3.7 are obtained. This is the outcome of calculations using two methods proposed in Section 3.3. From Figures 3.6 and Figure 3.7, it can be observed that the blue edges represent the edges of the previous frame, the green edges are the edges inferred after affine transformation based on the movement of the key points, and the green arrows are the motion vectors between the corresponding key points in two frames. If some edges are only green but not blue, it means that this edge has no motion between two frames, and the edge of the previous frame is covered by the edge of this frame. The accuracy of method 1 is obviously not as good as method 2. In Figure 3.6, the marking lines on the basketball court should not move between two frames because the camera is stationary. However, due to the influence of some outliers in the gating area, it caused undesirable motion. In comparison, although the operation of method 2 is slower, from a logical point of view, the motion of the edges is more consistent with common sense. In order to compare the error values of the two methods, the errors need to be compared with the ground truth image of the second frame, this comparison method is shown in Figure 3.8, and the loss of the comparison can be

[3] cv2.estimateAffinePartial2D()
[4] cv2.estimateAffine2D()

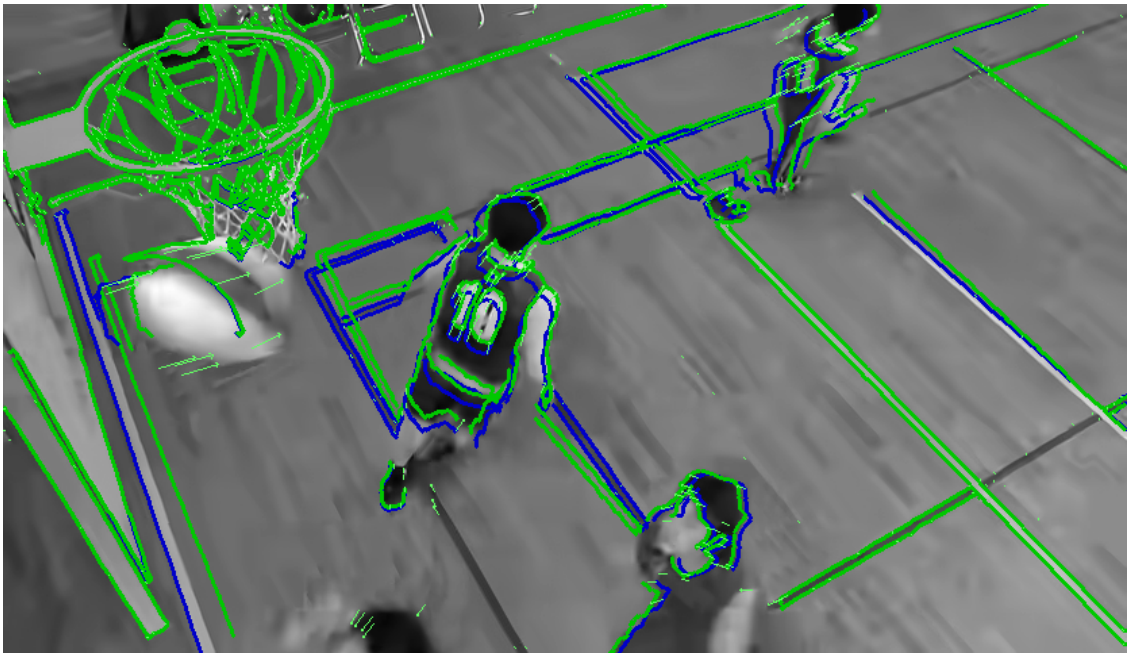

Figure 3.6: Result of edges tracking with method 1

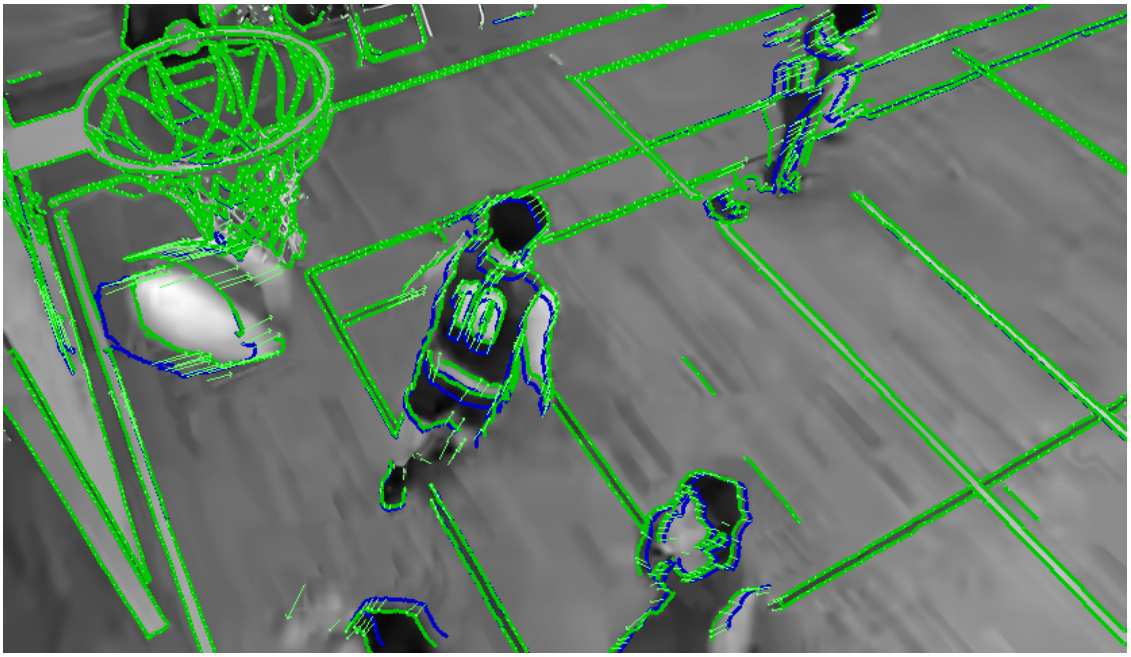

Figure 3.7: Result of edges tracking with method 2

calculated by

$$\text{MSE} = \frac{1}{mn} \sum_{i=0}^{m-1} \sum_{j=0}^{n-1} [I(i,j) - K(i,j)]^2. \tag{3.5}$$

Where I and K are the pixel matrices of the two images respectively, m and n are the height and width of two images. This formula calculates the squared error

between each pixel value in the two images, then adds them and divides them by the total number of pixels.

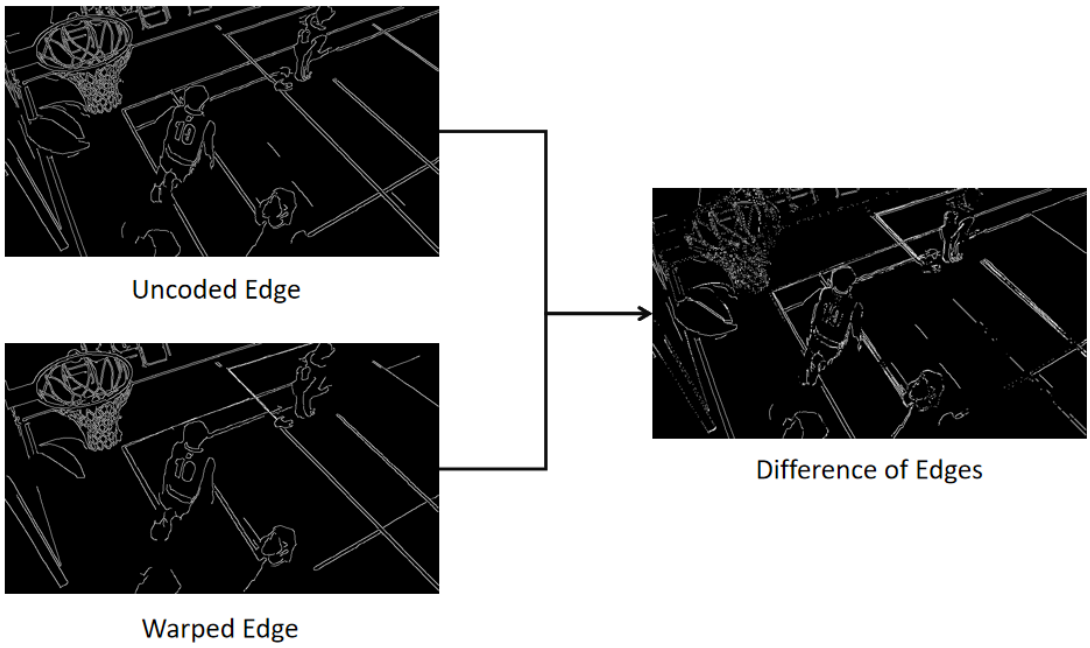

Figure 3.8: Loss calculation

In this way, the results from both methods can be compared with the original uncoded video to determine the accuracy. In order to better test the accuracy of the two algorithms, the two algorithms were tested on different video data, and the average loss was recorded. The comparison results of the two algorithms are in Table 3.1. By comparison, both methods have their own advantages. The algorithm

Table 3.1: Performance Comparison of two Edges Tracking Methods, N: Number of edges, M: number of pixels on one edge, K: Number of key points, the error unit is the square of the pixel value.

Method	**Assignment method**	**Robustness**	**Complexity**	**Average Loss**
Method 1	Gating area	Bad	$O(NK)$	0.0825
Method 2	Euclidean distance	Good	$O(NMK)$	0.0779

of method 1 has low algorithm complexity, but the accuracy is about 6% less than method two. Affected by the gating area threshold, it is difficult to set the threshold for each video and each edge, and it is easily affected by outliers, so there is no guarantee that this is a method with strong generalization ability. The key points in method 2 are closer to the edges, the edges receive more key points and the more edges are tracked, so method 2 is more robust.

3.6 Opimization

In the process of edge tracking, there are still many edges whose positions are inaccurate after warping, so it is considered that it is caused by the inaccuracy of key points after KLT tracking, therefore, optimization methods to reduce key point tracking errors should be introduced. According to the reprojection errors [26] that often occur in some 3D reconstruction technologies, technicians generally use the Bundle Adjustment [27] method to reduce reprojection errors and make the 3D reconstructed point cloud positions more accurate. Therefore, this task attempts to use this method to optimize the affine transformation matrix parameters of 2D images.

The current known variables include the key points $\mathbf{X}$ and $\mathbf{X}'$ related to the two edges in 2 frames and the affine transformation matrix $\mathbf{A}$ estimated by the KLT algorithm. Therefore, the key point positions $\mathbf{X}_{\mathrm{a}}$ obtained through affine transformation can be expressed as

$$\mathbf{X}_{\mathrm{a}} = \mathbf{X}\mathbf{A}. \tag{3.6}$$

Then the reprojection error could be represented by

$$E = \sqrt{\frac{1}{n}\sum_{i=1}^{n}\frac{(\mathbf{X}_i\mathbf{A} - \mathbf{X}_i')^2}{\sigma_I}}. \tag{3.7}$$

Only considering the summation within the square root, the loss formula can be expressed as

$$E(\mathbf{p}) = \sum_{i=1}^{n}\frac{\mathbf{r}^2(\mathbf{p})}{\sigma_I}, \tag{3.8}$$

where $\mathbf{r}$ is the residual between the key points from the first frame projected onto the second frame and the corresponding key points in the second frame $(\mathbf{X}_i\mathbf{A}-\mathbf{X}_i')^2$, and the σ_i is normalization factor, the stacked residuals can be represented by

$$E(\mathbf{p}) = \mathbf{r}(\mathbf{p})^T\mathbf{N}\mathbf{r}(\mathbf{p}), \tag{3.9}$$

where $\mathbf{N}$ is the normalization matrix. To minimize the projection error, non-linear least squares [28] is a good approach such as Gauss-Newton [29] and Levenberg-Marquardt [30] method , non- linear least squares problem can be efficiently optimized using standard second-order tools. The Gauss-Newton method iterates by linearizing residuals,

$$\mathbf{r}(\mathbf{p}) = \mathbf{r}(\mathbf{p}_i) + \nabla_{\mathbf{p}}\mathbf{r}(\mathbf{p}_i)(\mathbf{p} - \mathbf{p}_i), \tag{3.10}$$

$$\mathbf{J}_i := \nabla_{\mathbf{p}}\mathbf{r}(\mathbf{p}_i) \in \mathbb{R}^{dim(\mathbf{r})\times dim(\mathbf{p})}, \tag{3.11}$$

$$E(\mathbf{p}) = \frac{1}{2}\mathbf{r}(\mathbf{p})^T\mathbf{N}\mathbf{r}(\mathbf{p}), \tag{3.12}$$

$$\nabla_{\mathbf{p}}E(\mathbf{p}) = \mathbf{J}_i^T\mathbf{N}\mathbf{r}(\mathbf{p}), \tag{3.13}$$

$$\nabla^2_{\mathbf{p}} E(p) = \mathbf{J}_i^T \mathbf{N} \mathbf{J}_i := \mathbf{H}_i \in \mathbb{R}^{dim(\mathbf{p}) \times dim(\mathbf{p})}. \tag{3.14}$$

Then find minimum of linearized system, linearize and set $\nabla_{\mathbf{p}} E(\mathbf{p}) = 0$,

$$\nabla_p E(\mathbf{p}) \approx \nabla_{\mathbf{p}} E(\mathbf{p}) + \nabla^2_{\mathbf{p}} E(\mathbf{p}_i)(\mathbf{p} - \mathbf{p}_i), \tag{3.15}$$

$$\mathbf{p}_{i+1} = \mathbf{p}_i - (\nabla^2_{\mathbf{p}} E(\mathbf{p}_i))^{-1} \nabla_{\mathbf{p}} E(\mathbf{p}_i) = \mathbf{p}_i - \mathbf{H}_i^{-1} \mathbf{J}_i^T \mathbf{N} \mathbf{r}(\mathbf{p}_i). \tag{3.16}$$

Due to linearization, it may not be a good approximation of the Hessian far from the optimum (could even be degenerate). So the Levenberg-Marquardt method should be introduced: the idea of this method is „damping“ of step-length trades-off between Gauss-Newton and gradient descent:

$$\mathbf{p}_{i+1} = \mathbf{p}_i - (\mathbf{H}_i + \lambda \mathbf{I})^{-1} \mathbf{J}_i^T \mathbf{N} \mathbf{r}(\mathbf{p}_i). \tag{3.17}$$

If error decreases, decrease λ to shift towards Gauss-Newton, if error increases, reject update and increase λ to rather perform gradient descent. The Levenberg-Marquardt can converge from worse starting conditions than Gauss-Newton, however, it may necessitate additional iterations. Based on the above, the Levenberg-Marquardt algorithm has more advantages than the Gauss-Newton algorithm, so the Levenberg-Marquardt algorithm is selected for this optimization task. Since Levenberg-Marquardt is an iterative algorithm, if there are many key points, the calculation will be slower. Therefore, after each affine transformation, the reprojection error E is calculated, if this error is greater than a certain threshold θ, the Levenberg-Marquardt algorithm is used instead of optimizing the affine transformation matrix parameters of all edges. Here are some comparisons of the effects before and after optimization in statistical graph in Figure 3.9, it is a comparison of the reprojection errors of some key points before and after optimization. The

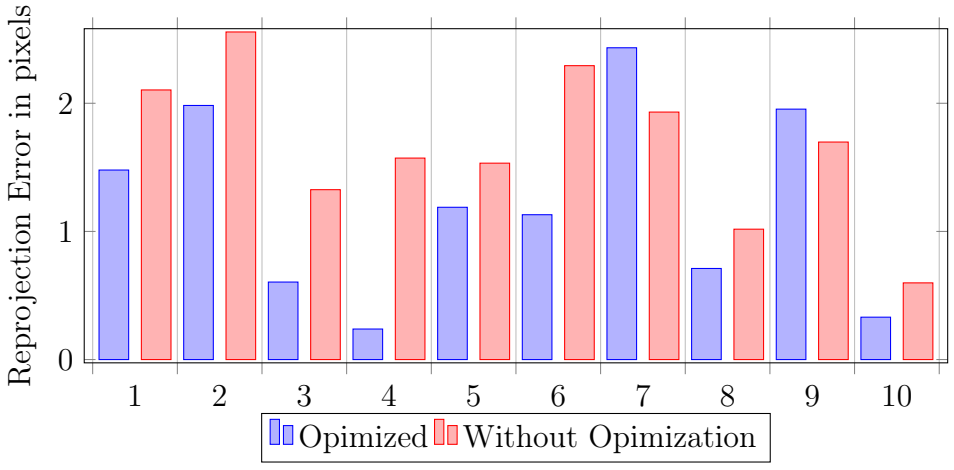

Figure 3.9: Error optimization comparison

histogram describes the reprojection error of key point sets belonging to different edges. It can be clearly seen that after optimizing, some key point sets' reprojection error has been significantly reduced, which can make the subsequent edge warping

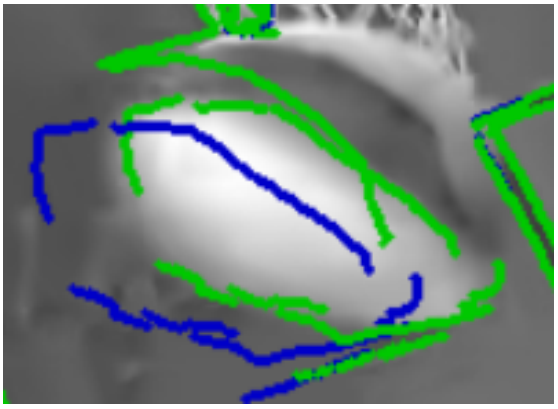

Figure 3.10: Basic result

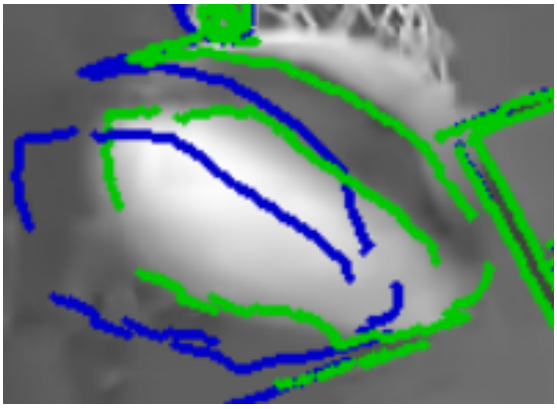

Figure 3.11: Optimized result

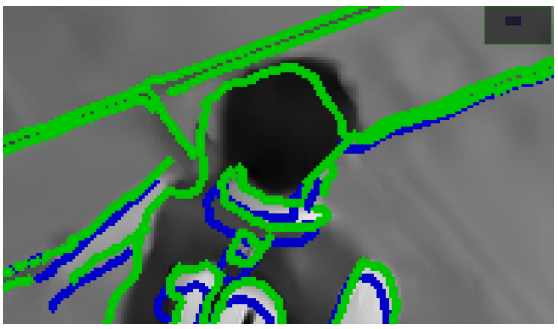

Figure 3.12: Basic result

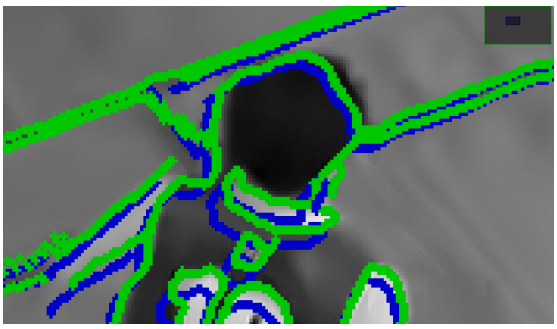

Figure 3.13: Optimized result

Figure 3.14: Error caused by outliers, in the process of key point tracking, the occurrence of outliers can lead to inaccuracies in the prediction of the affine transformation matrix, resulting in significant errors in the warped edges.

transformation more accurate. The following two sets of pictures show the results before and after optimization.

Algorithm 3: Edge Tracking Method 1

Input: $GrayImg1$, $GrayImg2$
Output: $AffinedEdges$

$EdgesImg1 \leftarrow CannyEdge(GrayImg1)$;
$EdgesImg2 \leftarrow CannyEdge(GrayImg2)$;
$\Omega \leftarrow EdgesImg1$;
$n \leftarrow \text{len}(\Omega)$;
$KeyPointsSet \leftarrow$ Null;
$KeyPoints1, KeyPoints2 \leftarrow KLT(EdgesImg1, EdgesImg2)$;
for $i = 1$ *to* n **do**
    `Build Mahalanobis distance for` Ω_i;
    for *keypoint in* $KeyPoints1$ **do**
        if *Mahalanobis distance*$(\Omega_i, keypoint) \leq$ *thres* γ **then**
            assign *keypoint* to $KeyPointsSet_i$;
        end
    end
    $AffineMatrix_i \leftarrow \text{leastsquare}(KeyPointsSet_i, KeyPoints2_i)$;
    $ReprojectionError \leftarrow$
     $||KeyPointsSet_i \cdot AffineMatrix_i - KeyPoints2||^2$;
    if *ReprojectionError* $\leq$ *thres* θ **then**
        $AffineMatrix_i \leftarrow \text{LevenbergMarquardt}(ReprojectionError)$;
    end
    $AffinedEdges \leftarrow \Omega_i \cdot AffineMatrix_i$;
end

In Figure 3.10, a green edge obtained by warping should coincide with a contour in the middle of the basketball. A green warping edge in Figure 3.12 should also coincide with the contour of the player's body. However, because the KLT tracker has slight errors in tracking key points in low-quality videos, the parameters in the subsequently obtained affine transformation matrix are less accurate, resulting in less than ideal results. Figure 3.11 and Figure 3.13 show the optimized results. After optimization, it can be clearly observed that the green edge of warping is closer to or has a higher degree of coincidence with the edge in the image. From the reprojection error statistics in Figure 3.9, it is also evident that occasionally, certain sets of key points exhibit an increase in error after optimization. This is because the least-squares optimization method is highly sensitive to outliers. If outliers are present during the key point tracking process of KLT, it can easily lead to larger errors after optimization. Figure 3.14 illustrates one such scenario of outliers. Due to the influence of background motion, KLT's corner tracking is affected, resulting in significant errors in optical flow prediction. Such errors are challenging to avoid. Therefore, in situations affected by such outliers, optimizing the reprojection error becomes a challenging task. In summary, the edge tracking method can be easily written as two methods as Algorithm 3 and Algorithm 4.

Algorithm 4: Edge Tracking Method 2

Input: $GrayImg1$, $GrayImg2$
Output: $AffinedEdges$

$EdgesImg1 \leftarrow CannyEdge(GrayImg1)$;
$EdgesImg2 \leftarrow CannyEdge(GrayImg2)$;
$\Omega \leftarrow EdgesImg1$;
$n \leftarrow \text{len}(\Omega)$;
$KeyPointsSet \leftarrow$ Null;
$KeyPoints1, KeyPoints2 \leftarrow KLT(EdgesImg1, EdgesImg2)$;
$KeyPoints1, KeyPoints2 \leftarrow KLT(EdgesImg1, EdgesImg2)$;
for $i = 1$ *to* n **do**
    for *keypoint in* $KeyPoints1$ **do**
        for *edgepoint in* Ω_i **do**
            if $||keypoint - edgepoint||^2 \leq thres\ \gamma$ **then**
                assign *keypoint* to $KeyPointsSet_i$;
            end
        end
    end
    $AffineMatrix_i \leftarrow \text{leastsquare}(KeyPointsSet_i, KeyPoints2_i)$;
    $ReprojectionError \leftarrow$
     $||KeyPointsSet_i \cdot AffineMatrix_i - KeyPoints2||^2$;
    if $ReprojectionError \leq thres\ \theta$ **then**
        $AffineMatrix_i \leftarrow \text{LevenbergMarquardt}(ReprojectionError)$;
    end
    $AffinedEdges \leftarrow \Omega_i \cdot AffineMatrix_i$;
end

The overall algorithmic workflow can be broadly depicted using Figure 3.15.

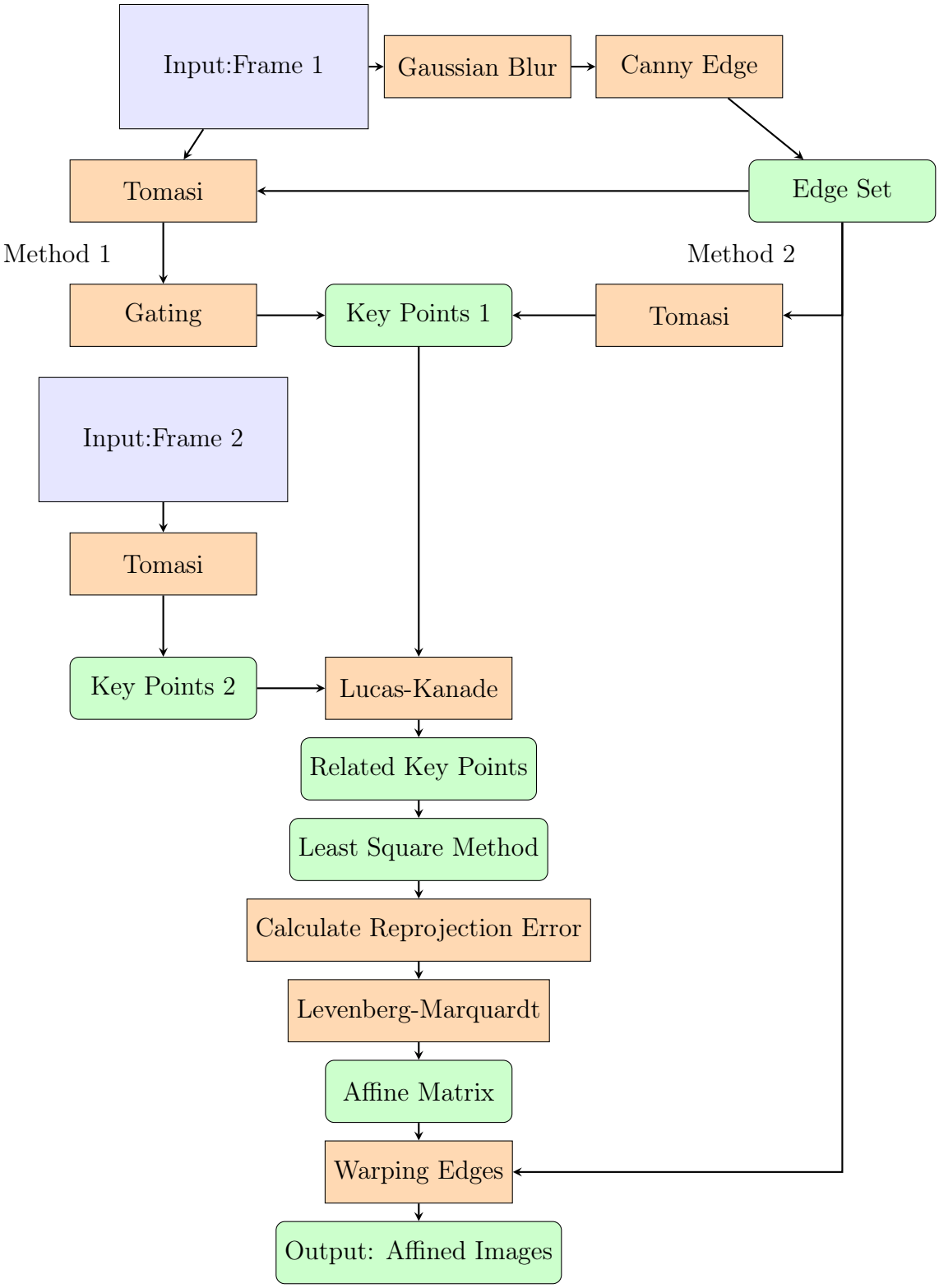

Figure 3.15: Edge Tracking Algorithm

4 Experimental Results

In this chapter, the results of the edge tracking algorithm development and experimentation will be presented. The results are organized to provide a comprehensive understanding of the performance and efficacy of the proposed algorithm. Emphasis will be placed on quantitative metrics assessing accuracy and efficiency. Subsequently, attention will be given to qualitative assessments, including visual representations of the algorithm's output. The capabilities of the algorithm and its contributions to the field of video coding will be demonstrated through these results. This chapter primarily focuses on the outcomes obtained from the edge tracking algorithm. It involves the visual and data-based evaluation of these results, as well as a comparative analysis between the outcomes of this algorithm and those of other optical flow methods. The objective is to explore the strengths and limitations of this algorithm, thereby shedding light on its superiority and constraints.

4.1 Tracking Result

The methodology presented in Chapter 3 allows for the computation of rotation and translation parameters, namely affine transformation parameters, for every edge in two successive frames. This enables us to establish the motion relationship of each edge between the two frames. Consequently, we can not only detect the edges in the second frame using Canny edge detection but also transform the edges from the first frame to match those in the second frame using affine transformations. In practical terms, this means that we can deduce the edges in the second frame from those in the first frame and obtain measurements for the edges in the second frame. This provides us with a more accurate depiction of edge motion between 2 frames, which is highly beneficial for the video encoding process. From Figure 4.1, it is evident that in the uncoded video frames, which are essentially distortion-free, the contours of every edge appear remarkably sharp and exhibit high clarity. Moreover, the Canny edge algorithm successfully detects a multitude of edges in these frames, visually highlighting their distinctiveness. However, in Figure 4.2, due to the severe distortion and blurriness introduced in the encoded video frames, the quality of the detected edges is notably inferior when compared to the uncoded videos. Additionally, the encoded frames exhibit a considerably reduced number of detected edges in contrast to their uncoded counterparts. In Figure 4.3, it can be observed that, as a result of the motion between the previous frame and the current frame, the edges projected from the previous frame impart essential information to the current frame. Consequently, the current frame acquires both the edges detected in the current frame and the positional and motion data of the edges from

the previous frame.

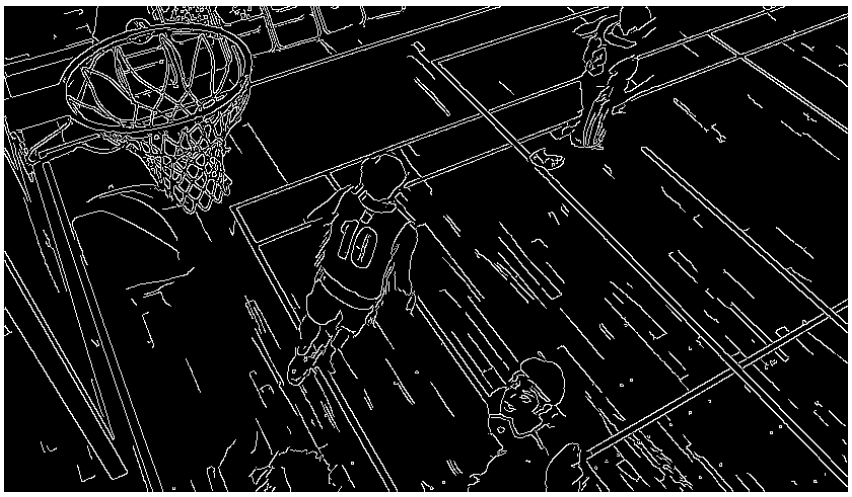

Figure 4.1: Uncoded frame

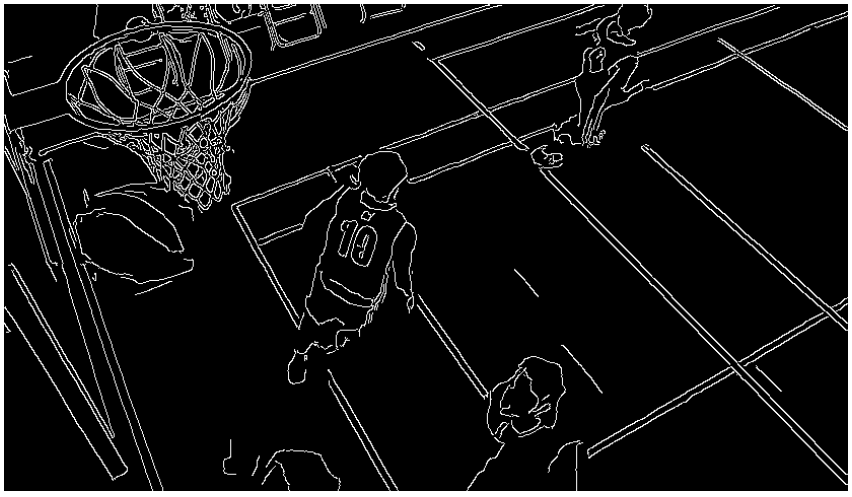

Figure 4.2: Coded frame

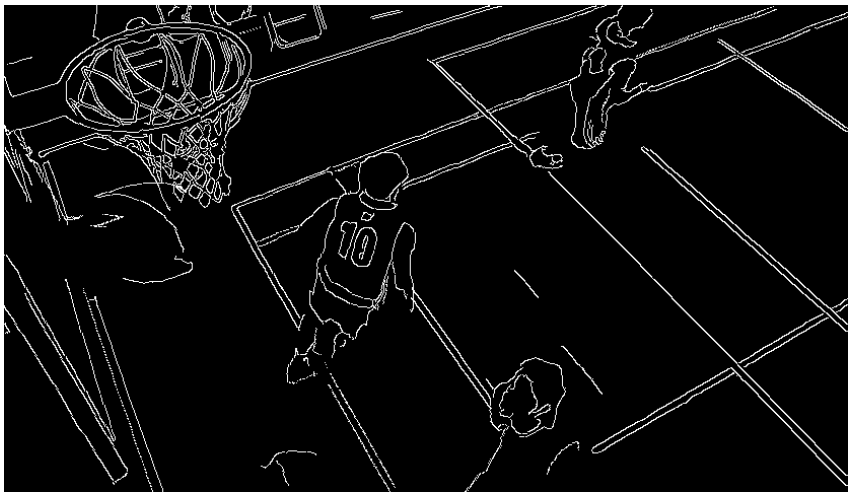

Figure 4.3: Warped frame

To assess the robustness and generalization capabilities of this algorithm, it is essential to conduct testing on a broader range of videos. In this task, we tested the algorithm's tracking performance on several videos, including BQsquare, Cactus (2K), Arena of Valor (2K), and Daylight Road 2 (4K). To rigorously evaluate the algorithm's tracking performance robustness, we deliberately selected videos of the lowest quality, adhering to the $Qp = 42$ standard. Furthermore, in pursuit of achieving superior tracking results, the second approach outlined in Chapter 3 has been adopted.

Figure 4.4: Edge tracking for Arena of Valor

Figure 4.5: Coded frame of Arena of Valor

Comparing Figure 4.4 and Figure 4.5, it becomes evident that there are notable differences in the edges between the two frames. In this video, some finer edges from the previous frame were not successfully projected onto the current frame. Meanwhile, the edges detected in the current frame exhibit more detail. This could be attributed to edges detected in the current frame that were not present in the previous frame or to certain minor edges in the previous frame that were not assigned to three or more key points, as a result, the obtained tracking performance is not optimal.

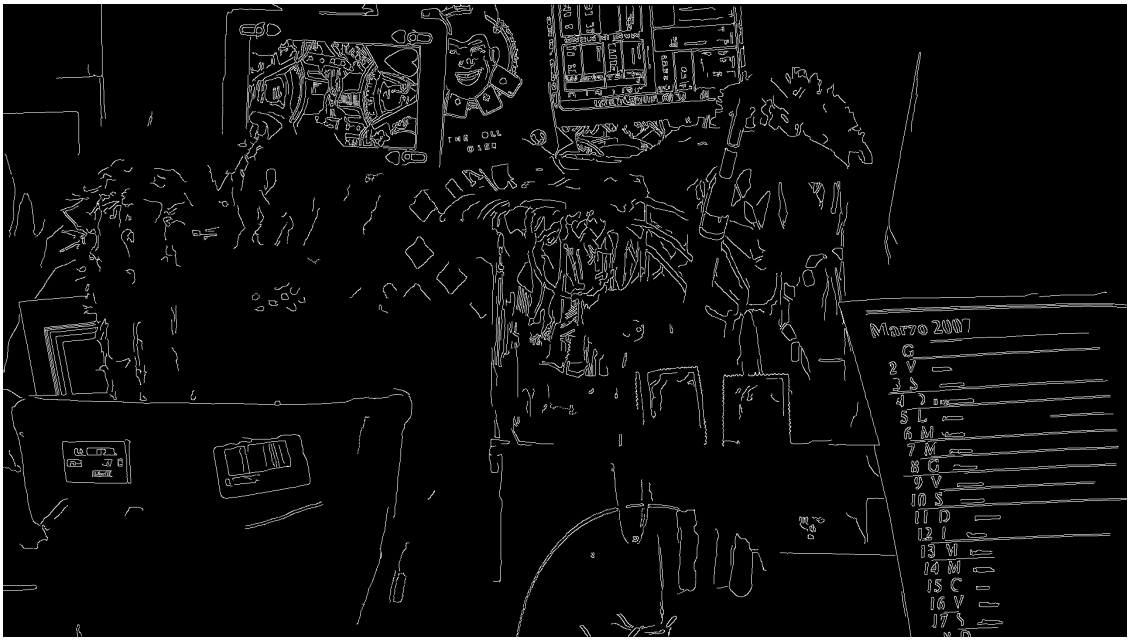

Figure 4.6: Edge tracking for Cactus

Figure 4.7: Coded frame of Cactus

When comparing Figure 4.6 and Figure 4.7, it becomes evident that the tracking results closely align with the detection results of the current frame. However, there are still some challenges in tracking certain fine-grained and relatively short edges.

Figure 4.8: Edge tracking for Daylight Road 2

Figure 4.9: Coded frame of Daylight Road 2

Comparing Figure 4.8 and Figure 4.9, it becomes evident that tracking finer details in 4K videos is notably challenging. This necessitates the introduction of a considerable number of key points, thereby requiring increased computational memory and processing time. It's important to note that tracking is merely an

auxiliary algorithm. Attempting to track every intricate detail and edge within 4K videos would significantly consume both memory and processing time resources.

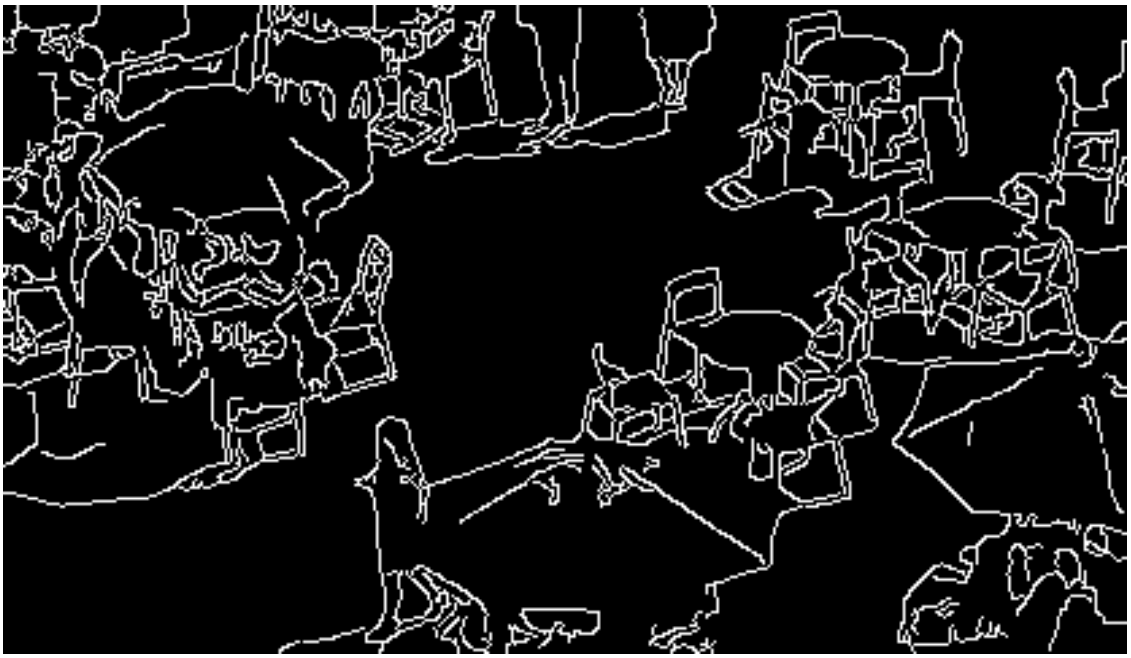

Figure 4.10: Edge tracking for BQsquare

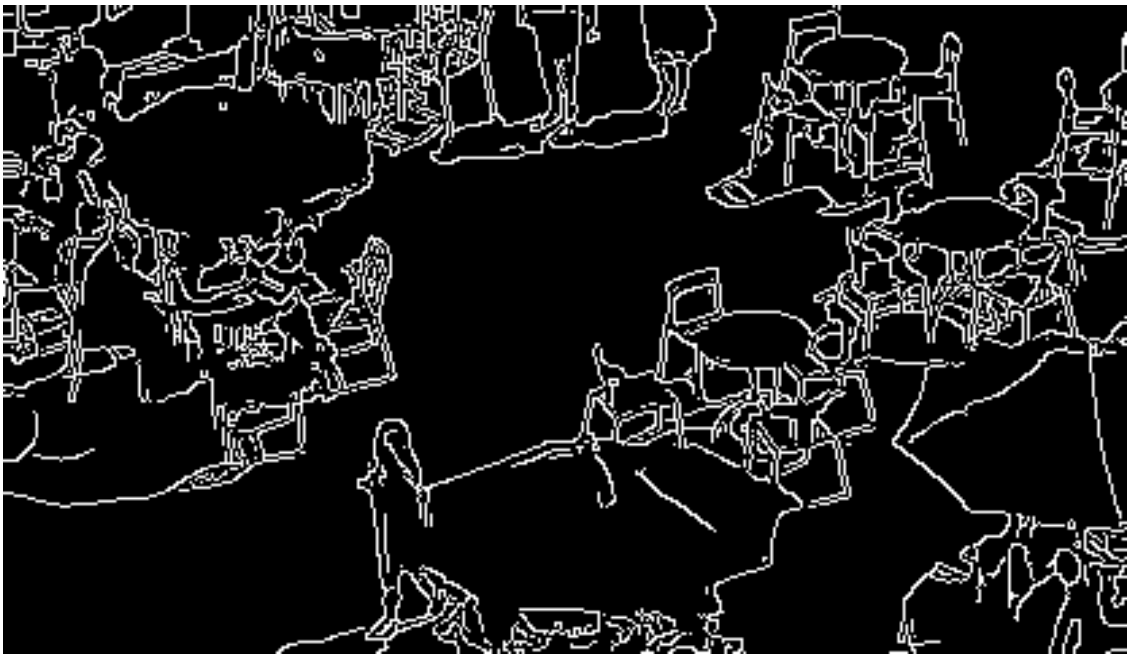

Figure 4.11: Coded frame of BQsquare

Comparing Figure 4.10 and Figure 4.11, it's evident that BQsquare is a relatively straightforward video. In this video, the camera experiences minimal movement, and objects within the scene move slowly. Furthermore, the video has a low pixel resolution. As a result, when tracking this video, there is minimal loss of fine details

in the images, and the tracked edges closely match the edges in the current frame.

4.2 Comparison of Results under Different Video Quality

To assess the impact of varying video qualities on tracking results, this section continues to employ edges extracted from videos of different qualities and compares their accuracy with edges detected in unencoded videos using Mean Squared Error (MSE) as defined in Equation 3.5.

Figure 4.12: $Qp = 27$

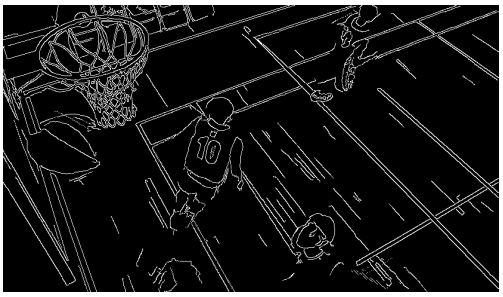

Figure 4.13: $Qp = 32$

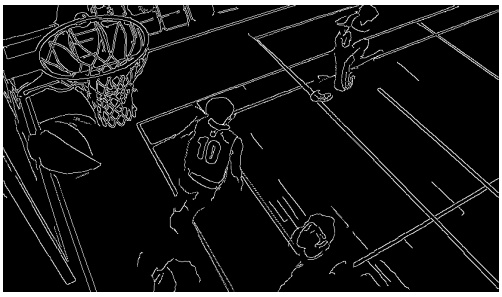

Figure 4.14: $Qp = 37$

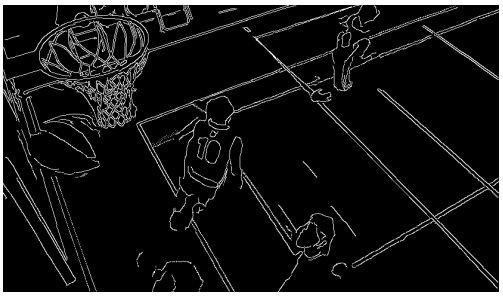

Figure 4.15: $Qp = 42$

The sequence of results from Figure 4.12 to Figure 4.15 illustrates four different scenarios corresponding to varying video qualities. From a visual standpoint, it becomes evident that as video quality deteriorates, fewer edges are successfully tracked. This behavior can be attributed to the fact that our algorithm relies on the Canny edge detection method and key point tracking. In lower-quality video frames, the inherent blurriness in the video imagery leads to a limited number of edges detected by the Canny edge algorithm. Simultaneously, the influence of distortion and block artifacts disrupts the key point tracking process, resulting in challenges in locating or accurately tracking key points. Consequently, fewer key points are assigned to each edge due to these combined factors. In order to accurately assess the precision of tracking results in comparison with uncoded video, it is imperative to select multiple videos and conduct comparisons under varying quality conditions.

Videos	27	32	37	42
BQsquare	0.053	0.058	0.068	0.083
Basketball Drill	0.045	0.051	0.056	0.064
Cactus(2K)	0.066	0.070	0.074	0.079
Arena of Valor(2K)	0.049	0.053	0.056	0.063
Daylight Road 2(4K)	0.043	0.045	0.046	0.048

Table 4.1: Mean square error in different quality videos, the entries are the average error of 32 frames, the error unit is the square of the pixel value.

From Table 4.1, it is evident that as video quality deteriorates, the tracking algorithms for edges yield increasingly higher error values. Furthermore, the more intricate the details within a video, the higher the tracking errors tend to be. For instance, in the case of the Cactus video, despite the relatively straightforward object motion, the video contains a wealth of intricate object details, making it challenging to comprehensively track its edges.

4.3 Compared to other Optical Flow Methods

In order to assess the accuracy and practicality of the edge tracking algorithm, it is imperative to compare this algorithm with previously implemented optical flow methods. The optical flow methods to be considered in this comparison are the Farneback method and the FlowNet2 method, as mentioned earlier. These two op-

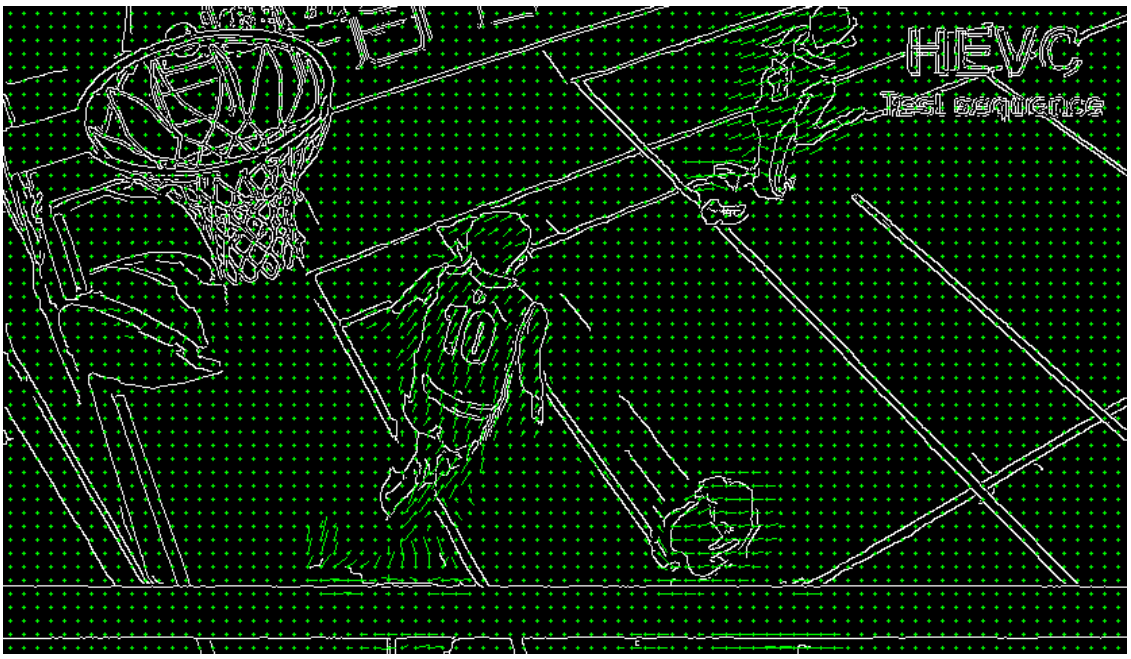

Figure 4.16: Optical flow of Farneback method

tical flow techniques enable the exploration of pixel motion across the entire image,

rather than solely relying on sparse tracking. This broader perspective allows for the assessment of motion along the edges throughout the entire image, facilitating a comprehensive comparison with the edge tracking algorithm. There is a result of Farneback method in Figure 4.16. Farneback optical flow uses a polynomial-based model to fit the motion of local image patches. These polynomial models can be used to estimate the motion of neighboring pixels, building the optical flow field. In the optical flow image, the average optical flow of each pixel block is represented by green arrows. These arrows indicate the direction of motion for each pixel block, and the length of the arrows reflects the intensity of motion. Longer arrows signify more vigorous motion in the corresponding pixel block, while shorter arrows represent less intense motion. This visualization allows us to discern that objects in motion, such as a basketball and the surrounding pixel blocks near the athlete, exhibit stronger motion patterns. The final results of Farneback optical flow can

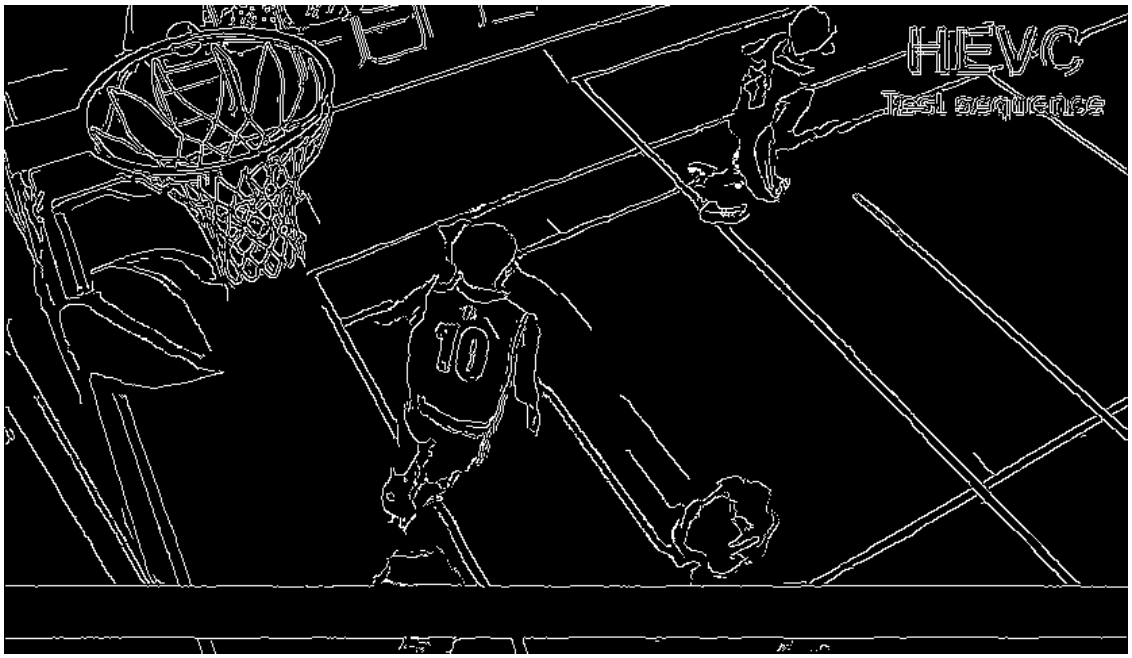

Figure 4.17: Warping result of Farneback method

be observed in Figure 4.17. The tracking implemented using this method exhibits relatively high noise levels. It is evident from the results that the tracking outcomes for each edge lack the desired smoothness. In Figure 4.18, we observe the optical flow generated by FlowNet2. Optical flow maps are typically color-coded to represent various motion directions and speeds. For instance, the color red indicates motion in the positive horizontal direction (from left to right), while blue signifies motion in the negative horizontal direction (from right to left). This color encoding aids in visualizing the optical flow information for each pixel. The FlowNet2 model employed in this task is the largest within the FlowNet series. Consequently, its inference process involves relatively high computational complexity. However, in terms of optical flow, the results are quite accurate. This is because the targets involved in this task, such as athletes and basketball, exhibit clear optical

Figure 4.18: Optical flow of FlowNet2

flow patterns, and their edges are relatively smooth. Nonetheless, some blurriness may also be present. There is the architecture of FlowNet2 in Figure 4.19, it is a

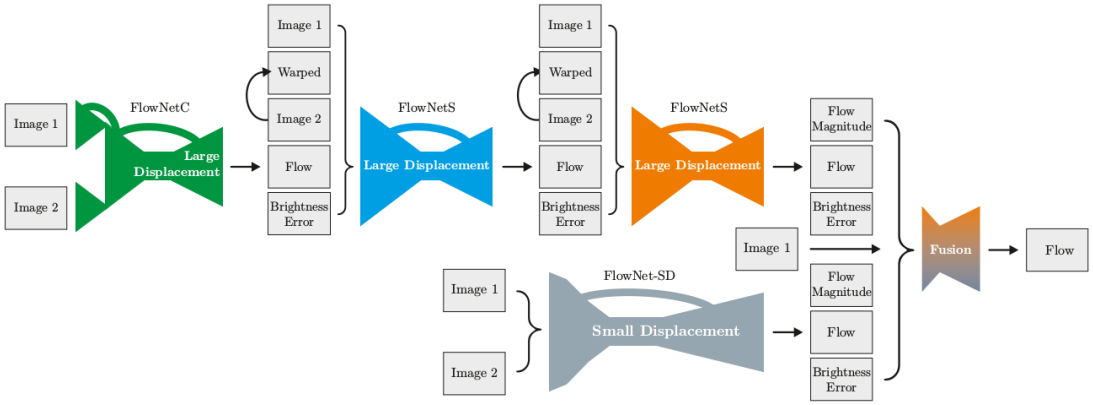

Figure 4.19: FlowNet2 architecture [2]

stacked architecture, FlowNet2 utilizes a sophisticated architecture incorporating several instances of FlowNetC and FlowNetS stacked together, enabling the estimation of large-displacement flow. Additionally, a dedicated sub-network is designed to handle small motions, ensuring precise motion analysis. These components are seamlessly integrated through a fusion layer, enhancing the model's overall performance and accuracy in capturing diverse motion patterns. Figure 4.20 displays the inference results of FlowNet2, and it is evident that in certain areas where the background meets the foreground, there are no ambiguous variations observed. In

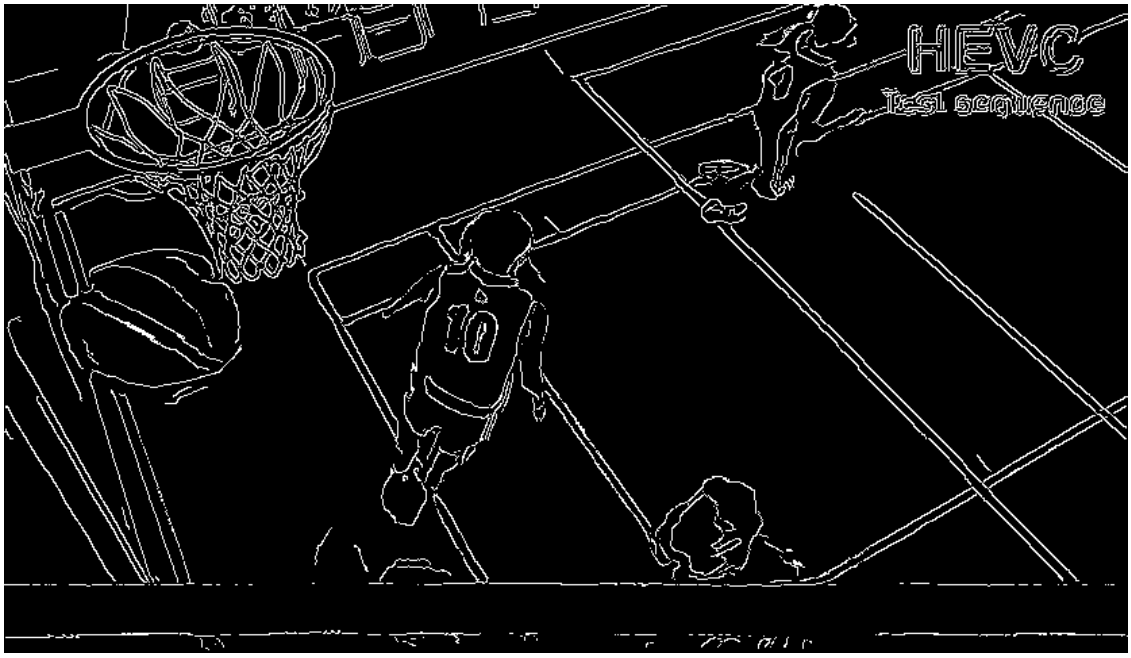

Figure 4.20: Warping result of FlowNet2

this regard, FlowNet2 outperforms the Farneback method. However, there are still some noise artifacts present in certain edges, which do not appear as smooth as they do in the optical flow results. This implies that neural networks may exhibit some inaccuracies in estimating optical flow. For the Edge Tracking algorithm,

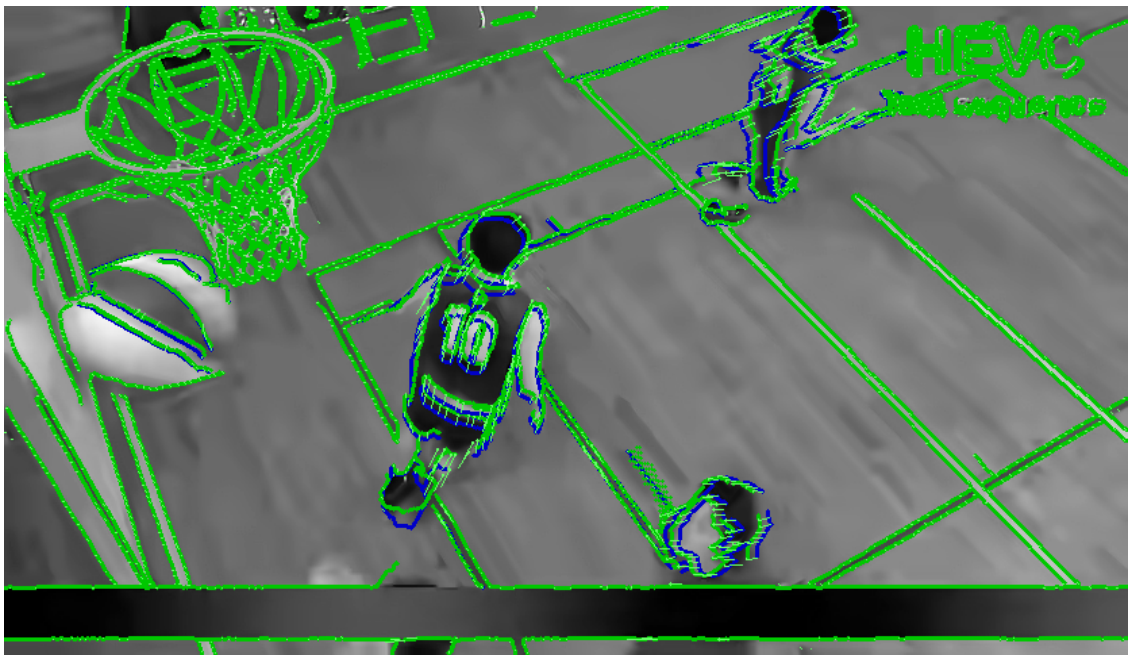

Figure 4.21: Edge warping of edge tracking method

detailed explanations can be found in Chapter 3. As depicted in Figure 4.21, in

comparison to the first two optical flow algorithms, the Edge Tracking algorithm is specialized in tracking edges. In the entire image, noticeable motion changes are

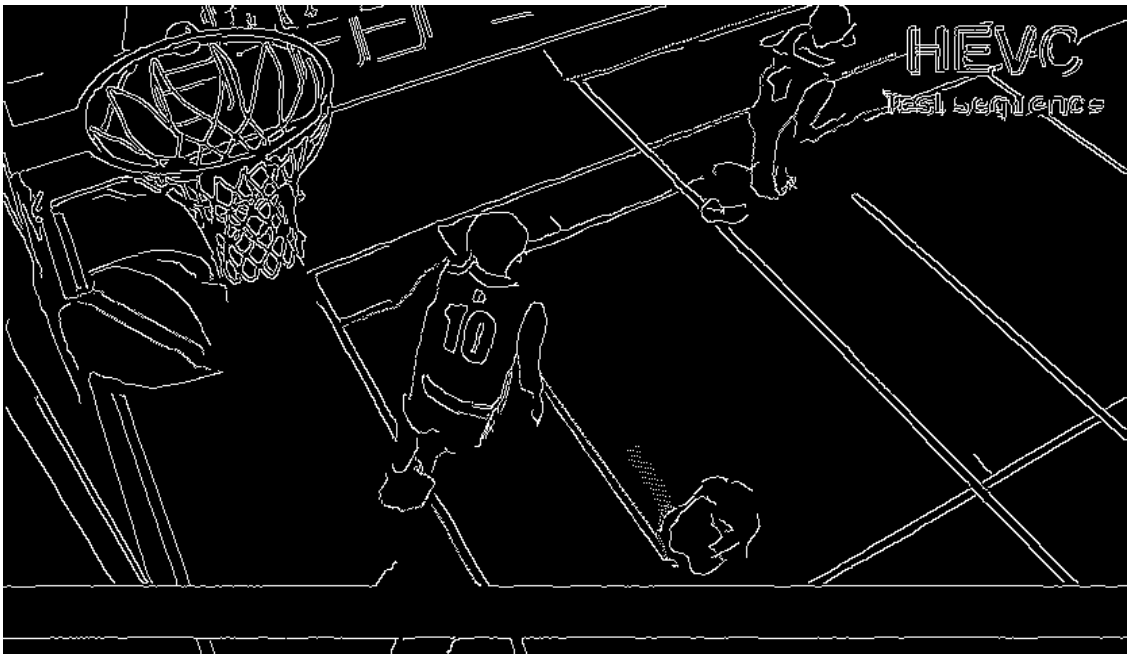

Figure 4.22: Result of edge tracking method

primarily observed around the edges of the athlete and the basketball. Specifically, the difference in position between the blue edges from the previous frame and the green edges predicted for the current frame signifies the manifestation of optical flow or, in other words, "edge flow" within this algorithm. Compared to the previous two methods, as it shown in Figure 4.22 edge tracking algorithms primarily focus on tracking the prominent edges within an image, rather than all the pixels in the entire image. This approach can be described as a form of sparse edge tracking algorithm. It's evident that, in comparison to the first two algorithms, the post-tracking results exhibit less pronounced noise along the edges. Moreover, the tracked edges appear smoother, and the regions where moving objects intersect with the background are visually more accurate than in the Farneback optical flow case.

To conduct a more precise analysis of the differences between the three algorithms, a visual comparison of the algorithms should be initiated, focusing on image details. Figures 4.23 to Figure 4.26 illustrate the predictions of basketball motion in the Basketball Drill video using these three algorithms. This exploration delves into the post-tracking effects of the basketball itself and its impact on the surrounding background. Figure 4.23 displays the edges extracted from the unencoded video frame as a reference, while Figure 4.24 illustrates the results obtained using the Farneback method. Due to the relatively high speed of the basketball's motion, it introduces some distortion to its edges. The use of the Farneback algorithm can make foreground motion easily interfere with the background's motion, leading to undesirable distortions in the background edges as well. Figure 4.25 illustrates the

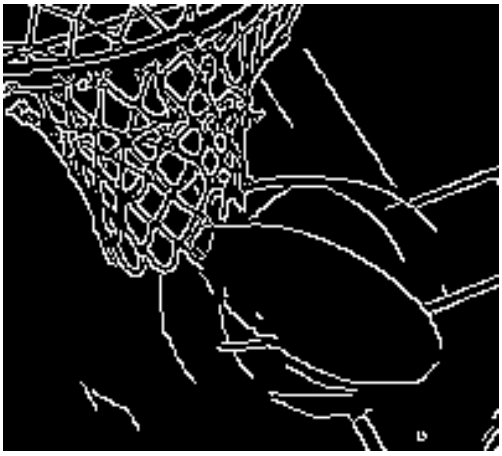

Figure 4.23: Ground truth

Figure 4.24: Farneback result

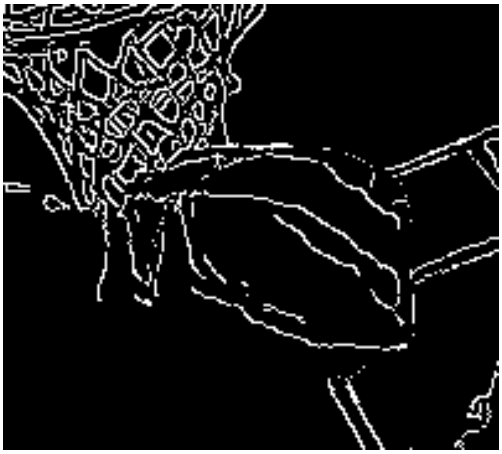

Figure 4.25: FlowNet2 result

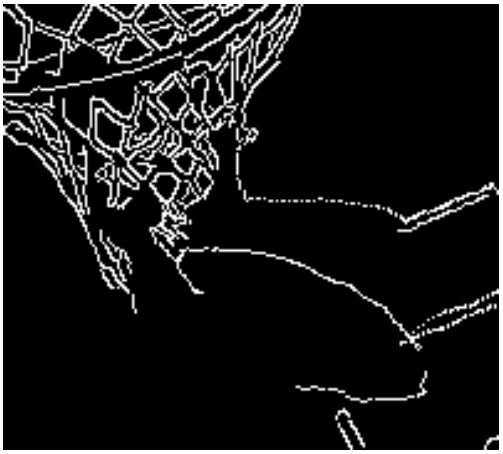

Figure 4.26: Edge tracking result

results of the FlowNet2 method. Due to the dynamic blurring caused by basketball motion, ghosting artifacts appear along the basketball's boundary. This means that the boundary from the previous frame is still predicted by the neural network in the current frame. However, it's noticeable that the motion of the basketball doesn't affect the edges of the background. Figure 4.26 illustrates the results of the edge tracking algorithm. While this algorithm tracks fewer edges, it does so without introducing ghosting artifacts in the image. Moreover, it has a minimal impact on the edges between the basketball and the background. In achieving these improvements, this algorithm addresses the issues present in the two aforementioned algorithms, albeit at the cost of sacrificing global tracking.

To assess the accuracy of tracking using three different methods, it is essential to calculate the losses associated with each method. In this context, Formula 3.3 is employed for loss computation. By comparing the errors under the worst video quality conditions, specifically the $Qp = 42$ standard with uncoded video, we can evaluate and compare the performance of the three algorithms. In comparison Table 4.2, we can see the mean square error for three different methods: the Farneback Method, FlowNet2, and our edge tracking method. The edge tracking method

Videos	Farneback Method	FlowNet2	Edge Tracking
BQsquare	0.101	0.100	0.083
Basketball Drill	0.079	0.089	0.064
Cactus (2K)	0.093	0.110	0.079
Arena of Valor (2K)	0.075	0.063	0.053
Daylight Road 2 (4K)	0.065	Out of Memory	0.048

Table 4.2: Mean square error in different Methods, the entries are the average error of 32 frames, the error unit is the square of the pixel value.

stands out with the lowest error values, highlighted in green. This emphasizes edge tracking algorithm's optimal performance in accurately tracking edges in various video sequences, even under challenging conditions. When it comes to handling 4K videos, it is evident that FlowNet2 requires a robust GPU processor to execute, highlighting its high computational complexity. For tasks that exclusively involve edge tracking, the accuracy of edge tracking algorithms surpasses that of the first two optical flow algorithms in terms of positional precision.

5 Results

Based on previous experimental results and insights from discussions, this chapter delves into the application outcomes of edge tracking algorithms and explores some potential challenges. Following edge tracking, we obtain the rotation and translation for each edge. Consequently, we can interpolate edge images between two frames, and even predict the positions of each edge that appears in the next frame based on the motion of edges from the previous frame to the current frame. However, since this algorithm is based on Lucas-Kanade and affine transformation methods for sparse tracking, it may introduce distortions, generate artifacts, and result in inaccuracies. This chapter aims to elucidate the application and issues of the edge tracking method, shedding light on the ultimate effectiveness of the algorithm's results and some underlying problems.

5.1 Application of Edge Tracking Algorithm

After obtaining the motion of each edge using edge tracking algorithms, one can apply a frame interpolation technique akin to optical flow to video sequences. This process involves using the edge motion to interpolate between two consecutive video frames, resulting in the generation of an intermediate frame. Frame interpolation is a valuable video processing technique with a wide range of applications. By inserting additional frames between video frames, it serves to increase the frame rate, thereby enhancing smoothness, reducing flickering, and enabling special effects like slow-motion and time stretching. Furthermore, frame interpolation can be employed for video format conversion, ensuring that the video complies with specific frame rate requirements. In terms of improving video image quality and the viewing experience, frame interpolation plays a vital role and finds widespread use in entertainment, scientific research, and various other domains.

As shown in Figure 5.1, the blue edges represent the edges detected in the previous frame, while the green edges are the edges tracked from the previous frame to the current frame based on their motion. If interpolation is required, the inserted frame should be positioned between the blue and green edges. The pink edges in the image correspond to the edges of the interpolated middle frame. It is evident that the motion of the pink edges is slower than that of the green edges. The interpolation operation applied here involves slowing down the velocity of each edge, thereby obtaining the intermediate position of edge motion between the two frames. The results of frame interpolation, as depicted in Figure 5.4, show that the positions of the edges in the inserted frame are determined based on the motion between the edges in Figure 5.2 and Figure 5.3, the two adjacent frames.

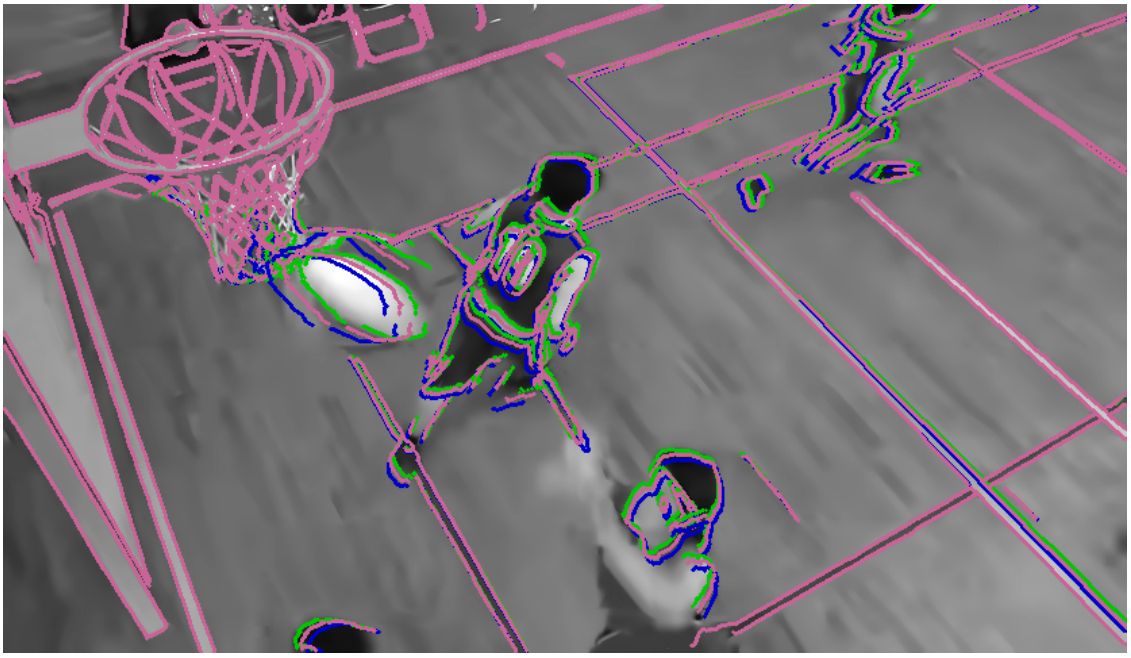

Figure 5.1: Interpolated edges, blue edges are the edges of the last frame, green edges are warped edges in current frame and the pink edges are interpolated edges between 2 frames.

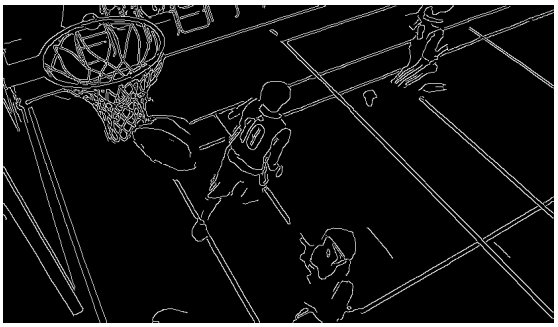

Figure 5.2: Edges of last frame

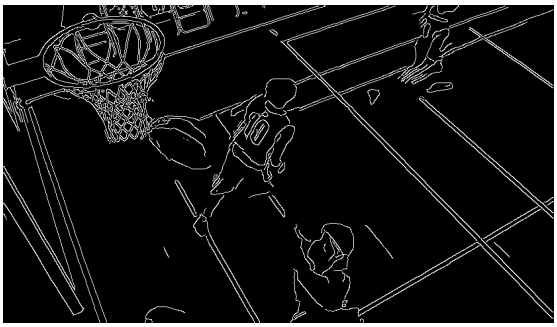

Figure 5.3: Edges of current frame

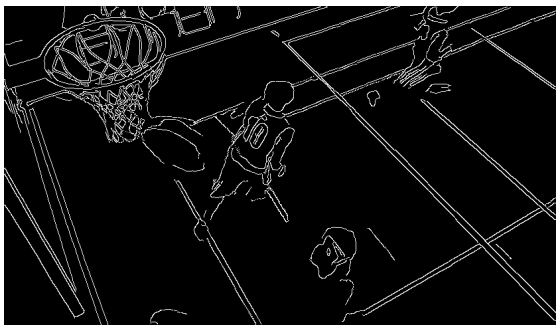

Figure 5.4: Interpolated intermediate frame

In addition to interlacing the videos of edges, it is also possible to predict the

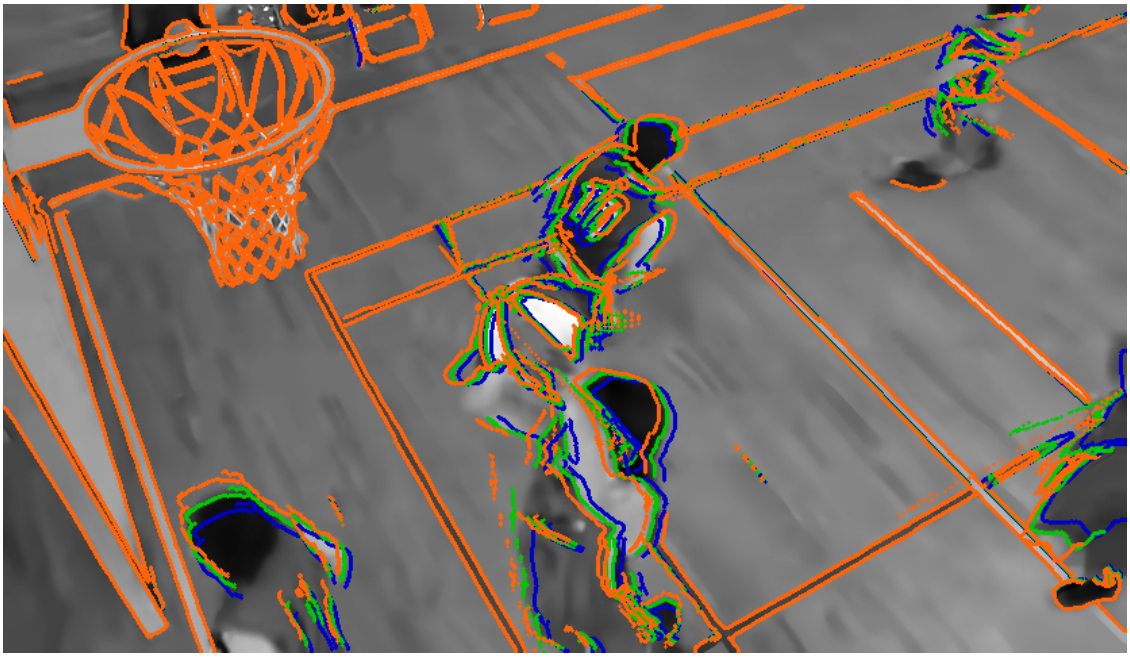

Figure 5.5: Predicted edges, blue edges are the edges of the last frame, green edges are warped edges in current frame and the orange edges are predicted edges for next frame.

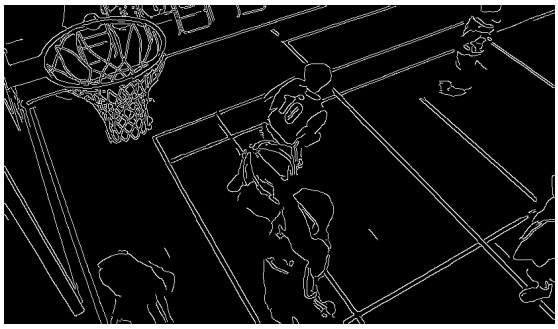

Figure 5.6: Edges of last frame

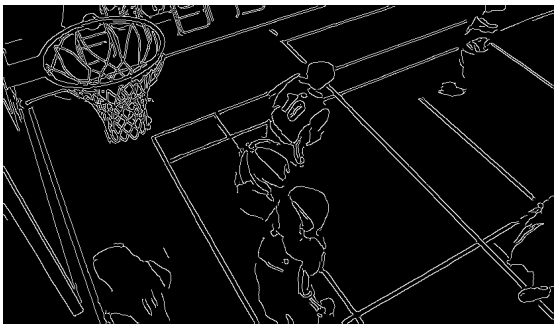

Figure 5.7: Edges of current frame

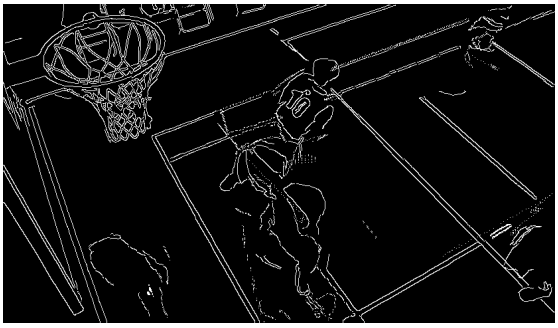

Figure 5.8: Predicted next frame

positions of edges in the next frame based on their locations in the previous frame and the current frame. Predicting the next frame in a video offers multiple benefits.

It can reduce bandwidth demands during video streaming, making it particularly advantageous in low-quality network conditions. In real-time video processing, this prediction enhances responsiveness by minimizing latency. Additionally, in object tracking applications, predicting the next frame's position improves tracking accuracy and stability as it anticipates the object's location [31]. In Figure 5.5, as in the previous scenarios, the blue edges represent edges from the previous frame, the green edges are the edges obtained by projecting the current frame using edge tracking algorithms, and the orange edges are inferred for the next frame based on the motion of edges in the current and previous frames. It is evident that for moving objects, the positions of the surrounding orange edges are ahead of the confirmed motion direction indicated by the blue and green edges. Figure 5.8 illustrates the prediction of the next frame based on the previous frame in Figure 5.6 and the current frame in Figure 5.7.

5.2 Algorithm Problems

When applying the developed edge tracking algorithm, several challenges have emerged. Firstly, this algorithm is built upon the KLT algorithm, which places certain demands on video quality. In cases of lower video quality, it is highly likely to encounter outliers during the tracking process. Additionally, a crucial step in edge tracking involves affine transformations, where the least squares method is employed to estimate these transformations based on the tracked key points. This method is sensitive to outliers, and thus, when using the Tomasi algorithm to detect corner points, lower-quality corner points can lead to significant errors during the application of the Lucas-Kanade method for tracking. As illustrated in Figure 3.2 of Chapter 3, a single erroneous outlier can profoundly impact the accuracy of the affine transformation, resulting in the projection of edges to non-meaningful locations. In the presence of outliers, the projection of edges can result in artifacts, as illustrated in Figure 5.11. Around moving objects, the Tomasi corner detection method is susceptible to detecting artifacts as corner points in low-quality videos due to the influence of dynamic blurring and blocking effects can lead to tracking errors in KLT key point tracking. Such errors result in incorrect parameter predictions within the affine transformation matrix. The accuracy of parameters within the affine transformation matrix is crucial for the precise positioning of edge projections. Even the slightest deviation in these matrix parameters can lead to significant errors in the entire affine transformation. As a consequence, discontinuous artifacts, represented by the points in the image, emerge. These points are artifacts caused by the inaccurate predictions of a particular edge's projection due to errors in the affine transformation matrix.

To address this issue, in future work, it is advisable to explore additional tracking algorithms. For instance, a combination of robust methods such as the Scale-Invariant Feature Transform (SIFT) [32] algorithm and the Random Sample Consensus (RANSAC) [33] denoising algorithm could be considered for key point tracking. Such combinations are often used in key point matching tasks, but it is im-

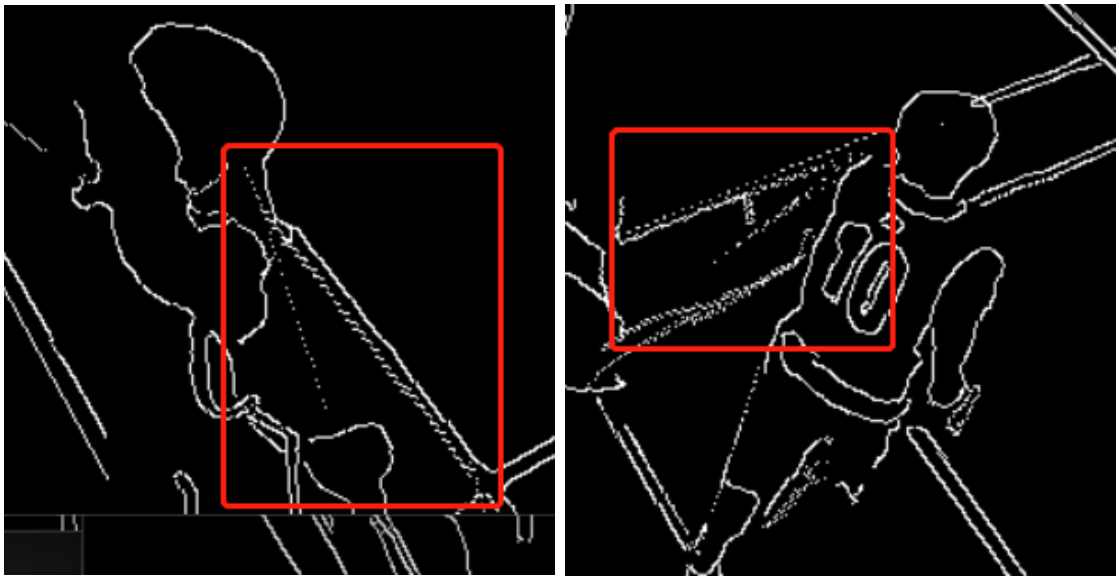

Figure 5.9: Artifacts generated from edge tracking method, these discontinuous points in the picture are artifacts.

portant to note that they come with higher computational complexity compared to the Tomasi algorithm. Additionally, the robustness of the SIFT algorithm against noise is not guaranteed to be superior in all cases. Therefore, this aspect will also become a part of the algorithm's future optimization efforts.

Apart from the artifacts caused by key point tracking errors, there are also errors

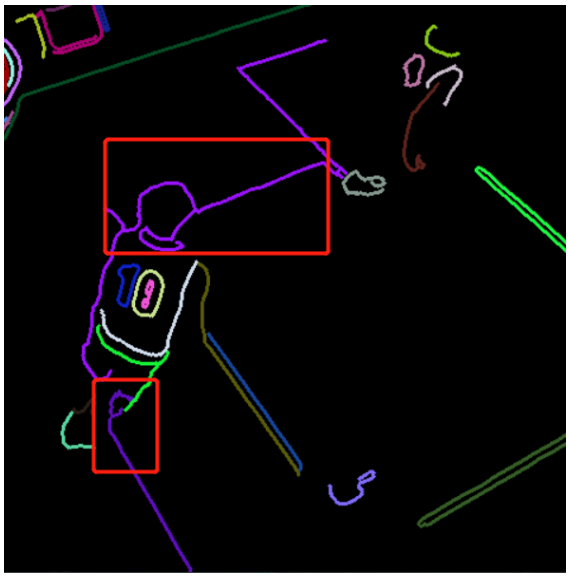

Figure 5.10: Incorrectly connected edges, within the red bounding box, combine the edge of the moving person in the foreground with the reference line on the background motion field into a single edge.

stemming from the generation of the edge set. Specifically, the challenge arises

when the connections of each edge fail to accurately distinguish between the background and foreground, moving objects and stationary objects. This type of error can result in the erroneous merging of two edges that should have distinct motion due to inaccuracies in the edge connection process. Consequently, these merged edges share a common motion, leading to unexpected artifacts. As depicted in

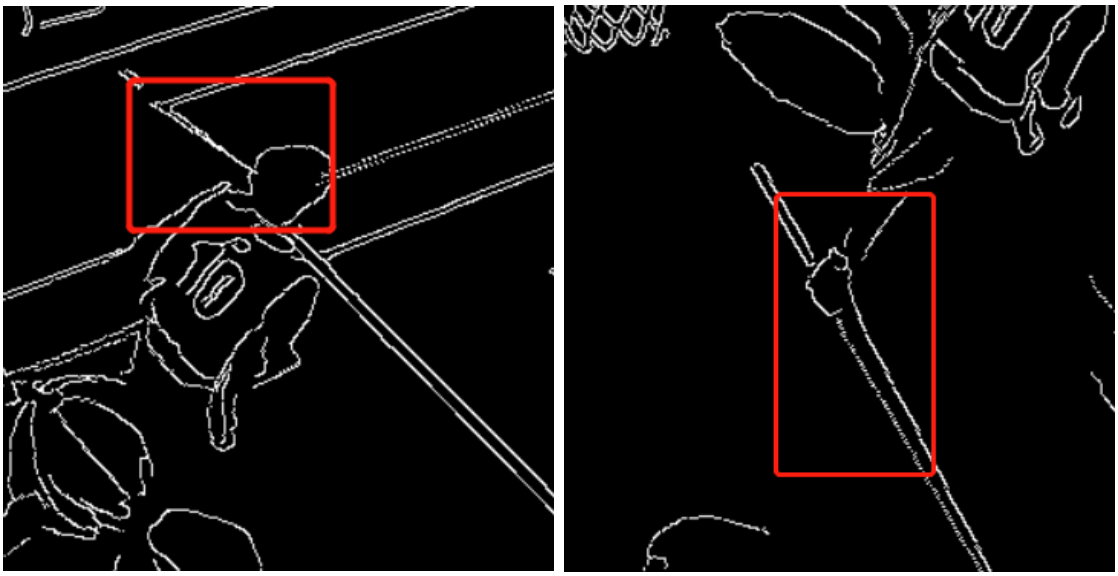

Figure 5.11: Artifacts generated from edge tracking method, the edges within the red bounding box are expected to represent two distinct stationary lines. However, due to limitations in the edge linking process, the lines denoting the stationary field markings are unintentionally dragged by the edges associated with the moving foreground.

Figure 5.10, the edges of the athlete's head should be separated from the edge on the sports field. However, due to factors such as the photography angle, detection process and using the Canny edge algorithm, these edges appear connected. When predicting the parameters for affine transformation, this edge, which combines both foreground and background elements, will exhibit identical motion characteristics. Consequently, this can result in the motion of the foreground interfering with the motion of the background. Figure 5.11 illustrates the consequences of the erroneous connections caused by this type of edge. Parts of the background that should ideally remain stationary exhibit unexpected motion due to the movement of the foreground. To enhance the connection method, the next chapter will introduce an edge-splitting approach to address the edge connection issue, which will also serve as a subject of future research.

6 Additional Method and Future Work

While the method introduced in this study has yielded notable advancements, particularly in the domain of edge tracking, where its accuracy surpasses that of several traditional optical flow approaches. However, due to the challenges in distinguishing between foreground and background using edge connectivity methods, this chapter introduces an approach to address this issue. The key lies in how to separate the edges of the foreground from those of the background. On the other hand, edge tracking method is even outperforming the deep learning-based optical flow method, FlowNet2, it is essential to acknowledge its significant computational complexity, primarily attributed to the key point allocation process. Hence, this chapter seeks to investigate the feasibility of achieving edge tracking without relying on key points.

6.1 Methods Introduction

There are two methods will be proposed in this chapter. The first approach is aimed at addressing the issue of inaccurate edge projection tracking, as mentioned earlier, which results from the imprecision in edge connections. Based on the previous discussion, the current method of connecting edges cannot precisely distinguish whether a certain segment of an edge belongs to the foreground or the background. Therefore, there is a need for a refinement method that divides edges into smaller segments and then tracks these smaller edge segments. In concept, this approach can effectively separate the edges of the foreground from those of the background, reducing inaccuracies in edge tracking caused by incorrect connections. Second method involves initially identifying corresponding edges in edge set Ω in two images. In theory, there isn't a significant change in the edges between two frames. Therefore, it's possible to directly determine the motion relationship between the edges in these two frames based on the existing edge points. This relationship typically includes rotation and translation. Subsequently, we employ an affine transformation to evaluate the warping parameters by the points on the edges, ultimately achieving edge tracking. The purpose of this approach is to reduce the complexity of key point matching. It achieves this by establishing correspondences between edges in the current frame and their counterparts in the previous frame. By utilizing a matching algorithm, the corresponding edges between the two frames are inferred. Subsequently, the motion between these corresponding edges is calculated based on their respective positions.

6.2 Edge Splitting

Based on the preceding discussion, due to the limitations of the edge connectivity approach in accurately distinguishing whether a particular segment of an edge belongs to the foreground or background, there is a need for a refinement method to break down the edges into smaller segments. These smaller edge segments can then be individually tracked for more precise analysis.

6.2.1 Splitting Edges Using Contour Detection Method

The initial three steps of this method are identical to the initial three steps of the method described in Chapter 3, which are as follows:

1. Use Canny edge method extract the edges.
2. Use the detected edge to find different edges with the connected element search method, add the edges to an edge set.
3. Use LK find the corresponding key points between 2 frames' edges.

In the fourth step, we will utilize key points detected around edges for edge splitting, creating a smaller edge set. Since estimating affine transformation parameters requires a minimum of three key points, contour-based methods are employed to locate these key points along edges. Whenever the algorithm identifies three key points along an edge, that segment of the edge is included in the smaller edge set. This approach ensures that each small edge segment possesses an independent motion, thereby reducing the impact of detected foreground-background edge connections. After incorporating these refined edges into the edge set, the next

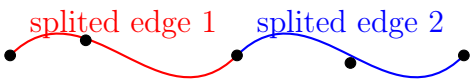

Figure 6.1: Edge splitting method, if the contour search method detects three key points along an edge, it separates this segment of the edge from the preceding and succeeding edges, creating an independent segment of the edge.

step involves combining these fine-grained edges and utilizing the information from the three key points surrounding each of these small edge segments to compute the affine transformation parameters for the motion of this particular segment of the edge. The result of dividing an edge into multiple smaller segments is depicted in Figure 6.2. After segmenting the edges, as in the two methods described in Chapter 3, affine transformation matrices are computed based on the corresponding key points detected on each edge in the two consecutive frames. These transformation matrices are calculated with respect to the key point positions on

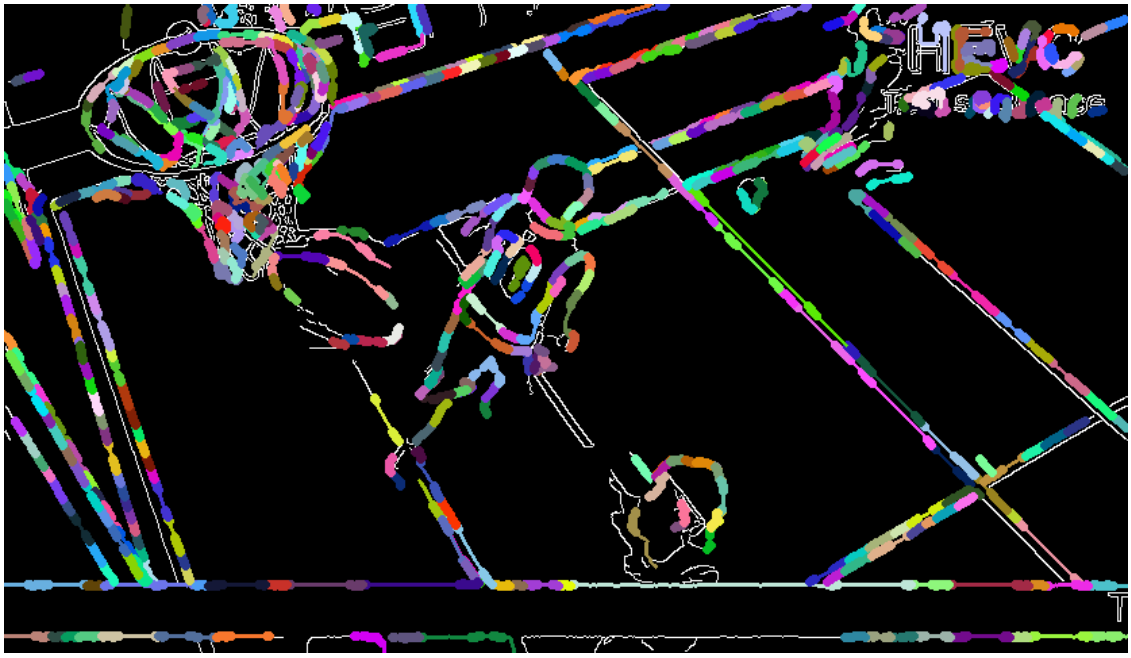

Figure 6.2: Separated segments of edges are represented in different colors, with each individual segment's edges having their own set of three key points.

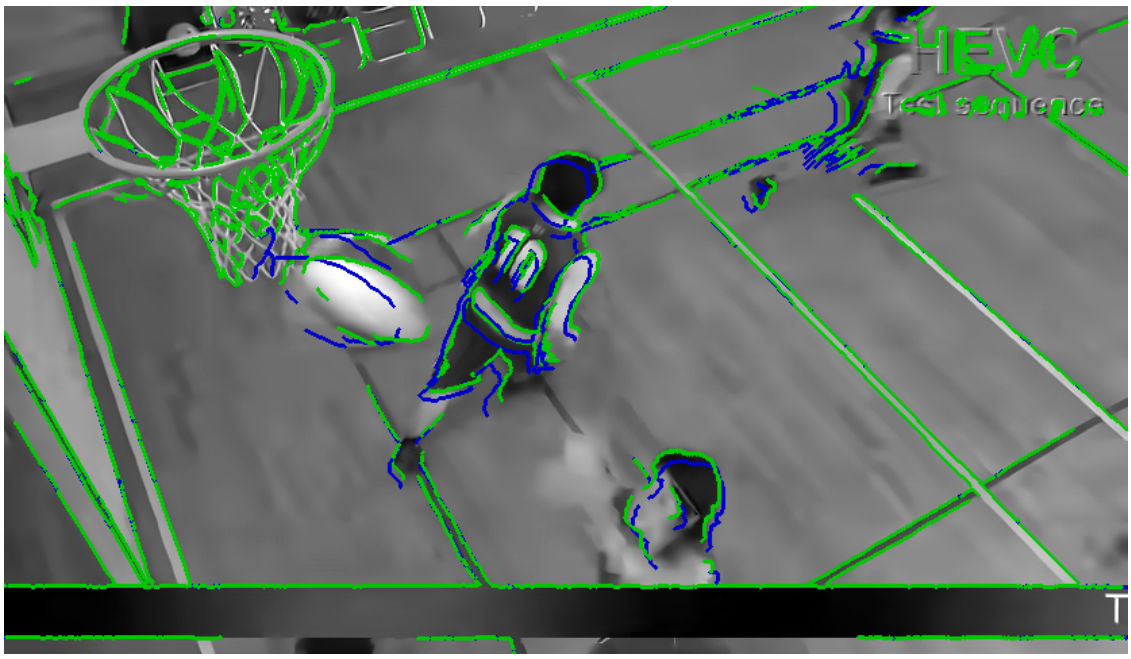

Figure 6.3: Edge splitting tracking result

each separated edge from the previous frame. Subsequently, these transformation matrices are applied to warp the small edge segments, inferring the positions of the edges in the current frame. The above is an overview of the edge splitting method. This algorithm is generally similar to the two methods in Chapter 3, with the key distinction being the refinement of edges obtained through the second edge

connection method. The results are depicted in Figure 6.3. To better examine

Figure 6.4: Edge splitting tracking result in edge image, as mentioned in the previous method descriptions, the blue edges represent the edges from the previous frame, while the green edges are the edges of the current frame inferred based on affine transformation and the edges from the previous frame.

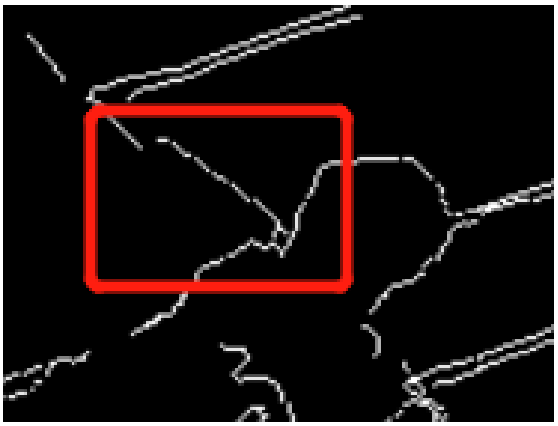

Figure 6.5: Edge splitting tracking, a part of result edge image

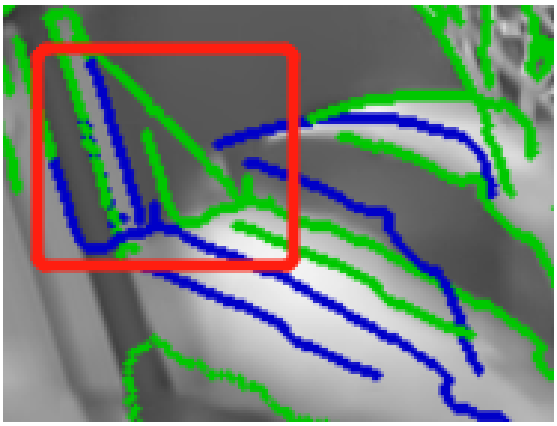

Figure 6.6: Edge splitting tracking, a part of warping image

the results obtained by this method, it is essential to directly observe the generated edge image, as shown in Figure 6.4. In this approach, each edge is associated with only three key points, and these key points are non-repetitive during the assignment process. This results in sparser and fewer tracked edges compared to the first two methods. Furthermore, since key point tracking introduces noise, each small

edge relies on just three key points to determine its motion. Any noise introduced by these three key points during tracking can lead to highly inaccurate predictions of affine transformation parameters, as illustrated in Figure 6.4. The upper-right corner of this figure demonstrates a chaotic tracking scenario, primarily due to the limited number of key points associated with each small edge. This inaccuracy in key point tracking significantly impacts the quality of the results. To better assess the effectiveness of this method, we will examine the details of the results. As shown in Figure 6.5, compared to the left image in Figure 5.11, it is evident that the foreground and background have indeed been separated at this particular location. However, due to the influence of foreground motion tracked by key point tracking, the independently separated edge also exhibits motion. Figure 6.6 illustrates the use of the edge separation method. However, due to the relatively random nature of the edge splitting approach, there are instances where some foreground and background edges are not fully separated. As a result, this algorithm encounters several challenges. To accurately separate foreground and background edges, further in-depth research is required in the future. The Edge Splitting Tracking algorithm can be summarized as Algorithm 5.

Algorithm 5: Edge Splitting Tracking Method

Input: $GrayImg1$, $GrayImg2$
Output: $AffinedEdges$

$EdgesImg1 \leftarrow CannyEdge(GrayImg1)$;
$EdgesImg2 \leftarrow CannyEdge(GrayImg2)$;
$KeyPointsSet \leftarrow$ Null;
$KeyPoints1, KeyPoints2 \leftarrow KLT(EdgesImg1, EdgesImg2)$;
$KeyPoints1, KeyPoints2 \leftarrow KLT(EdgesImg1, EdgesImg2)$;
$KeyPoints1, KeyPoints2 \leftarrow FindContour(EdgesImg1, KeyPoints1)$;
if *Find 3 key points kps on edge* **then**
    $KeyPointsSet \leftarrow kps$;
    Separate a segment of the edge e passing through these three points;
    $\Omega \leftarrow e$;
end
$n \leftarrow \text{len}(\Omega)$;
for $i = 1$ *to* n **do**
    $AffineMatrix_i \leftarrow \text{leastsquare}(KeyPointsSet_i, KeyPoints2_i)$;
    $ReprojectionError \leftarrow$
     $||KeyPointsSet_i \cdot AffineMatrix_i - KeyPoints2||^2$;
    if $ReprojectionError \leq thres\ \theta$ **then**
        $AffineMatrix_i \leftarrow \text{LevenbergMarquardt}(ReprojectionError)$;
    end
    $AffinedEdges \leftarrow \Omega_i \cdot AffineMatrix_i$;
end

6.3 Edge Tracking Without Key Points

In order to reduce the complexity of edge tracking, this approach adopts a bold assumption: it refrains from utilizing key points from two frames for the estimation of motion parameters associated with key points or edges. Instead, it directly identifies corresponding edges in the two frames and employs information such as shape and color characteristics of these edges to estimate their positions.

6.3.1 Edge Matching

In order to achieve edge matching, the shape, position, and color of the edges in two frames should be considered. To begin, Gaussian distributions are employed to represent edge positions. This entails calculating the mean of the coordinates along the edge to represent its position and using the variance to describe its shape. This method, as previously discussed in Chapter 3, enables the establishment of position and shape information for each edge using simple Gaussian distributions. Additionally, it is essential to extract the color information surrounding the edges to understand the colors in the vicinity of each edge. This can be accomplished by employing a sliding window approach to compute the mean of pixel values around the edge. The Algorithm 6 shows how to extract colors around edges. It starts by detecting edges within the image. Next, it slides a small window along these edges, covering the pixels around the edge points. For each pixel inside this window, it calculates the average color, capturing a representative color from the surrounding area. This process helps in extracting color information around the edges, which, in turn, aids in analyzing the color correlation between corresponding edges in two frames.

Algorithm 6: Get Color Information from Edges

Input: Img
Output: Edge map with color points

```
Detect edges using Canny edge detection;
Convert the edge map to color;
Initialize window size and an empty color list;
for each pixel (y, x) in the edge map do
    if edge strength at (y, x) is greater than 0 then
        Extract a window of pixels around (y, x);
        if window size is as expected then
            Calculate the mean color of the pixels in the window;
            Append the mean color to the color list;
        end
    end
end
```

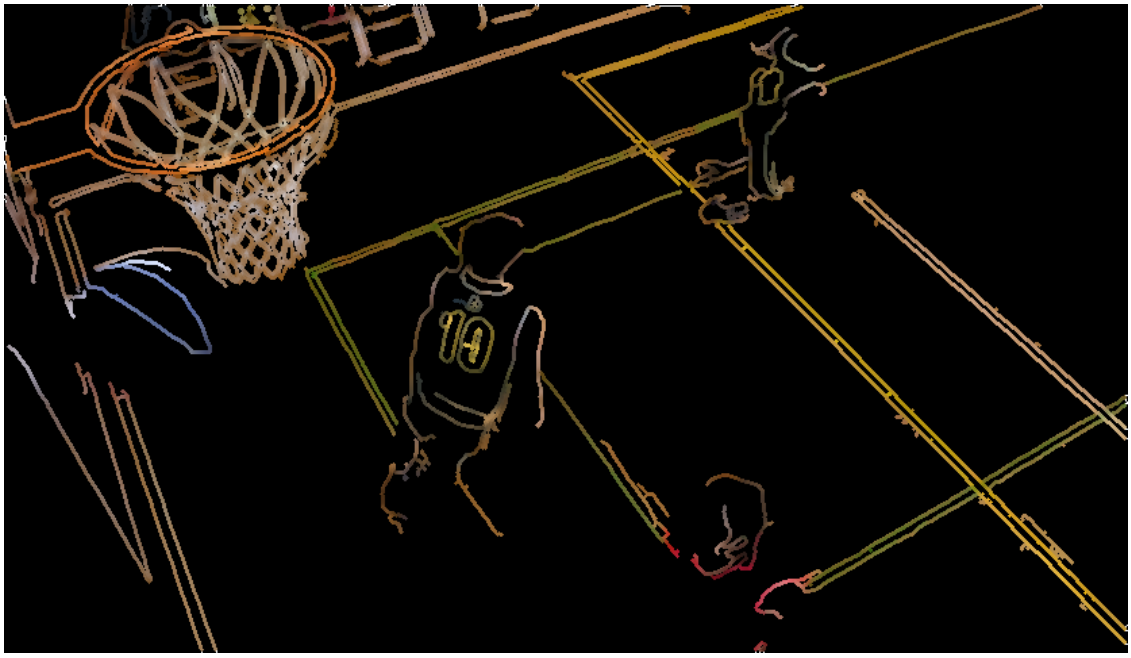

Figure 6.7: Get color information surrounding edges

The results of color information extraction around the boundaries can be observed in Figure 6.7. After extracting the location, shape, and color information of edges, we proceed to compare the similarity of these three variables for every edge between two frames. To accomplish this task, the Hungarian algorithm [34] should be introduced to match edges between the two frames by utilizing the extracted information. One of the main advantages of the Hungarian algorithm is its efficiency. Compared with the brute force allocation algorithm, the Hungarian algorithm can find the global optimal solution in a shorter time without trying all possible combinations. This efficiency is particularly significant in large-scale allocation problems because it can find the optimal resource allocation solution in a reasonable time.

Before using the Hungarian algorithm, a **CM** (cost matrix) should be constructed. Assuming there are two frames, with the first frame containing m edges and the second frame containing n edges, the shape of this cost matrix should be $m \times n$. Each element $\mathbf{CM}_{i,j}$ in the cost matrix represents the cost of assigning one edge from the first frame to each edge in the second frame. Where i and j represent the elements in the cost matrix found at the intersection of the i-th row and the j-th column.

The first component in the cost matrix is determined by the positional difference pd between edges in two consecutive frames. In theory, the motion of edges between these frames should exhibit continuity. For instance, an edge that is on the far left of the image in the first frame logically cannot jump to the far right of the image in the second frame. Therefore, in this aspect of loss determination, when edges in the two frames are closer in distance, there is a higher likelihood that they represent the same edge, resulting in a smaller cost.

In the cost matrix, the second element determining the loss is related to shape ls.

As mentioned earlier, the shape of edges is represented by the covariance matrix with a Gaussian distribution. In this matching task, the changes in edge shape between consecutive frames are not expected to be significant. For instance, an edge should not transition from a circular shape in the first frame to a square shape in the second frame. Even if there is some shape variation, it should be gradual over several frames before reaching a completely different shape. If the covariance matrices of two edges in the two frames are quite similar, it can be assumed that they are roughly the same edge. In other words, the shape-based loss value would be relatively small in such cases.

The third factor that determines the loss in the cost matrix is color lc. After extracting the color of each edge, the color of each edge in the first frame is compared to the color of each edge in the second frame. Similar to assessing angles, the colors around the edges theoretically should not change significantly between the two frames. If the similarity in color between the edges in the two frames is high, it can be reasonably assumed that they represent the same edge. By combining the three aforementioned matching criteria, we can establish an equation to calculate the loss value of each element $\mathbf{CM}_{i,j}$ in the cost matrix. This loss value can be defined by the following formula

$$\mathbf{CM}_{i,j} = w_1 pd_{i,j} + w_2 ls_{i,j} + w_3 lc_{i,j}. \tag{6.1}$$

Because these three factors have different units, distance is measured in pixels, shape is represented by a covariance matrix in Gaussian distribution, which is purely numerical without units, and color is measured in pixel values. To create a loss function that combines these three factors, it is essential to introduce weighting factors. These weighting factors ensure that each factor contributes differently to the overall comparison, ultimately yielding more accurate and meaningful results. Therefore, this is an empirical formula based on practical coding experience, and it has been found that setting w_1, w_2, and w_3 to 0.4, -9, and 0.02, respectively, yields the best results. When using the Hungarian algorithm, it's important to note that this algorithm automatically seeks one-to-one matches. Even when the losses are significant, the Hungarian algorithm doesn't exclude them. Therefore, it's advisable to set a threshold τ to filter out matches with considerably high losses. From Table

$Edges^1$ \ $Edges^2$	$Edge_1^2$	$Edge_2^2$	$Edge_3^2$	$Edge_4^2$	$Edge_5^2$
$Edge_1^1$	0.4	2.1	3.2	0.6	0.8
$Edge_2^1$	2.3	0.3	2.5	2.5	2.2
$Edge_3^1$	0.5	2.3	2.4	1.0	0.8

Table 6.1: Hungarian algorithm, The elements highlighted in green in the matrix represent the minimum loss, indicating the best global matching positions, exclude an element if it is above a threshold, threshold $\tau = 0.9$.

6.1, the globally optimal solution is identified by the Hungarian algorithm, which implies that the matching with the least overall loss is achieved.

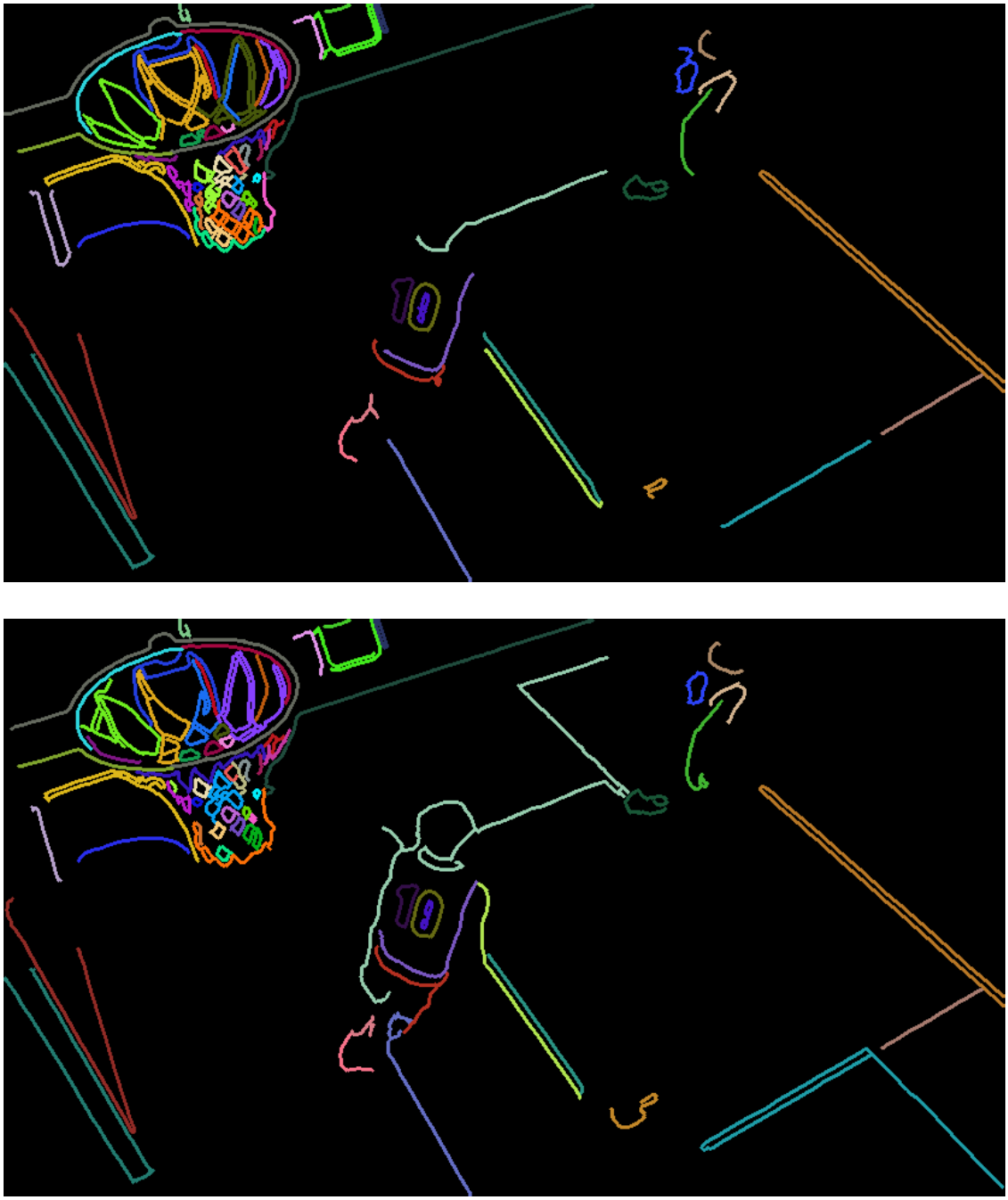

Figure 6.8: Macthed edge in 2 frames, edges of the same color in two frames represent the same aligned edge. However, it's important to note that in some cases, the length of these edges can significantly vary between the two frames.

However, in cases where no match can be found, it is signified that this edge in the current frame is significantly dissimilar to any edge in the first frame, with a loss greater than a certain threshold. This could be due to the fact that this edge has only been detected in the second frame, and as a result, this match should be excluded. Figure 6.8 displays the edges matched by our method in two frames.

It's evident that the corresponding edges in the two frames make sense. While the lengths of these matched edges differ between the frames, the matching results are quite satisfactory. To the naked eye, it's nearly impossible to discern any matching errors. The significant differences in the lengths of corresponding edges between the first frame and the second frame can be attributed to two main factors. Firstly, the Canny Edge algorithm may detect the same edge in both frames but yield different lengths. Secondly, during the process of associating edges into edge sets, the varying connectivity of edges in each frame results in different lengths of the detected connected edges. This issue is a critical challenge associated with this method. Once matching edges are identified, an evaluation of their motion, including rotation and displacement, is performed for each individual edge. In the next section, this issue will be discussed in detail. The problem is encountered due to differences in the lengths of corresponding edges, which make it difficult to establish a clear relationship between edge positions across two frames. This complexity presents a significant challenge when trying to define an optimization function for predicting the affine transformation matrix.

6.3.2 Motion Estimation

After obtaining corresponding edges from two frames, the motion of these edges, including rotation and translation, needs to be predicted. This process still relies on the affine transformation parameter matrix operations used in previous methods. In theory, if the length difference of edges between the two frames is not significant, it is possible to sample an equal number of points on these corresponding edges for approximate point matching. As depicted in Figure 6.9, when two corresponding edges are identified between two frames, the calculation of parameters for an affine transformation can be achieved by identifying five corresponding points on each of these corresponding edges.

Figure 6.9: Corresponding edges in 2 frames, the green points are sampled from the edges.

As shown in Figure 6.10, the blue and green edges represent two corresponding edges in first frame and second frame that clearly illustrate the position, length, and size changes of the edges between the two frames. After extracting four corresponding points from these edges, an affine transformation matrix was calculated. This matrix was then used to project the edges onto the positions of the edges in the second frame, as depicted in Figure 6.11.

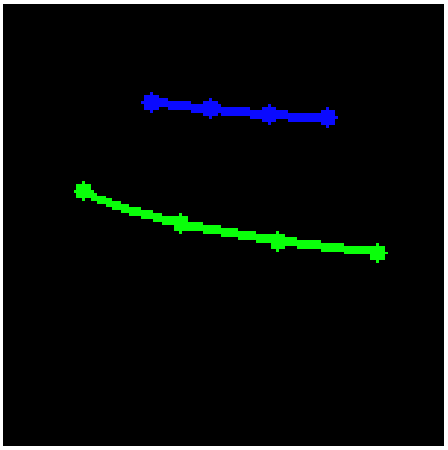

Figure 6.10: Corresponding edges

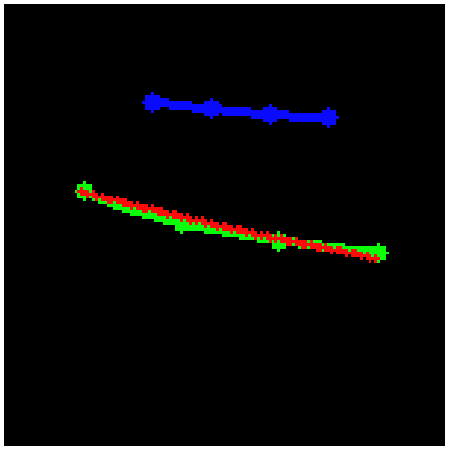

Figure 6.11: Projection result

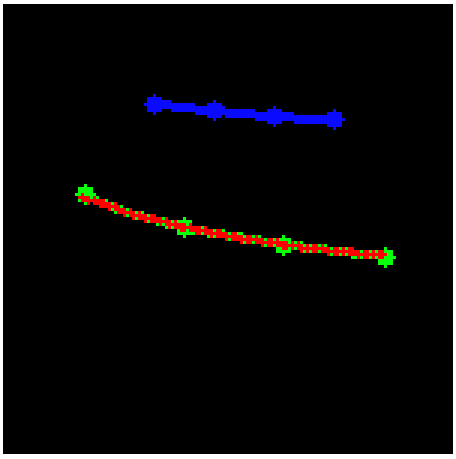

Figure 6.12: Optimized projection

However, in Figure 6.11, after the affine transformation, the edges do not perfectly align with the edges in the second frame. Therefore, at this stage, the parameters of the affine transformation matrix were optimized using the Levenberg-Marquardt algorithm. The results of this optimization, as shown in Figure 6.12, demonstrate that after optimization, the alignment of edges from the first frame to the second frame is significantly improved for greater accuracy. However, in many instances, a significant difference in edge lengths between corresponding edges was encountered, as illustrated in Figure 6.13. This issue prevents the approximate matching of corresponding points sampled along the edges since the sampled points are too far apart. Consequently, the computed affine transformation matrix becomes highly inaccurate. The primary challenge posed by the new algorithm is the inability to locate corresponding points along two corresponding edges, which is crucial for precise parameter calculation.

This phenomenon arises from the fact that the continuity of edges detected by the Canny edge algorithm varies between frames. For example, in the first frame, one edge might be continuous, while in the second frame, due to object motion,

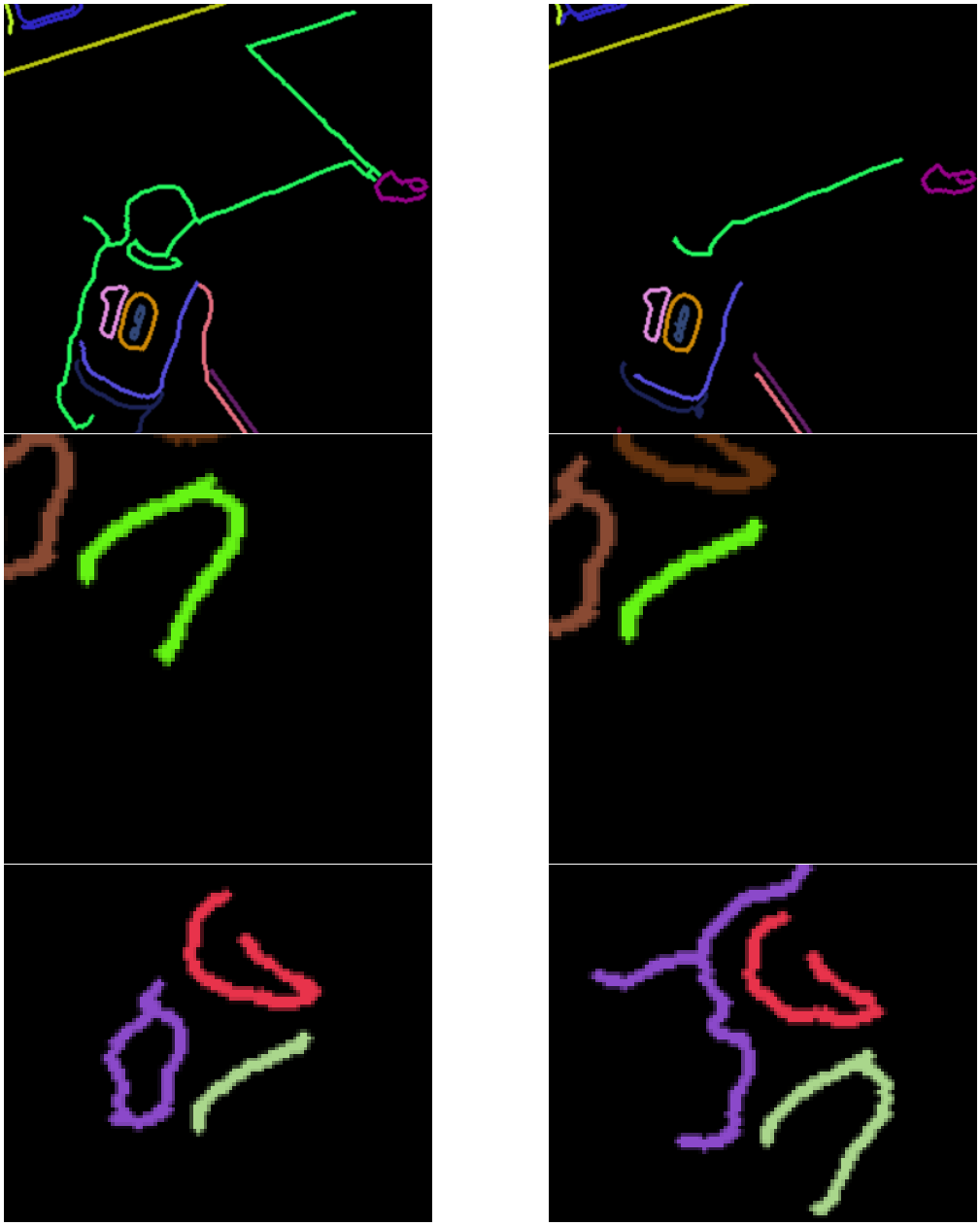

Figure 6.13: Corresponding edges just include a part of similarities

changes in lighting, and other factors, this same edge might be detected as two discontinuous segments. Consequently, this results in a substantial difference in edge lengths between corresponding edges in the two frames.

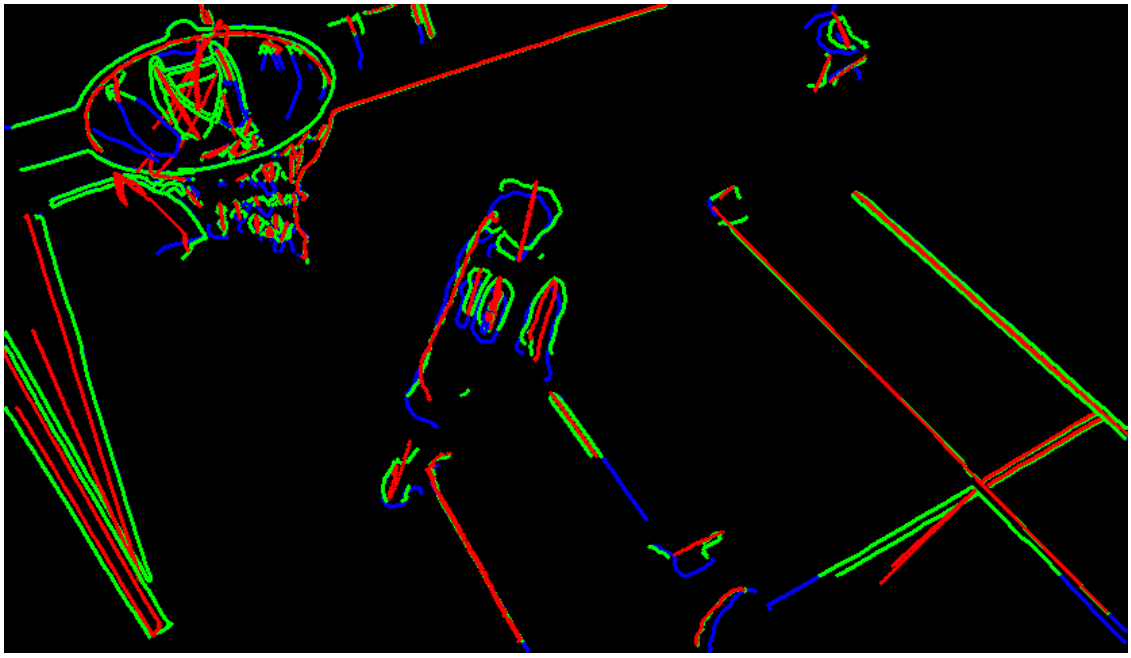

Figure 6.14: Result of additional method, red edges are affined edges, blue edge is from the previous frame, and the green edge is from the current frame.

Figure 6.14 illustrates the inaccurate outcomes of this tracking approach. The red edges represent the edges after affine transformation, while the blue and green edges correspond to the edges in the first and second frames. It is evident that the results of this method are highly inaccurate due to the inability to accurately locate corresponding points on the edges between the two frames. Therefore, if improvements are to be made in this method, it is crucial to establish a fundamental approach for achieving more accurate correspondence of edge points between the two frames. This will be a key focus for future research in this paper.

7 Conclusions

This paper focuses on the analysis of edges in coded videos, addressing the issues of artifacts and distortions that typically arise in coded videos. Initially, the key point tracking algorithm is employed to compare the effectiveness of tracking key points in different videos under varying encoding qualities. It is observed that as video quality decreases, the accuracy of key point tracking diminishes. However, even at the lowest video quality, the tracking errors amount to an average of only 2 to 3 pixels, which falls within an acceptable range. Due to the impact of video distortion, certain key points become challenging to track, resulting in the loss of their trajectories. Nevertheless, this issue is effectively mitigated by re-detecting key points in each frame. Following the evaluation of key point tracking, key points are spatially distributed, and their motion between frames is utilized to calculate edge motion, encompassing both rotation and translation.

After assigning each key point to its adjacent or nearest neighbor edge, affine transformations are inferred for the motion of key points based on the motion of key points between these two frames. Once the affine transformation parameters for each edge are obtained, warping calculations are applied to the edges. This enables the inference of the positions of edges in the current frame based on the motion of edges and key points from the previous frame. The edge tracking algorithm also encompasses two distinct key point assignment methods. The first method involves establishing a Gaussian distribution for each edge, computing Mahalanobis distances, and determining whether a key point belongs to the edge based on these Mahalanobis distances. The second method, on the other hand, determines whether a key point lies on a particular edge by calculating the Euclidean distance from the point to the edge. The former method boasts lower computational complexity but lower accuracy, while the latter method exhibits higher computational complexity but higher accuracy.

In this study, algorithm evaluation was conducted on multiple videos such as Basketball Drill, BQ Square, and Cactus. By computing the mean squared error (MSE) against uncoded videos and comparing with other optical flow algorithms like FlowNet2 and Farneback optical flow, it was observed that this algorithm achieved approximately 10% to 20% lower MSE compared to the two aforementioned optical flow methods. After implementing this edge tracking algorithm, it becomes possible to apply it for frame interpolation and prediction of edges within videos. This algorithm provides motion information for each edge in the image between two frames. By adjusting the speed of these edges, we can achieve frame interpolation by slowing down the motion or predict the positions of each edge in the next frame based on the motion between the current frame and the previous frame when accelerating. However, a limitation of this algorithm lies in its inabil-

ity to guarantee an effective separation of background and foreground edges during the edge connection process. As a result, it can lead to distortions and inaccurate motion estimation at the boundaries where background and foreground objects are in motion.

To address the issue of separating foreground and background edges, Chapter Six attempted an edge splitting approach. However, the results were not entirely satisfactory, and further optimization is required as part of future work. The second part of Chapter Six explored a method that avoids key point tracking and allocation, focusing directly on the analysis of edges to compute their motion. This approach employed the Hungarian algorithm to obtain correspondences between edges in two frames. However, due to variations in the shapes and lengths of edges between the two frames caused by the Canny edge detection and edge linking methods, finding corresponding points between edges proved challenging, making motion estimation difficult. This aspect also presents opportunities for future optimization.

In summary, this thesis has implemented an edge tracking algorithm that effectively computes the motion of edges in encoded videos. Nonetheless, there are areas in need of refinement, and future work will involve conducting additional experiments and research to enhance accuracy and robustness.

Bibliography

[1] B. D. Lucas and T. Kanade, "An iterative image registration technique with an application to stereo vision," in *IJCAI'81: 7th international joint conference on Artificial intelligence*, vol. 2, pp. 674–679, 1981.

[2] E. Ilg, N. Mayer, T. Saikia, M. Keuper, A. Dosovitskiy, and T. Brox, "Flownet 2.0: Evolution of optical flow estimation with deep networks," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 2462–2470, 2017.

[3] H. Series, "Audiovisual and multimedia systems, infrastructure of audiovisual services—coding of moving video, advanced video coding for generic audiovisual services," *International Telecommunication Union*, vol. 41, p. 131, 2003.

[4] D. Grois, T. Nguyen, and D. Marpe, "Coding efficiency comparison of av1/vp9, h. 265/mpeg-hevc, and h. 264/mpeg-avc encoders," in *2016 Picture Coding Symposium (PCS)*, pp. 1–5, IEEE, 2016.

[5] D. Mukherjee, J. Han, J. Bankoski, R. Bultje, A. Grange, J. Koleszar, P. Wilkins, and Y. Xu, "A technical overview of vp9—the latest open-source video codec," *SMPTE Motion Imaging Journal*, vol. 124, no. 1, pp. 44–54, 2015.

[6] Y. Chen, D. Murherjee, J. Han, A. Grange, Y. Xu, Z. Liu, S. Parker, C. Chen, H. Su, U. Joshi, *et al.*, "An overview of core coding tools in the av1 video codec," in *2018 picture coding symposium (PCS)*, pp. 41–45, IEEE, 2018.

[7] S. Vivienne, B. Madhukar, and J. Gary, "High efficiency video coding (hevc): algorithms and architectures," *Cham, Switzerland: Springer*, 2014.

[8] F. Brand, J. Seiler, and A. Kaup, "Intra frame prediction for video coding using a conditional autoencoder approach," in *2019 Picture Coding Symposium (PCS)*, pp. 1–5, IEEE, 2019.

[9] S. D. Kim, J. Yi, H. M. Kim, and J. B. Ra, "A deblocking filter with two separate modes in block-based video coding," *IEEE transactions on circuits and systems for video technology*, vol. 9, no. 1, pp. 156–160, 1999.

[10] D. Fleet and Y. Weiss, "Optical flow estimation," in *Handbook of mathematical models in computer vision*, pp. 237–257, Springer, 2006.

[11] B. Galvin, B. McCane, K. Novins, D. Mason, S. Mills, *et al.*, "Recovering motion fields: An evaluation of eight optical flow algorithms.," in *BMVC*, vol. 98, pp. 195–204, Citeseer, 1998.

[12] B. K. Horn and B. G. Schunck, "Determining optical flow," *Artificial intelligence*, vol. 17, no. 1-3, pp. 185–203, 1981.

[13] G. Farnebäck, "Two-frame motion estimation based on polynomial expansion," in *Image Analysis: 13th Scandinavian Conference, SCIA 2003 Halmstad, Sweden, June 29–July 2, 2003 Proceedings 13*, pp. 363–370, Springer, 2003.

[14] J. K. Suhr, "Kanade-lucas-tomasi (klt) feature tracker," *Computer Vision (EEE6503)*, pp. 9–18, 2009.

[15] P. Bagherpour, S. A. Cheraghi, and M. bin Mohd Mokji, "Upper body tracking using klt and kalman filter," *Procedia Computer Science*, vol. 13, pp. 185–191, 2012.

[16] M. I. Ribeiro, "Kalman and extended kalman filters: Concept, derivation and properties," *Institute for Systems and Robotics*, vol. 43, no. 46, pp. 3736–3741, 2004.

[17] C. M. Bukey, S. V. Kulkarni, and R. A. Chavan, "Multi-object tracking using kalman filter and particle filter," in *2017 IEEE International Conference on Power, Control, Signals and Instrumentation Engineering (ICPCSI)*, pp. 1688–1692, IEEE, 2017.

[18] X. Li, K. Wang, W. Wang, and Y. Li, "A multiple object tracking method using kalman filter," in *The 2010 IEEE international conference on information and automation*, pp. 1862–1866, IEEE, 2010.

[19] M. Elhoseny, "Multi-object detection and tracking (modt) machine learning model for real-time video surveillance systems," *Circuits, Systems, and Signal Processing*, vol. 39, pp. 611–630, 2020.

[20] L. Rosenthaler, F. Heitger, O. Kübler, and R. von der Heydt, "Detection of general edges and keypoints," in *Computer Vision—ECCV'92: Second European Conference on Computer Vision Santa Margherita Ligure, Italy, May 19–22, 1992 Proceedings 2*, pp. 78–86, Springer, 1992.

[21] K. Mikolajczyk, T. Tuytelaars, C. Schmid, A. Zisserman, J. Matas, F. Schaffalitzky, T. Kadir, and L. V. Gool, "A comparison of affine region detectors," *International journal of computer vision*, vol. 65, pp. 43–72, 2005.

[22] R. A. Jarvis, "A perspective on range finding techniques for computer vision," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, no. 2, pp. 122–139, 1983.

[23] K. Hinckley, J. Tullio, R. Pausch, D. Proffitt, and N. Kassell, "Usability analysis of 3d rotation techniques," in *Proceedings of the 10th annual ACM symposium on User interface software and technology*, pp. 1–10, 1997.

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

[25] W. Luo, X. Zhao, and T.-K. Kim, "Multiple object tracking: A review," *arXiv preprint arXiv:1409.7618*, vol. 1, p. 1, 2014.

[26] Y. Hung and W. Tang, "Projective reconstruction from multiple views with minimization of 2d reprojection error," *International Journal of Computer Vision*, vol. 66, pp. 305–317, 2006.

[27] B. Triggs, P. F. McLauchlan, R. I. Hartley, and A. W. Fitzgibbon, "Bundle adjustment—a modern synthesis," in *Vision Algorithms: Theory and Practice: International Workshop on Vision Algorithms Corfu, Greece, September 21–22, 1999 Proceedings*, pp. 298–372, Springer, 2000.

[28] P. Teunissen, "Nonlinear least squares," 1990.

[29] R. G. Pratt, C. Shin, and G. Hick, "Gauss–newton and full newton methods in frequency–space seismic waveform inversion," *Geophysical journal international*, vol. 133, no. 2, pp. 341–362, 1998.

[30] A. Ranganathan, "The levenberg-marquardt algorithm," *Tutoral on LM algorithm*, vol. 11, no. 1, pp. 101–110, 2004.

[31] W. Liu, W. Luo, D. Lian, and S. Gao, "Future frame prediction for anomaly detection–a new baseline," in *Proceedings of the IEEE conference on computer vision and pattern recognition*, pp. 6536–6545, 2018.

[32] D. G. Lowe, "Distinctive image features from scale-invariant keypoints," *International journal of computer vision*, vol. 60, pp. 91–110, 2004.

[33] M. A. Fischler and R. C. Bolles, "Random sample consensus: a paradigm for model fitting with applications to image analysis and automated cartography," *Communications of the ACM*, vol. 24, no. 6, pp. 381–395, 1981.

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