Delft University of Technology
Master of Science Thesis in Embedded Systems

High-speed moving object recognition based on event camera

Jiacong Li

TU Delft Delft University of Technology

High-speed moving object recognition based on event camera

Master of Science Thesis in Embedded Systems

Embedded Systems Group
Faculty of Electrical Engineering, Mathematics and Computer Science
Delft University of Technology
Van Mourik Broekmanweg 6, 2628 XE Delft, The Netherlands

Jiacong Li

20th Aug 2024

Author
Jiacong
Title
High-speed moving object recognition based on event camera
MSc Presentation Date

27th Aug 2024

Graduation Committee

Qing Wang	Delft University of Technology
Chang Gao	Delft University of Technology

Abstract

Unmanned aerial vehicles (UAVs), often referred to as autonomous drones, are becoming more and more prevalent in our daily lives. Drones are usually equipped with traditional frame-based cameras and have functions like object detection. However, the high energy consumption of frame-based cameras presents a challenge to drone endurance. In addition, due to its own hardware limitations, problems such as blurring and deformation will occur when capturing high-speed moving objects. In contrast, event cameras, as one of the latest neural technology cameras, have the characteristics of low power consumption, low latency, and sensitivity to high-speed objects. This makes them well-suited for integration into drone platforms. In this thesis, we introduce a method for high-speed moving object recognition based on event cameras. This approach involves enhancing objects with externally added "common features": fiducial markers, and employing a custom-developed deep learning neural network to detect these markers. These marks can carry some messages, such as the information of related objects. When these marks are detected, it is considered that objects are detected. We also developed a fiducial marker decoding method based on region segmentation to obtain the message content, thereby achieving interaction with the detected object. Evaluation results show that the proposed method has a low computing time of 26.9 ms, low storage and memory usages of 670 MB CPU memory and 750 MB GPU video memory, and a high accuracy of 77.9%, making it suitable for high-speed object recognition based on event cameras.

Preface

First and foremost, I would like to thank my supervisor Prof. Qing Wang, who generously agreed to supervise my thesis last December after the company I initially planned to complete my thesis with went bankrupt. Qing gave me a lot of advice and guidance throughout the thesis process, and provided a lot of help so that I could successfully complete my thesis. Thanks to Prof. Guohao Lan for lending me his event camera. Without it, my thesis work would have been much harder.

Special thanks go to my parents for their constant support and encouragement. And also, I appreciate all my friends from the CESE program; meeting you from different countries in the Netherlands has enriched my experience immensely.

To my friends from SMU, who have been with me on this journey to TUD for the past four and a half years, and Choisum, who brought unexpected joy into my life, thank you all for making my time in the Netherlands vibrant and full of life.

Finally, I would like to thank Tony, who I call my second supervisor in jest. His early guidance when I was first exploring deep learning was really helpful.

I hope each of you finds the path that fits you best and becomes the person you most aspire to be.

Jiacong Li

Delft, The Netherlands
20th August 2024

Contents

Chapter 1

Introduction

1.1 Motivation

An unmanned aerial vehicle (UAV), commonly known as an autonomous drone, has indirectly affected people's lives in the last decades. Such as aerial photography, which allows people to see the world from a different perspective, or is used for environmental monitoring to ensure the environment can be protected more efficiently. With the rapid development of drone technology, drones have gradually become closer to people's daily lives and have been applied widely by companies to save human resources, e.g., Amazon has applied drones for the parcel delivery business [2].

Although the application scenarios of drones are varied and highly convenient, the energy supply of drones has always been a problem since their invention. While equipping them with large, high-capacity batteries seems like a straightforward solution, due to the technical limitations of modern batteries, it's always costly and can add substantial weight. Usually, this will not significantly improve transportation efficiency. Especially for those application scenarios that require cargo loading, the maximum load capacity is also strictly limited. Based on this, Qing et al. [38] propose a method called 'hitchhike'. The main idea involves having a drone land on a moving vehicle that is traveling in the same direction. This way, the drone can hitch a ride without using its own power, saving energy during transit. But this method faces a significant problem: how to identify and distinguish vehicles that move in the same direction or with the same destination as the drone. Inspired by this, this thesis generalizes this problem to a broader range of application scenarios and focuses on exploring and solving the problem of **high-speed moving object recognition**.

When it comes to object recognition, the cameras for filming objects used by mainstream drones on the market today are traditional frame-based cameras. Although this type of camera can perfectly record the spatial details of the captured scenes, such as texture, color, etc., and the algorithms and neural network models designed for object recognition based on this camera are very mature. However, due to the frame rate limitations and CMOS hardware technology, ordinary frame-based cameras often cannot perform well in capturing high-speed moving objects or recording in low-light scenes. Therefore, the particular camera proposed by Lichtsteiner et al. [18], event camera, becomes the

first choice. The event camera, also known as the event vision sensor (EVS), has the advantage of high temporal resolution, low latency, wide dynamic range, and very low energy consumption, which is ideal for high-speed moving object recognition tasks on embedded platforms.

1.2 Research Problem and Objectives

With the background mentioned in Section 1.1, the research problem is formulated as:

> ***How to realize high-speed moving object recognition based on event camera?***

The research objectives can be formulated as follows:

To design and implement methods:

- *To detect and identify moving objects with an event camera*

  The existing frame-based cameras always have problems with high-speed moving object filming, such as blurring, distortion, and noise. While cameras designed for capturing high-speed objects exist, their heavy weight and large energy consumption make them less likely to become the primary choice for drone platforms. In contrast, the output of event cameras is clearer when dealing with high-speed moving objects. Therefore, this thesis firstly aims to develop a moving object detection method based on event cameras. And this method should be able to pinpoint the accurate position and edge information of the detected object. Detection is only a part of object recognition. We hope to know the types of objects when they are detected, so object identification is also an essential part of object recognition. Although there are already methods for object recognition using event cameras, we aim to develop an innovative way to achieve the same goal.

- *Be efficient to run on embedded system platforms*

  In the drone applications discussed, a common issue with conventional frame-based cameras is their high energy consumption. This consumption increases as the video capture frame rate increases. The low power consumption of event cameras makes them ideal for embedded platforms. However, most of the existing similar algorithms are not suitable for embedded platforms due to excessive resource requirements. To fully leverage this benefit, the high-speed object recognition method developed using event cameras must be further optimized specifically for these platforms.

- *Be scalable for various purpose usages*

  Contemporary object recognition methods are often highly specialized and tailored for certain types of tasks. These methods frequently utilize neural networks, which require substantial resources for training. When these models are designed for a single function, adapting them to new purposes can lead to significant computing resource waste. Consequently, there is a need for a more universal approach to object recognition that can efficiently handle diverse usages.

- *Be interactive with detected objects*

  Current object detection and identification algorithms typically yield binary outcomes, indicating whether an object is detected or specifying the object's label. However, this thesis proposes to develop a methodology in which the system can interact dynamically with the recognized object, allowing for the transmission of information through the object itself.

1.3 Contribution

This thesis proposes a methodology for high-speed moving object recognition based on event cameras. The contribution can be summarized as follows:

- A novel moving object recognition pipeline based on an event camera and fiducial marker.
- A new event dataset with a small volume, including the fiducial markers moving in different postures, can be used for training fiducial marker detection.
- A deep learning model called *EvMarkNet* trained on the aforementioned dataset and used for the detention of fiducial markers with different contents.
- A new method to decode optimized fiducial marker based on event camera.

The methodology is designed based on the optimized fiducial marker which will be introduced in Section 3.2. To the best of our knowledge, this is the first work that attempts to use optimized decode fiducial for both object recognition and data transmission. This is also the first work that attempts to detect fiducial markers in event cameras based on a neural network. Detailed design will be introduced in Chapter 3.

1.4 Structure

This report is dedicated to demonstrating the design, implementation, and optimization process of the method proposed in this thesis. Following this introductory chapter, there are four subsequent chapters. Each presents different aspects as outlined below:

- Chapter 2: Review and analysis of previous research related to the thesis topic.
- Chapter 3: Detailed introduction to the content and design concept of the method proposed in the thesis.
- Chapter 4: Evaluation of the performance of the proposed method. Qualitative and quantitative analysis and comparison to the evaluation results, as well as analysis of the advantages and disadvantages of the design.
- Chapter 5: Conclusion and summary of the proposed methods and analysis of the limitations and possible future development routes.

Chapter 2

Related Work

2.1 Event Camera

2.1.1 General Introduction

Event cameras, also known as event vision sensors (EVS) event-based cameras or "silicon retina" cameras, are a new type of camera proposed by Lichtsteiner et al. [18]. The name silicon retina comes from the similarity of these cameras to the retina in the human eye in the way they sense images. Each individual photoreceptor cell in the human eye responds independently to the light it detects, and the same principle is applied to event cameras. Each pixel on the optical sensor of an event camera produces an independent signal output.

To understand the advantages of event cameras, it is necessary to understand the working mode of traditional cameras first. The pixels of the sensor of traditional frame-based cameras output a unified and synchronized signal at a fixed time interval. This output is called a frame. The speed of output frames is called the frame rate. Because traditional cameras need to read and process signals from all pixels on the sensor, the frame rate is greatly limited. In contrast, the asynchronous output of event cameras can reduce the correlation between pixels, and the output of one pixel is not subject to other pixels. This evidently increases the temporal resolution of the event camera output. In addition, the pixels of event cameras do not capture the absolute brightness of light, but the change in light brightness. Figure 2.2 shows the essential difference between the output of the frame-based camera and the event camera. If the brightness at time t of a pixel at a location (x, y) is given by $I_{\mathrm{t,(x,\ y)}}$, an event is triggered when:

$$\| \log(I_{t+1,(\mathbf{x},\mathbf{y})}) - \log(I_{t,(\mathbf{x},\mathbf{y})}) \|_1 \geq \tau \tag{2.1}$$

Here τ is a threshold which will determine if an event is triggered or not. Each event outputs the following data:

$$e = \{(x, y), t, p\} \tag{2.2}$$

where p is the polarity of an event having a number of 1 or 0. The polarity represents the trend of brightness change. If the brightness changes from low to high, the polarity is equal to 1 and the event is called an on-event. In contrast, if brightness changes from high to low, then the polarity is equal to 0 and the

Figure 2.1: **The camera used in this thesis, DAVIS346.**

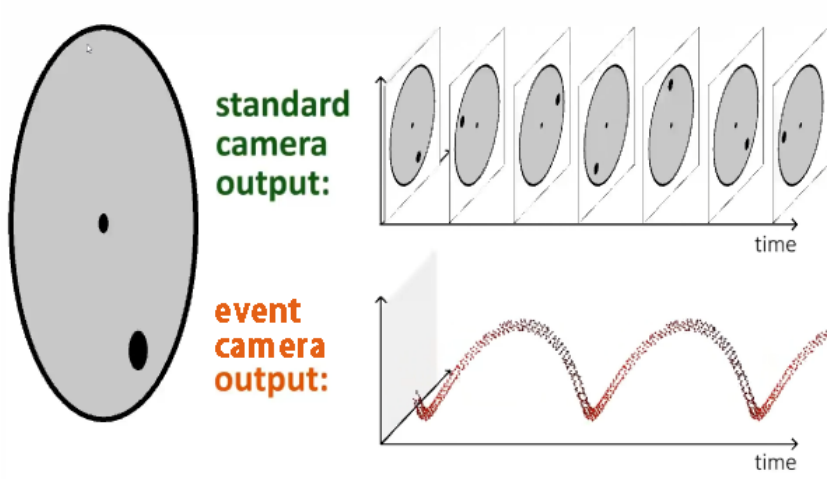


Figure 2.2: **The figure shows a rotating disk with a dot on it. For a frame-based camera, the output frame contains the entire disk and the dot. For an event camera, only the motion of the dot causes obvious brightness changes, so its output is only the moving dot. [30]**

event is called an off-event. In order to use these events efficiently, the event camera provides an output format, namely the event frame. The event camera accumulates all events within a time window T and finally outputs a data packet. The data frame can be defined as:

$$E\left(\mathbf{x}, T\right) = \left(\sum_{t=t_0}^{t_0+T} e\left(\mathbf{x}, t, p = \pm 1\right)\right) \tag{2.3}$$

Figure 2.3 depicts the visual output of an event frame.

2.1.2 Features

Advantages

As a state-of-the-art camera with a completely different working principle from the frame-based camera, the event camera has the following advantages:

- *High Temporal Resolution and low latency*: Because each pixel in an event camera detects changes independently and generates events asynchron-

Figure 2.3: **Comparison between the RGB frame of a frame camera and the event frame of an event camera. In the event frame, white and black pixels represent the on and off of events. Gray pixels indicate that no event has been triggered or the overlap area of on and off events. It can be clearly seen that the outline of the high-speed swinging racket is blurry in the RGB frame but clear in the event frame. [36]**

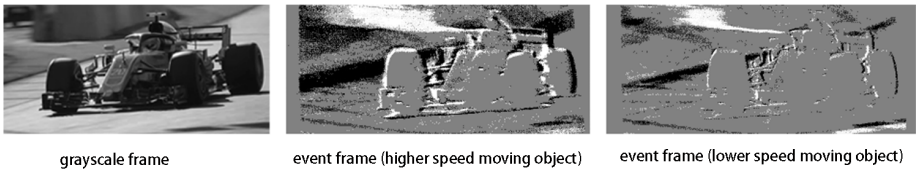


Figure 2.4: **Inconsistency is reflected in different outputs when the same object moves at different speeds.**

ously, the system can respond almost instantly to changes in the scene. These events can be output at intervals of up to 1 ms. This high temporal resolution enables them to track fast-moving objects without motion blur, making them ideal for high-speed applications such as robotics, motion analysis, and motion tracking. More continuous footage can be captured.

- *Low Power Consumption*: Event cameras process only pixels that have changed, significantly reducing data processing and power consumption compared to frame-based cameras that output signals from all pixels regardless of whether changes have occurred. This efficiency is very useful for battery-powered or power-sensitive applications.

- *High Dynamic Range (HDR)*: Due to the special design of the hardware, event cameras usually have a higher dynamic range (usually over 120 dB), which allows them to handle scenes with extremely high contrast, such as seeing shadows or bright lights, without losing details. Frame-based cameras usually struggle with such scenes due to saturation or underexposure.

Disadvantages

Event cameras are not perfect. Compared with frame-based cameras, they have the following disadvantages:

- *Low Spacial Resolution*: Event cameras detect only changes in brightness and cannot capture static elements in a scene. The outputs have less spatial detail, such as texture. In addition, the sensor resolution of the event camera can only reach up to 640 x 480. This makes them less suitable for applications that require complete scene information.
- *Less sensitive to low-speed moving objects*: Because the motions of objects can cause a change in brightness. But when objects move at a slow speed, the corresponding brightness changes can be few. This means only a few events can be triggered. If the object is static, there is even no output at all.
- *Inconsistency of outputs*: Due to the event camera's special working principle, even when shooting the same moving object, the output will look different. For example, when the object moves at different speeds or only translates or rotates, the output looks very different as shown in Figure 2.4.

2.1.3 Research of Event Cameras

Gallego et al. have conducted a detailed investigation [10] [37] into the development and application of event cameras. In this chapter, we will list some of the main application scenarios and research of event cameras.

The original purpose of event cameras was to detect high-speed moving objects. The algorithm for detecting moving vehicles using event cameras was proposed by Litzenberger et al. [19] when event cameras were first invented. This algorithm is mainly used for situations where the camera is in a static state. Subsequently, Anton et al. proposed a moving object recognition method [23] when the camera is in a dynamic state. This method estimates the camera's ego motion by analyzing the movement trend of a specific pattern in the scene and compensates for the event frame output. The compensated event frames are clearer. By comparing event frames at different times, moving objects can be distinguished from stationary objects. Inspired by this, Zhao proposed a method [40] using an event camera with IMU to measure the ego-motion directly, and finally the camera output is optimized to reduce motion blur and obtain clearer output results.

Recently, deep learning has received great attention in this emerging field, and a large number of technologies have been developed for various purposes. Zheng et al. conducted an investigation [42] on how deep learning is combined with event cameras. For example, Sanket et al. proposed a method [33] that combines drones with event cameras and uses neural networks to identify objects flying toward the drone and avoid them. It also proposes a method to denoise and deblur event frames using neural networks. Another method proposed by Durvasula et al. [7] uses an enhanced *U-Net* network to improve the speed and accuracy of robot image recognition. In the method proposed by Li et al. [17], traditional algorithms are combined with deep learning networks for autonomous driving target detection based on event cameras.

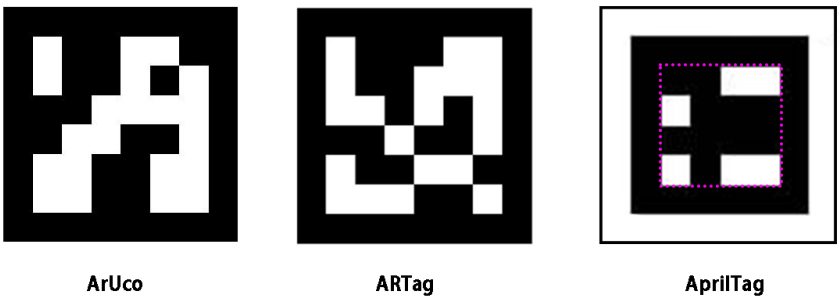


Figure 2.5: **Examples of different kinds of fiducial markers.**

In addition, event cameras are often used in conjunction with traditional frame-based cameras, combining the high temporal resolution of event cameras output with the high spatial resolution of frame-based cameras output. Pan et al. proposed a method to reconstruct a high-frame-rate video from a single blurry RGB frame and event camera output. Messikommer [22] proposed a method combining event and grayscale frames to realize enhanced feature tracking.

2.2 Fiducial Marker

2.2.1 General Introduction

Fiducial markers are reference objects or patterns placed in the physical environment or in an image to provide known reference points. These markers are used in various fields such as computer vision, robotics, augmented reality (AR), and photogrammetry to help systems determine orientation, scale, and position within a space. Fiducial markers are essential for precise measurement, alignment, and tracking tasks.

Common fiducial markers include ARTag [8], ArUco [29], and AprilTag [26]. They have different coding systems and their own applicable scenarios. AprilTag16 is selected in this thesis, which has 16 encodable bits. As shown in the Figure 2.5, AprilTag16 consists of an outer white frame and a black encodable area, and the pink dotted line is the precise encodable area range. A black or white square represents each bit, black represents 0, and white represents 1. These squares are arranged in different shapes to represent different IDs. These shapes are fixed and tailored to ensure that the Hamming distance between different shapes is maximized to reduce the misjudgment rate during detection. And it can be least affected by the orientation of the marker during decoding. In a system using a fiducial marker, the sub-system performing the marker identification will store a lookup table, which contains the correspondence between the shape and the ID. As long as the decoded content has the same shape as the shape in the lookup table after it's arranged in a certain order, it is considered to be decoded successfully.

2.2.2 Features

The fiducial markers have the following features:

- *High Contrast*: Fiducial markers typically have high contrast patterns, making them easily detected and distinguishable by cameras or sensors in different light situations.
- *Unique Patterns*: Each marker typically has a unique pattern or code, which enables the system to recognize and distinguish multiple markers in a scene.
- *Known Geometry*: The shapes of the markers are predetermined and known to the system, allowing precise calculation of spatial properties and easily detected when partially obscured.

2.2.3 Research of Fiducial Markers with Event Cameras

Since the image style of the event camera output is completely different from that of the frame-based output, the fiducial marker recognition method proposed in A is completely unusable. Therefore, Sarmado et al. proposed a fiducial marker recognition method based on event cameras [34]. However, this method is not related to the direction of movement of the marker, and has extremely low accuracy for diagonal movement. The recognition method [1] proposed by Loch et al. is based on the method [9] used for fiducial marker recognition with frame-based cameras proposed by Romero-Ramirez et al. The limitation of method [9] makes [1] only applicable when the event frame is very clear and has distinct edges, which has low versatility. To our best knowledge, these are the only two studies for fiducial marker detection using event cameras.

2.3 Image Segmentation Methods

2.3.1 General Introduction

Image segmentation is a computer vision technique used to divide an image into multiple regions. The segmentation process involves assigning a label to each pixel in the image so that pixels with the same label have certain common characteristics, such as color, intensity, or texture. Image segmentation is widely used in fields like medical imaging to segment cells and tumors and in autonomous vehicles to understand the environment by segmenting pedestrians, roads, and other vehicles. Image segmentation can be divided into semantic segmentation, which groups pixels with similar features in an image into the same category, and instance segmentation, which not only groups pixels into categories but also distinguishes different instances of the same type of object. Or the combination of the two is called panoptic segmentation.

When segmenting an image, it is necessary to understand a concept for evaluating whether a pixel is segmented correctly. This concept will be further used in Setion 4.2.1 for model evaluations. Figure 2.6 shows the definition of different types of pixel segmentation results:

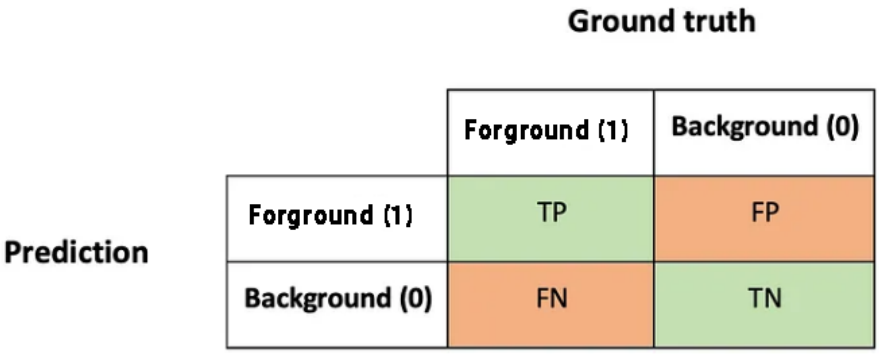


Figure 2.6: **Definition of different types of pixel segmentation results.**

- *TP (True Positive)*: TP represents the number of foreground pixels that have been properly classified as foreground.
- *FP (False Positive)*: FP represents the number of background pixels being misclassified as foreground.
- *FN (False Negative)*: FN represents the number of foreground pixels being misclassified as background.
- *TN (True Negative*): TN represents the number of background pixels that have been properly classified as background.

2.3.2 Research of Image Segmentation

Traditional Algorithm

The simplest traditional algorithm uses thresholds to segment images. It sets all pixel values above a certain threshold to one value (such as white) and all other pixel values to another value (such as black). Otsu [28] proposed a method to determine the threshold dynamically. This method classifies objects into two categories: foreground and background. The threshold is calculated to minimize the variance within the two categories and maximize the variance between the two categories. And also, there is edge-based image segmentation. Canny [5] and Sobel [15] are two commonly used image segmentation methods. They are basically based on image gradient calculation, but Canny is not as sensitive to noise as Sobel. In addition, there are image segmentation methods based on clustering algorithms. Clustering methods such as K-Means [21] group pixels into clusters based on their features (such as intensity, color, or texture). Each cluster corresponds to a segment in the image.

Deep Learning for Image Segmentation

Compared with traditional algorithms, using deep learning for image segmentation can learn complex features and patterns in images, and achieve more accurate and robust segmentation even in challenging scenarios. For example, different object sizes, shapes and directions, as well as blurred or overlapping

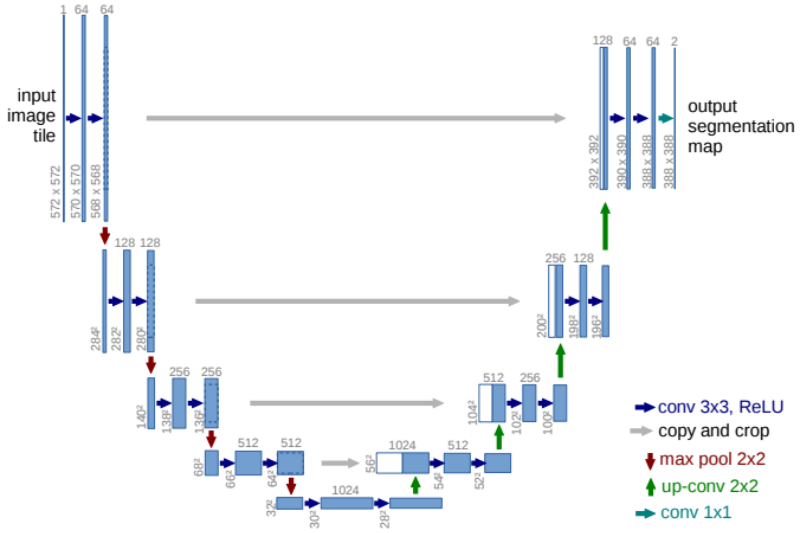


Figure 2.7: **U-net architecture. The horizontal gray arrows represent skip connections.**

objects. Ning et al. first proposed the method [24] using a convolutional network for biological image segmentation. Later, *FCN* [20] proposed by Long et al. laid the foundation for image segmentation using deep learning. *FCN* replaces the fully connected layer in traditional *CNN* with convolutional layers, which allows it to accept inputs of different sizes. However, since the last layer only performs a simple upsampling, some image details will be lost.

Later, Long et al. proposed *U-Net* [31] for medical cell segmentation. This time, U-Net added more deconvolution layers to the decoder part based on *FCN* to obtain better image details, and added skip connections to keep spatial information and ensure image details. However, this skip connection causes *U-Net* to have extremely high memory usage and the model prediction time was long. Figure 2.7 shows the model structure of *U-Net*. Our method uses a network with a U-Net-like structure and will be introduced in Section 3.3.2.

Furthermore, Badrinarayanan et al. proposed *SegNet* [3]. Although *SegNet* has an encoder-decoder structure similar to *U-Net*, *SegNet* does not directly pass the feature map in the encoder to the decoder and combine it with other decoder inputs. Instead, it stores the key pixel locations at each downsampling to obtain pooling indices, and restores these indices to the corresponding positions during upsampling to retain some image details. This greatly reduces memory usage, but the accuracy is not as good as *U-Net*.

Recently, Kirillov et al. proposed a complex network architecture *SAM* [16]. It can be generalized to segment any object in any image, without the need to retrain or fine-tune for a specific task, so it is incredibly versatile. However, due to its complexity and the use of advanced techniques, *SAM* requires much more computing power and memory, so it is not suitable for deployment on embedded platforms.

So after that, Zhao et al. proposed *FastSAM* [41], and Zhang et al. proposed *MobileSAM* [39]. Both are designed specifically for embedded platforms. *FastSAM* is 98% faster than *SAM*, and *MobileSAM* has achieved a 75% speed increase on this basis and achieved an accuracy that is not far from *SAM*. This makes *MobileSAM* one of the most suitable image segmentation models for embedded platforms. However, since we only discovered this network when writing this report, we did not have the opportunity to combine it with our proposed method. We mentioned the impact of using *MobileSAM* on our method in Section 5.2.1.

Chapter 3

Design

3.1 Overview

This chapter introduces the design of the proposed method to achieve the research objectives raised in Section 1.2. The final solution is inspired by the method of Sanket et al. [25], which aims to solve the positioning problem of quadcopter drones by using neural networks to detect the common feature of all quadcopters: propellers. Here, **"common feature"** is a pivotal concept in the approach. Similarly, for scenarios like detecting moving vehicles, pinpointing key features of the vehicles could allow for their recognition. However, the challenge arises from the diversity of vehicles. Especially from the drone's perspective, discernible common features are not apparent. Furthermore, training neural networks merely to identify moving vehicles raises issues with flexibility and expandability. Given these challenges, developing a universal feature applicable across different objects and scenarios to enhance object detection and identification is considered to be a proper solution. Consequently, in order to guarantee the versatility and scalability of the final solution, the fiducial marker is chosen.

The fiducial marker plays an important role in the proposed method. Unlike traditional object detection methods that check the object directly, this method employs fiducial markers to represent the objects for detection. Ideally, markers will be attached to the corresponding objects. Furthermore, they are not ordinary fiducial makers but optimized and encoded to carry specific information, e.g. the object they are attached to. Detection of the markers is considered detection of the objects themselves, simplifying the recognition process.

The pipeline of the method can be divided into two parts, as shown in Figure 3.1. The first part only has one step, which is the detection of the moving fiducial marker from the event frame.

The second part, the identification, can be divided into two steps. Firstly, with the output from the first part, the location of the fiducial mark in the event image is determined. This means the corresponding content can be extracted from the event frame and saved as marker candidates. As the method is designed to recognize fiducial markers moving in the 3D space, the candidates always need to be unwarped to get the best visual posture for the further processing step. As mentioned before, the optimized fiducial marker can contain some messages

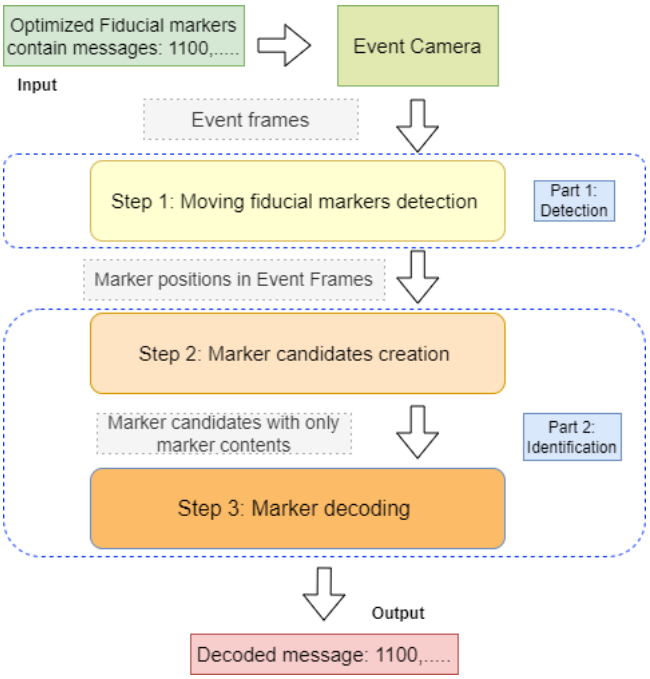


Figure 3.1: **The pipeline for moving object detection based on event camera.**

to be transmitted. So the last step is to decode the marker candidates according to the output from the previous steps. Section 3.3 to Section 3.5 will introduce the design of each step respectively. But before that, Section 3.2 will introduce the concept of optimizing the fiducial marker.

3.2 Optimized Fiducial Marker

3.2.1 Reason of Choice

Fiducial markers are initially used in computer vision and robotics to provide a readily detectable reference point or feature in an image or video stream. Similarly, to achieve easy and reliable object detection, fiducial markers are employed in this methodology. The technique of detecting fiducial markers with an frame-based camera can not be used with an event camera, which makes fiducial markers lose some advantages in this context, e.g., quick detection through image gradients and enclosed polygons. However, the essential characteristic of high contrast between encodable bits and the ambient environment remains beneficial. A moving high-contrast marker leads to obvious brightness variations and can be effectively captured by an event camera.

The fiducial marker type chosen in this methodology is AprilTag16 having 16 bits encodable bits. AprilTag also provides other types with more encodable bits like 25 and 36 bits. However, the choice is limited by the low resolution of the event camera we are using (DAVIS346 with a resolution of 346 * 260 pixels). If it is assumed different type markers take up the same size of the area

Figure 3.2: **Optimized fiducial with its direction indicator (black blocks on the white frame), and an output view of an event camera.**

in the event camera view, in order to increase the rate of successful decoding, representing a bit with more pixels is a better choice.

3.2.2 Optimization: Direction Indicators

As mentioned in Section 2.2.1, the fiducial marker is designed to have unique IDs with corresponding shapes consisting of different combinations of encodable bits. For AprilTag16, it has 30 various IDs. To fully use every encodable bit of the marker to transmit more information rather than merely 30 different messages, the direction indicators are added to markers.

As shown in Figure 3.2, the direction indicators are three black blocks on the white frame of the marker. In the traditional application of fiducial markers, the orientation of the marker is irrelevant because the shape of the encodable area corresponding to its ID is fixed. As shown with two markers on the left of Figure 3.3. However, when each encodable bit of the marker is utilized for transmitting information, it becomes crucial to indicate the starting bit of the information sequence. In our design, the indicator on the left-top corner indicates the start position should be the bit on the first column of the first row. The other bits will be encoded line by line from left to right. As shown with two markers on the right of Figure 3.3, even if both markers have the same encodable area shape with different start points, the decoded results are different.

3.3 Pipeline Step 1: Detection

3.3.1 Tradition Fiducial Marker Detection Method

For the application of fiducial markers with frame-based cameras, the detection method mentioned in [26] can be summarized into two main steps. 1) Finding contour and segmenting lines using image gradient directions and magnitude. 2) Iterating all line segments and contours to find the combination of closed polygons.

The efficacy of this method relies on image continuity. Typically, the output from an frame-based camera maintains its continuity. In contrast, the output from an event camera does not guarantee this, as illustrated in Figure 3.5. The fundamental working principle of an event camera is to detect shifts in light intensity. Therefore, when the marker moves slowly, rotates, or shifts

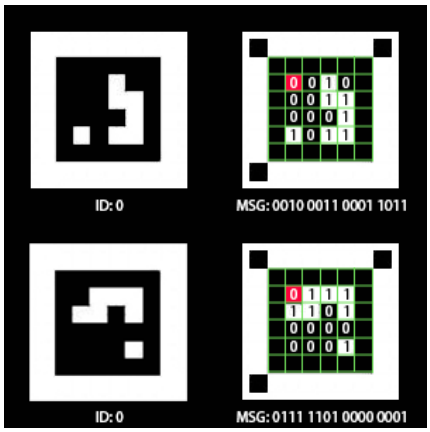


Figure 3.3: **Different decoding results with same encodable area shape, red blocks indicate start bits.**

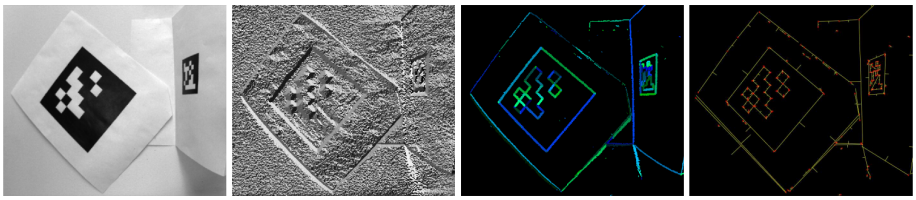


Figure 3.4: **Traditional detection method: Raw image (first), gradient direction (second), gradient magnitude(third), finding enclose polygons(fourth).**

unidirectionally, the change in light intensity might be insufficient to trigger an event on some pixels. As a result, the traditional method is not suitable for marker detection with event camera output.

As mentioned in Section 2.2.3, Sarmado et al. proposed a method [34] used for marker detection with event camera output. It has a similar idea to the traditional method: 1) Performing line segmentation. 2) Finding approximate polygons by iterating line segments and comparing the project between two different segments. If the angles and distances between two lines are below the specified thresholds and the endpoint of one line projects onto the other, these lines are considered adjacent sides of a polygon. While this approach effectively detects and decodes the marker, it is not universally applicable due to similar limitations encountered with previous methods (too many or too few lines can be detected because of discontinuity). Moreover, when the marker moves diagonally as illustrated in Figure 3.6, the detected line segments fail to project onto each other, preventing the formation of a valid approximate polygon. Consequently, there is a need for a more versatile method that does not depend on traditional algorithms.

Figure 3.5: **An output with continuous contours(first), discontinuity caused by unidirectional shift, rotation, and slow movement (second, third and forth). The edges of the markers are not connected anymore.**

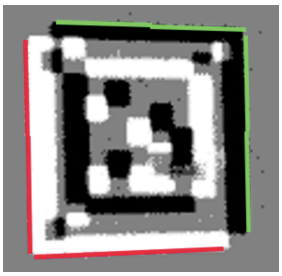


Figure 3.6: **A diagonally moving marker. Detected green lines can not project onto the closest red lines. And they are too far away from each other.**

3.3.2 Purposed Detention Method: *EvMarkNet*

To detect fiducial markers with different moving statuses, we propose a method using a deep learning neural network called *EvMarkNet*. This section discusses the dataset used to train *EvMarkNet* and the detailed design of the *EvMarkNet*.

Dataset for *EvMarkNet*

The dataset for training *EvMarkNet* contains 10k data samples captured by our event camera. Figure 3.7 shows a general view of the dataset. Image data samples with gray backgrounds are event frames containing markers, which is also the default visual output of our event cameras. The ground truth labels are binary masks, where the white polygon indicates the position of markers in image data samples, and black holes within the white polygon indicate the position of direction indicators.

For image data sample generation, we create a marker set with 30 variations of the optimized fiducial marker and create a pipeline to create videos called MKV containing moving markers. In every MKV, one to three random markers are chosen. Every marker is assigned a fixed number of random waypoints. The waypoints are the pixel coordinates, showing the corresponding markers' moving destination in MKV. Given that every MKV has a fixed duration, the time spent moving between waypoints remains consistent. This leads to varying speeds of marker movement across different videos. In addition, different markers in the same video may overlap due to random movement paths. This provides

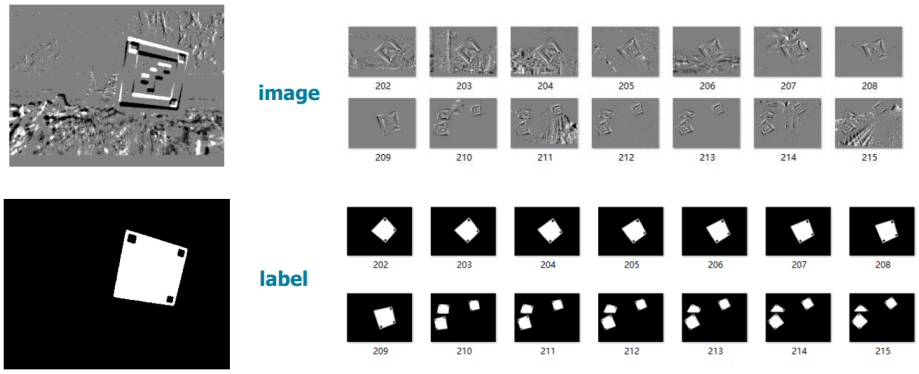


Figure 3.7: **Data samples in the dataset with their ground truth.**

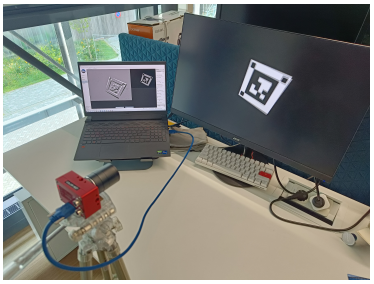


Figure 3.8: **Filming MKV with an event camera, event frame output can be seen on the left screen.**

additional training data for scenarios where markers are partially occluded. Besides that, random 3D rotation angles are also set to markers while they are shifting. It is worth mentioning that, as mentioned in Section 2.1.2, the inconsistency in the event camera's event frame output is caused by the different speeds of the objects moving relative to the camera. Therefore, the randomness of the marker movement speed when generating the MKV enriches the dataset's sample variety. As soon as the MKVs are generated, another type of video called LBV is also generated. LBVs are videos consisting of moving binary masks that share the same moving path and rotation angles as markers in MKV. In other words, LBVs are actually the position reference of MKV.

With both MKVs and LBVs, we film the screen playing these videos with our event camera, as depicted in Figure 3.8. For the MKV, we collect the default visual output from the event camera, which is the accumulated event frames. For LBV, since our event camera has a grayscale frame output mode specially designed for testing, we directly collect the grayscale frame output and subsequently adjust the contrast to produce grayscale binary masks. At last, both outputs are sampled to extract image data samples along with their corresponding ground truth labels. With a method similar to film MKV, we capture several videos of urban landscapes and roads which are filmed by drones equipped with frame-based cameras. This output is sampled and utilized as random noise in the background of the image data samples. The event camera

Table 3.1: **Event camera parameters for capturing data.**

Decay Function	Decay Param	Accumulation	Event Contribution	Frame Interval(for grayscale frame output)
Step	1.0E10	27ms	0.228	900
Max Potential	Min Potential	Neutral Potential	Exposure(for grayscale frame output)	Frame Mode (for grayscale frame output)
0.0	1.0	0.5	8500	Grayscale

used for data capturing is DAVIS346, having a resolution of 346 * 260, and all data is collected by its supporting software DV [13]. Table 3.1 shows the setup of the event camera for capturing data in DV.

Design for *EvMarkNet*

Since it is necessary to accurately know the position and edge information of the fiducial marker, detecting fiducial markers from the event frame output is intrinsically an image segmentation task. While advanced models like *SegNet* and *SAM* excel in image segmentation, their large size, substantial memory requirements, and slow processing speeds make them impractical for embedded platforms. Additionally, no existing applications integrate event camera output with these models. To address this gap, we developed a network specifically tailored for this purpose, named *EvMarkNet*, which is designed to effectively segment fiducial markers from the event frame.

EvMarkNet has a U-Net-like encoder-decoder structure, as shown in Figure 3.9. The first half of the network is called the encoder, which captures the critical characteristic of the input but in a reduced dimensionality. In our design, the encoder always consists of convolution layers followed by a batch normalization layer and a ReLu activation layer. As the number of layers increases, the features extracted by each convolutional layer will gradually become abstract. For example, the initial layers will extract some simple feature information, such as corners and edges. For deeper layers, they will analyze the combination of features extracted by the previous layers and finally discern some complex non-visual features, such as interactions between background and objects.

The rest of the network is called the decoder. A decoder takes the encoded data and reconstructs it back to the original input data or translates it into another form of useful output. In our case, the output should be binary masks showing the position of markers in the input event frames, similar to the ground truth labels in the dataset. In *EvMarkNet*, the decoder always consists of deconvolution layers to recover the image details.

The significant difference from the *U-Net* is the reduction of skip connection and max-pool downsampling, and the application of *ResBlocks*. As mentioned in Section 2.3.2, skip connection is designed to preserve the spatial details of the image at different depths in the network, preventing the loss of details during downsampling from causing the final output to be less accurate. However, the skip connection can take up a substantial amount of memory. So we decide to deprecate it. For downsampling steps, instead of using max-pooling, a convolution layer with a stride of two is applied to reduce information loss. The ResBlock is an important technology introduced in ResNet [14]. And the advanced version introduced in [12] is applied in the network. In a nutshell, *ResBlocks* help in training deeper networks to converge faster and show better performance on complex tasks. And they allow networks to learn robust and hierarchical features across different layers, improving the network's ability to

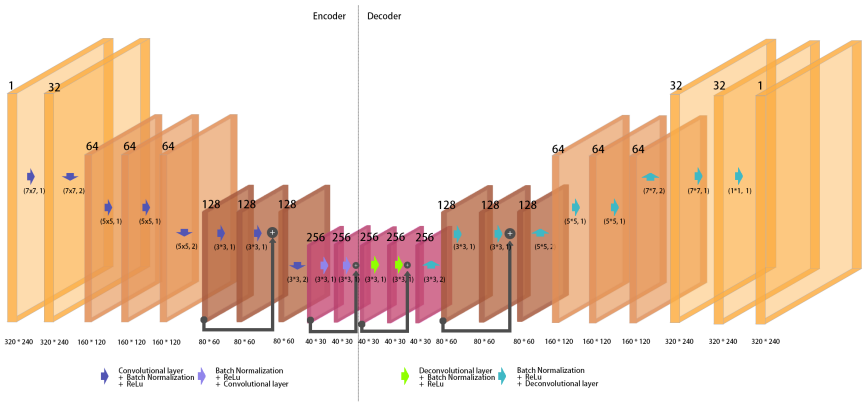


Figure 3.9: **The structure of *EvMarkNet***

Table 3.2: **Statistics of the symmetry and number of *ResBlocks* of different versions of *EvMarkNet*, where "Max Dim" is the maximum feature extraction and restoration dimension reached by the encoder and decoder. 'Layers' is the total number of (de)convolution layers. Each ResBlock has two (de)convolutional layers.**

EvMarkNet	Encoder				Decoder			
	Input Dim	Max Dim	Layers	*ResBlocks*	Output Dim	Max Dim	Layers	*ResBlocks*
V0	32	128	11	2	16	256	11	2
V1		256	11	3	16	256	11	3
V2		256	8	1	16	256	8	1
V3		256	10	2	32	256	10	2

generalize from training data to unseen data.

During the network design process, it is necessary to focus on the network's symmetry and the number of *ResBlocks*. The introduction of *ResBlocks* and the use of convolutional layers over pooling for downsampling result in asymmetry. The asymmetry is reflected in the difference between the maximum feature extraction dimension reached by the encoder and the maximum feature rebuilding dimension reached by the decoder. For instance, in our initial design, the output dimension of the encoder is 128. Afterward, a downsampling convolution layer increases the dimension to 256. This will be the input to the subsequent deconvolution layer belonging to the decoder. The output dimension from the encoder does not align with the input dimension of the decoder, which can be seen as asymmetry. The asymmetry impedes the encoder's ability to extract high-level features for the decoder to utilize in image restoration. In addition, excessive use of *ResBlocks* can cause a large-size model, while insufficient use can result in extremely slow training convergence. To figure out a balanced network structure, we design four versions of networks and compare their training and prediction performance. Detailed results will be discussed in Section 4.2.1. Table 3.2 shows the structure difference between the four versions (the statistics in this table don't count the last layer used for outputting a single-dimension grayscale image).

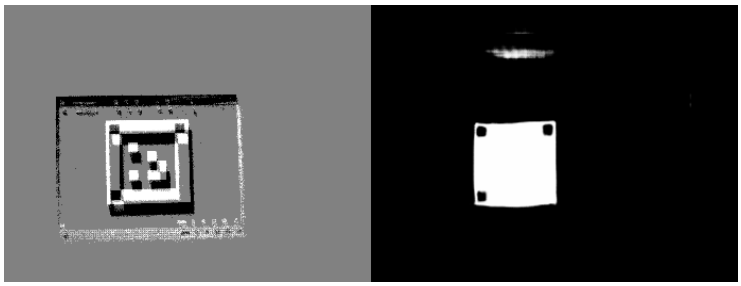


Figure 3.10: **An input and the prediction result of *EvMarkNet* V3.**

EvMarkNet V0, as it's just explained, the maximum reached dimension by encoder and decoder is not matched. Meanwhile, the network output dimension is less than the network input dimension. Therefore, the change from V0 to V1 mainly focuses on improving the symmetry of the network. As a result, V1 guarantees that the maximum dimension reached by the encoder and decoder is the same. Regarding V2, it is only used to validate the effect of the number of *ResBlocks*. For v3, based on V1, it ensures symmetry and an appropriate number of *ResBlocks* to achieve the most balanced prediction time, model size, and the best output results.

It is important to note that it's not evident from the table, but the transition from V0 to V3 includes an increase in the number of convolutional layers dedicated to low-level feature extraction and restoration, specifically the layers before and after the ResBlock. V0 contains only six layers for this purpose, whereas V3 expands to ten layers. This increase is believed to play a critical role in enhancing the spatial details of the final outputs.

Based on *EvMarkNet* V3, as depicted in Figure 3.9, the network is trained with the dataset mentioned in Section 3.3.2. Binary Cross Entropy is used as training loss that can be represented with the following formula:

$$\text{BCE} = -\left[y \log(\hat{y}) + (1 - y) \log(1 - \hat{y})\right] \tag{3.1}$$

where:

- y is the actual label (0 or 1), provided by the ground truth labels in our dataset.

- $\hat{y}$ is the predicted probability (between 0 and 1), provided by the prediction result of *EvMarkNet*.

Finally, *EvMarkNet* V3 is trained for 400 epochs with a batch size of 32. The training is warmed up with a constant learning rate of $7e^{-4}$ to 300 epochs. And a cosine decay strategy is used for faster convergence. The learning rate will decay to $2e^{-3}$ in the last 100 epochs. The Adam optimizer is applied. Figure 3.10 shows one of the prediction outputs from by *EvMarkNet* V3.

3.4 Pipeline Step 2: Identification: Create Marker candidates

After obtaining the marker position predictions from *EvMarkNet*, it is essential to extract the specific contents of the fiducial markers from the corresponding areas in the event frame as indicated by the white polygons in the binary masks. These are then used to establish marker candidates for decoding in the next step. However, as illustrated in Figure 3.10, the prediction from *EvMarkNet* is not flawless. The polygon edges are irregular, not accurately depicting the proper position of the markers. Additionally, there is a minor white shadow above the white polygon, indicating a predictive error in the marker's position. To achieve a more precise determination of marker positions, further processing of the binary mask output from *EvMarkNet* is necessary.

3.4.1 Optimizing polygon edges

Finding Contours

The initial step in image processing involves determining the edge coordinates of the polygon. To achieve this, we first apply a threshold of 80 to the binary mask to enhance the boundary visibility between black and white pixels. Subsequently, we employ the Suzuki-Abe Algorithm [35] and the Moore-Neighbor Tracing Algorithm [4] to segment the black and white pixel areas and obtain the contour coordinates of the white pixel areas. Since OpenCV [27] offers an optimized function, `cv2.findContours()`, based on these algorithms. We use this function directly to ensure the efficiency of our method. Typically, in the model's predictions, the white polygon indicating the marker's location has the largest contour perimeter. Thus, the largest contour found by `cv2.findContours()` is finally selected. Next, we utilize the `cv2.approxPolyDP()` function, which is based on the Douglas-Peucker algorithm [32], to approximate the shape of the white polygon to the nearest quadrilateral. The outcome of this process is the coordinates of the four vertices of the quadrilateral.

Determining Vertices

According to the vertices obtained from the last stage, we draw the quadrilateral and vertices. As shown in the first image in Figure 3.11, it is a perfect estimated quadrilateral with four vertices based on a regular output from *EvMarkNet* without many prediction errors. However, the second and third images contain significant errors affecting the number of vertices. Therefore, the estimations of the edge of the quadrilateral by `cv2.approxPolyDP()` have substantial bias. The red lines are completely not attached to the edges of white polygons.

To solve this kind of issue, we design algorithms to correct the position of vertices. The problem can be classified into two categories. The first category is the white polygons having more than four vertices. In this case, there must be several vertices clustering with each other. We have the following definition:

$$A_i = \frac{\sum_{j=1}^{n} dis(V_i, V_j)}{n} \tag{3.2}$$

where A_i is the average distance between vertex V_i and other vertices V_j. And n represents the number of all vertices. In the second image of Figure 3.11, given

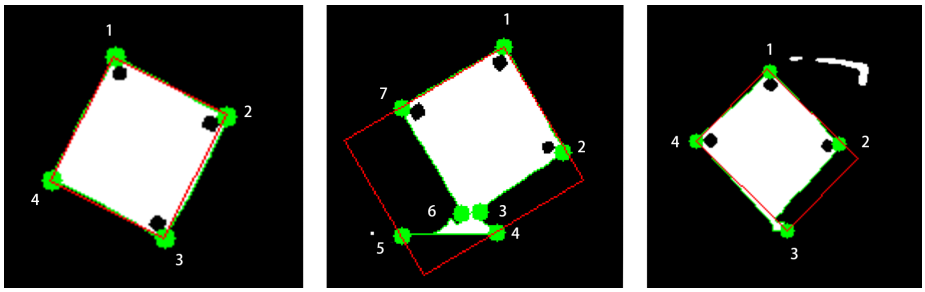


Figure 3.11: **The outlines (green lines), vertices (green dots), and approximate quadrilaterals (red lines) of different white polygons in different outputs**

that vertices 3, 4, 5, and 6 are close to each other, their average distance to all other vertices is smaller compared to the average distance between vertices 1, 2, and 7 and the rest. Consequently, vertices 3, 4, 5, and 6 can be identified as outliers and removed from the list of vertices.

The second category is the white polygons having four vertices, but one of the vertices is far away from the other three vertices, as depicted in the third image of Figure 3.11. The vertex 3 is located on the prediction error part of the white polygon and causes mistaken quadrilaterals. To solve this problem, we calculate angles between lines of every three points arranged clockwise. Given the points $V_a(a_x, a_y)$, $V_b(b_x, b_y)$, and $V_c(c_x, c_y)$, we can define the vectors: $\overrightarrow{BA}$ and $\overrightarrow{BC}$. The dot product of the vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$ is calculated as:

$$\text{DP} = (a_x - b_x) \cdot (c_x - b_x) + (a_y - b_y) \cdot (c_y - b_y) \tag{3.3}$$

The magnitudes of the vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$ are given by:

$$\|\overrightarrow{BA}\| = \sqrt{(a_x - b_x)^2 + (a_y - b_y)^2} \tag{3.4}$$

$$\|\overrightarrow{BC}\| = \sqrt{(c_x - b_x)^2 + (c_y - b_y)^2} \tag{3.5}$$

The cosine of the angle θ between the vectors $\overrightarrow{BA}$ and $\overrightarrow{BC}$ is calculated as:

$$\cos(\theta) = \frac{\text{DP}}{\|\overrightarrow{BA}\| \cdot \|\overrightarrow{BC}\|} \tag{3.6}$$

The angle can be obtained by performing the arc cosine operation on Equation 3.6. If the angle is less than 80 degrees, the corresponding vertex is considered an outlier and is removed from the vertices list. After filtering the vertices, three vertices are determined to be in reasonable positions for both types of problems. Assuming the desired quadrilateral should be symmetrical, the position of the fourth vertex is calculated based on the symmetry principle relative to the positions of the other three vertices. Eventually, the coordinates of these four vertices are considered as the position of vertices of fiducial markers in event frames.

3.4.2 Searching Direction Indicators

Besides locating the marker itself, it is also essential to identify the position of the direction indicator on the marker to determine its orientation. While the largest contour from `cv2.findContours()` is processed further as previously mentioned in Section 3.4.1, this section also utilizes other contour outputs. We can find out whether the positions of particular contours fall within the desired largest white polygon by comparing the center point of each contour to the main polygon.

If exactly three contours are detected within the range white polygon, their center coordinates are directly used as the position of the direction indicator. On the contrary, if the contours are not equal to three, the event frame input to *EvMarkNet* needs to be used. Since we have obtained the positions of the marker vertices in the event frame in the previous stage defined as V_i, the center point position C of the marker can be calculated from it. Therefore, there will be a vector from the center point to the vertex $\overrightarrow{CV}$ (green arrows in Figure 3.12). We can define a point M on this vector. The distance between M and C is eighty percent of the magnitude of $\overrightarrow{CV}$.

$$M(x, y) = (C_x + 0.8 * \|\overrightarrow{CV}\|_x,\ C_y + 0.8 * \|\overrightarrow{CV}\|_y) \tag{3.7}$$

The position of M is the approximate position of the center point of each direction indicator. We need to calculate the sum of the pixel color values within a 5*5 range around point M. At this time, gray pixels are considered to have a color value of 0, and black and white pixels all have color values of 128. As shown in Figure 3.12, except M_4, the remaining three points are surrounded by black or white pixels. So the sums of those pixels are significantly higher than the value sum of M_4. As a result, The corner where M_4 is located is considered as no direction indicator. After identifying the coordinates of the direction indicators, we arrange the three indicators for each marker in a clockwise sequence. The corner of the indicator in the middle of the sequence is designated as the upper left corner of the marker. The vertex closest to this indicator is also identified as the upper left vertex of the marker. The remaining vertices are then organized in a clockwise order based on their distance to the sorted indicators. With this arrangement, the orientation of the marker is determined.

3.4.3 Unwarping

In this step, we first extract the contents of the fiducial marks from the event frames using the contours and vertices outlined in Section 3.4.1 and Section 3.4.2. Since the markers move in three-dimensional space, the images captured by the event camera might have perspective deformations. To address this, we apply a homography transformation using `cv2.findHomography()` and `cv2.warpPerspective()`, aiming to align the extracted content more closely with a square shape. As the orientation of the marker was already determined in the previous section, all the marker contents after transformation should maintain a consistent orientation. Following the transformation, we scale the extracted content to generate outputs with various details. Figure 3.13 shows what the extracted content looks like before and after homography transform-

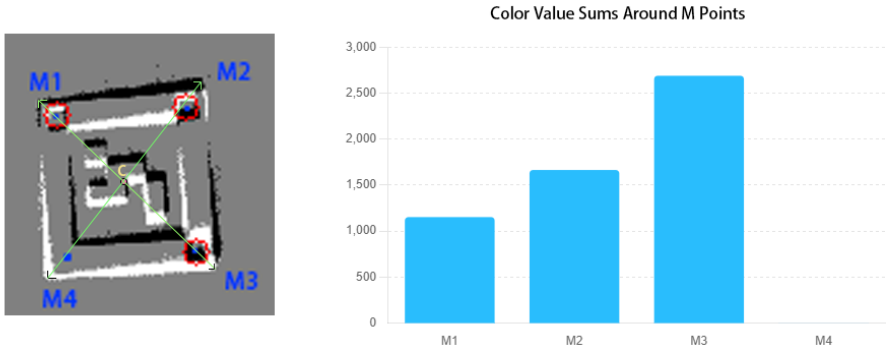


Figure 3.12: **Summing color values of pixels around blue points to find out direction indicators (red circles).**

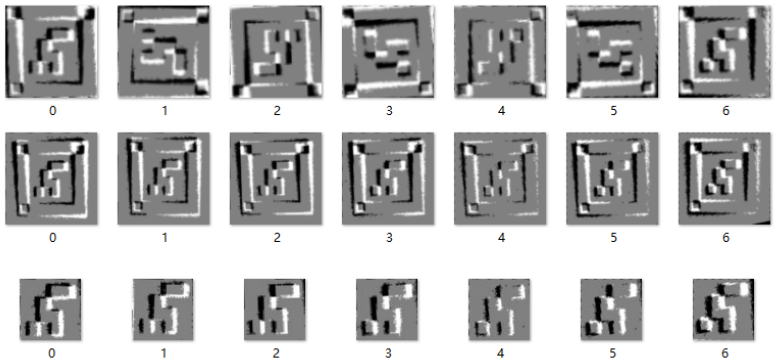


Figure 3.13: **Images on the three lines show the difference between contents extracted from the same event frames before homography transformation, after homography transformation and 1.2x rescaling, and after homography transformation and 0.7x rescaling respectively.**

ation, and after different scaling. The image contents that have undergone homography transformation and have been scaled down to 0.7x are finally chosen as the marker candidate used for decoding in the next step.

3.5 Pipeline Step 3: Identification: Decode

3.5.1 Exsiting Fiducial Marker Decoding Method Based on Event Camera

The method mentioned in Section 2.2.3 proposed by [34] divides events in a marker candidate into two candidates according to their types: an on-event candidate and an off-event candidate. They can be simply understood as composed of the white and black pixels in the original event frame. Both on-event and off-event candidates are then divided into a grid based on the number of encodable bits of the marker. The color of each cell in the grid is binary classi-

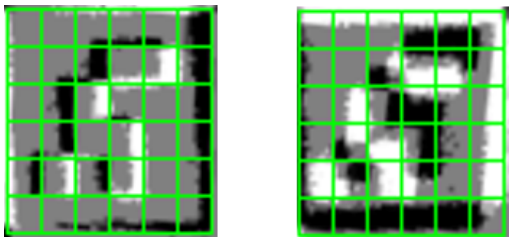


Figure 3.14: **The left image shows a perfectly gridded marker's encodable areas, with each grid containing only on or off events. The right image shows a grid with large errors due to the diagonal movement of the marker, with a very mixed pattern of events.**

fied into 0 or 1. Finally, these 1s and 0s will be sorted according to the direction of movement of the marker to obtain the final decoding result. Particularly in our application, where we decode dynamically changing information rather than a preset ID, misjudging the movement direction of the marker can lead to incorrect ordering of the binary sequence (0s and 1s). This error can result in entirely wrong decoded information.

Except this, another two primary issues with the method proposed in [34] prevent us from adopting it directly. First, the type of fiducial marker it decodes differs from ours. Its use of ArUco markers, which lack an outer white frame, simplifies identifying the boundaries of the encodable area. In contrast, our markers include wide white frames, adding complexity to edge detection. Second, as mentioned in **??** the method is tailored only for non-diagonal movements. It relies heavily on precise grid division. Each grid must align perfectly with each encodable bit for accurate results. Diagonal movements, however, often lead to event overflow, as illustrated in Figure 3.14. This can cause cells that should be classified as '0' to be misclassified as '1', leading to cascading decoding errors.

3.5.2 Decoding method based on enhanced region segmentation

In order to decode our markers and adapt to their movements in various directions, we propose a decoding method based on enhanced region segmentation. This approach utilizes the marker candidates obtained in Section 3.4. As mentioned before, these candidates are rescaled to 0.7x. This rescaling minimizes the presence of non-encodable areas within the candidate, facilitating more accurate decoding. Candidates are later divided into on-event candidates and off-event candidates. An example is shown in Figure 3.15. The marker in this example is called M_c containing a message: 1000,0010,0001,1011. Most later examples are based on M_c. Although the extracted content is already suitable for the subsequent decoding step, the inherent noise in the output from the event camera can introduce significant errors during decoding. To mitigate this, we apply Gaussian blur and a threshold of 200 to clean the candidates as depicted in Figure 3.16.

Figure 3.15: **A: marker M_c, B: the marker candidate extracted from M_c. C: M_c's on-event candidate. D: M_c's off-event candidate, the large white area on the top of this image is the white frame of our marker.**

Figure 3.16: **A: An on-event candidate before denoising. B: The same on-event candidate after denoising.**

First Refinement

Although the marker candidates have reduced the content of the non-encodable areas by scaling, small amounts of white frame contents still remain. Therefore, the initial refinement aims to determine the boundary between the white frame and the encodable area. Given that the following processing techniques are applicable to both on-event and off-event candidates.

To isolate the white frame from the encodable area in the candidate, four values need to be determined. We first calculate the sum of the color values for each column and row, resulting in arrays S_c and S_r respectively. Given that white pixels have a color value of 255 and black pixels are 0, a large sum in any column or row indicates a significant presence of white pixels. As illustrated in Figure 3.15d, marker borders appear as continuous white areas. We can establish a threshold and identify continuous segments in S_c and S_r that exceed this threshold to determine the horizontal and vertical coordinate ranges of the white areas.

We use four values to sign the boundaries of these borders, namely F_v, F_h, L_v, L_h. These values represent the approximate positions of the first transition from continuous black area to continuous white area in the vertical and horizontal directions, and the approximate position of the last transition from continuous white area to continuous black area in the vertical and horizontal directions.

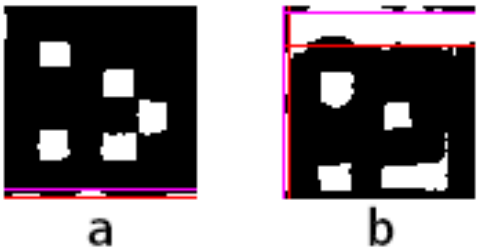


Figure 3.17: **First refinement of M_c's on-event and off-event candidates. The horizontal and vertical red lines correspond to L_h, L_v, and the horizontal and vertical pink lines correspond to F_h, F_v.**

Taking L_v and F_v as an example, it can be defined as follows:

$$L_v = \begin{cases} & 0 < i \leq N_c \wedge \\ i & S_c(i) \geq T_{\text{white}} \wedge \\ & S_c(i+j) \leq T_{\text{black}},\ \exists\, j \in [1,4] \\ \\ N_c - 1 & i \geq N_c - 4 \end{cases} \tag{3.8}$$

$$F_v = \begin{cases} & 0 \leq i < N_c \ \wedge \ i \leq Lv \wedge \\ i & S_c(i) \leq T_{\text{white}} \wedge \\ & S_c(i+j) \geq T_{\text{black}},\ \exists\, j \in [1,4] \\ \\ N_c - 4 & N_c - 4 \leq i \leq Lv \\ \\ 0 & i > Lv \end{cases} \tag{3.9}$$

Where N_c is the number of pixel columns of a candidate, $S_c(i)$ is the i_{th} sum value of array S_c, T_{white} and T_{black} indicate the threshold used to judge if column i is in a continuous white or black region. If column i is in the white area, and one of the next three adjacent columns is in the black area, then column i is considered to be the dividing line between the black and white areas, and vice versa. Figure 3.17 depicts the results of the first refinement of M_c. The white frame is perfectly detected. It can be observed from Figure 3.17a, only F_h and L_h are drawn. It's caused by the negligible size of the white frame area, and both F_v and L_v are 0 in this case.

Second Refinement

The initial refinement successfully eliminates most of the white frames from the candidate, yet the result is not entirely filled with the encodable area. Also, a black gap remains between the encodable area and the edge of the resulting image, as illustrated in Figure 3.18. Therefore, the focus of the second refinement is to determine the precise location of the encodable area.

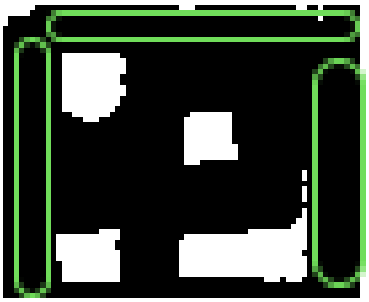


Figure 3.18: M_c**'s refined off-event candidate. The gaps between the codable and image edges are pointed out by green lines.**

Compared with the first refinement, the logic of the second refinement is much simpler. It is still necessary to obtain the sum of the candidate's row and column pixel color values S_r and S_c. At the same time, the candidate is scanned from four directions of the image to check whether the color value of each row and column exceeds the threshold T_b. Taking the determination of the right side edge position of the encodable area as an example, starting from the last pixel column of the candidate, if the value of column i is greater than the T_b, then check whether the values from $i-1$ to $i-I_c$ are all greater than the T_b (I_c is another threshold to indicate how many rows or columns to check in each iteration). If all values are greater than the T_b, then column i is considered to be the edge of the encodable area. It is worth mentioning that both T_b and I_c are adaptive rather than fixed. Since the number of events generated when the marker moves at different speeds and directions is different, the two thresholds will change according to the density of white pixels in the candidate. The higher the density, the higher the threshold. Figure 3.19 depicts some results of the second refinement.

$$\text{Column is an edge} = \begin{cases} & 0 < i \leq N_c \wedge \\ True & S_c(i) \geq T_b \wedge \\ & S_c(i-j) \geq T_b,\ \exists\, j \in [1, I_c] \\ \\ False & Otherwise \end{cases} \tag{3.10}$$

Decoding

For the final decoding step, we grid the results from the second refinement. We define the cell on the top left corner of the grid as (0,0) and the cell on the bottom right corner as (3,3). Unlike the method proposed in [34], the directions of our candidates are confirmed by the direction indicator at the time of creation, which ensures the order of the encodable bits is consistent and unaffected by the marker's movement direction. Most importantly, during the binary classification of cell contents, we employ an adaptive threshold. This threshold maximizes the tolerance to the bias of grid division. Even if white pixels overflow from one cell to another, the adaptive threshold ensures that these overflowed cells are accurately classified. This can be seen from Figure 3.20, which shows the

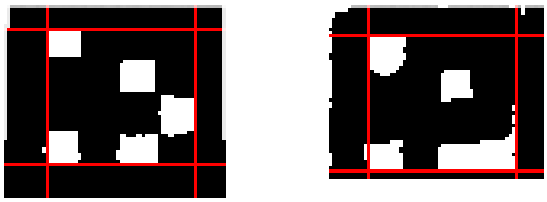


Figure 3.19: **Second refinement of M_c's on-event and off-event candidates. Red lines indicate the edge of the encodable areas.**

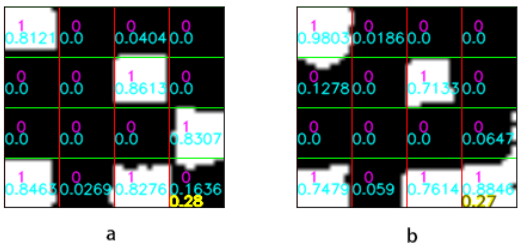


Figure 3.20: **Decoding results of M_c's on-event and off-event candidates. Pink texts are decoding the results of every bit. Blue texts are the proportions of white pixels in each cell. Yellow texts are the threshold of classification.**

final decoding results of M_c's on-event and off-event candidates. Regarding the threshold of Figure 3.20a, which is 28%. Though the cell (3,3) has 16% of white pixels in it, this portion is lower than 28%, so is still classified as a 0.

The results of these two candidates need to be read column by column starting from the bit (0,0). The message contained by Figure 3.20a is: 1000,0010,0001,1010. The message contained by Figure 3.20b is: 1000,0010,0000, 1111. Finally, the information decoded from the on-event and off-event frames is combined using an OR operation to get the message M_c containing: 1000,0010,0001,1011. This is the same as it mentioned in Section 3.5.2.

Chapter 4

Evaluation

This chapter will conduct a detailed quantitative and qualitative evaluation of the design proposed in Chapter 3 and analyze its results. In the end, we will summarize the performance of the entire methodology and analyze the existing problems and bottlenecks of this method.

4.1 Evaluation on Optimized Fiducial Marker

As the foundation for our marker system, AprilTag16 has only 16 encodable bits, which can be arranged into 30 distinct patterns to represent various IDs. However, when we utilize all bits for transmitting diverse information, the theoretical maximum number of different messages that can be communicated expands dramatically to 2^{16}. This significant increase in potential outputs arises because AprilTag16 is designed for robust detection, aiming to maximize the number of detectable or correctable bit errors. To achieve this, each designed pattern must consider its Hamming distance from other ID patterns, ensuring a substantial difference to effectively reduce error rates.

Fully utilizing each bit for information transmission compromises stability, particularly when the transmitted data is entirely random. This variation can significantly impact the efficiency of the decoding method we proposed. For instance, as depicted in Figure 4.1, consider a scenario where the encoded information in the marker consists of the sequences 1000,0010,0000,1010. Notably, every bit in the fourth column is '0' corresponding to black cells. The event camera's output reveals no on or off events in this last column. If our proposed method for region segmentation and gridding of the coding areas is applied, the decoded result turns out to be 1000,0001,0000,1001, which deviates substantially from the original encoded information.

One potential solution to improve the stability of the transmitted information is adding a header and footer, similar to the structure used in many communication protocols that include a header and a checksum. However, given the 16-bit limitation, designing a complex header is not feasible. To ensure the smooth execution of subsequent tests, we have to temporarily set both the first and last bits of the information sequence to '1'. Additionally, we avoid transmitting patterns that could entirely disappear at the edges of the encodable area, such as sequences where either the first four bits or the last four bits are all 1s.

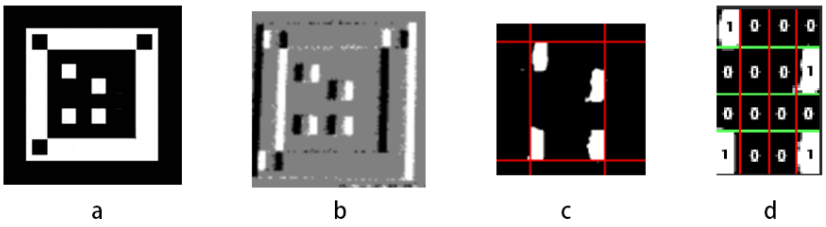


Figure 4.1: **A: a marker that contains all zeros in the last encodable column, B: the event frame output by the event camera, C: on-event candidate after the second refinement, D: on-event decoding result.**

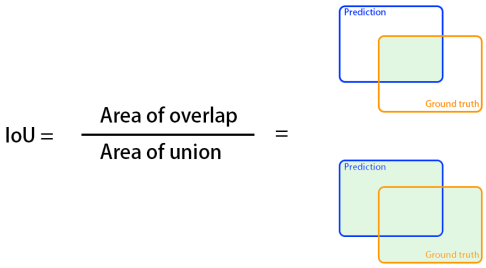


Figure 4.2: **The definition of IoU.**

Implementing this design constraint reduces the number of distinct information types that can be transmitted to 12,544. The variability is drastically reduced by 80%. Developing an efficient and stable coding system will be an essential task in the future.

4.2 Evaluation on *EvMarkNet*

4.2.1 Multi Version Comparison

As mentioned in Section 3.3.2, the *EvMarkNet* used for detecting is not the first version of the design. To evaluate the improvements of *EvMarkNet* V3 over its predecessors, we conducted a series of tests comparing different versions of the network. This comparison assessed the model size, parameter size, prediction time, and Intersection over Union (IoU) scores of the prediction results.

IoU is a metric for image segmentation tasks, quantifying the quality of the results by measuring the overlap between predicted and actual data. Based on the information introduced in Section 2.3.1, IoU can be defined as:

$$\text{IoU} = \frac{TP}{TP + FP + FN} \tag{4.1}$$

The accuracy can be defined as:

$$\text{Accuracy} = \frac{TP + TN}{TP + TN + FP + FN} \tag{4.2}$$

Table 4.1: **Evaluation between different versions of *EvMarkNet*.**

EvMarkNet	Model Size (MB)	Parameter (Million)	Avg. Speed (ms/it)	Avg. IoU (%)
V0	33.8	2.8	29	43.4
V1	59.3	4.8	38	63.8
V2	17.9	1.5	31	48.9
V3	55.1	4.6	43	81.6

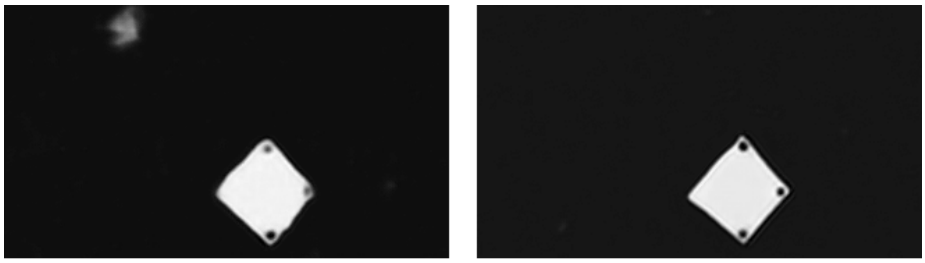


Figure 4.3: **A: an output from *EvMarkNet* v0, B: an output from *EvMarkNet* v1.**

The distinction between IoU and accuracy is shown in their treatment of True Negative (TN) data. IoU ignores TNs, meaning it does not consider the segmentation quality of the background content. As our image segmentation task focuses only on the foreground, i.g. the quality of fiducial marker segmentation, IoU is more suitable for evaluating the performance of *EvMarkNet* in this context.

We tested four versions of the model on a validation dataset of 1800 samples separated from the main dataset. And each model is trained by 200 epochs with a constant learning rate of $7e^{-4}$. The testing hardware setup is as follows: AMD EPYC 7413 24-core CPU, NVIDIA A40 GPU, and 16GB of video memory are allocated. The testing software setup is: Ubuntu 20.04, CUDA 12.3, cudnn 8.9.6, TensorFlow 2.16.1, python 3.11.0. Table 4.1 shows the results of our test.

From the data presented in the table, it is obvious that v2 has the smallest model size, while v1 has the largest. As detailed in Section 3.3.2, too many textitResBlocks contribute to larger model sizes. Hence, v2 with only two *ResBlocks* is smaller than v1 which includes six *ResBlocks*. This reduction in *ResBlocks* also results in faster prediction times for v2 compared to v1. Although v3 contains four *ResBlocks*, extra convolutional layers outside of the *ResBlocks* slightly increase its size, making it a similar size to v1. While v2 has advantages in terms of size and speed, its IoU for prediction results significantly falls behind v3. V2's prediction speed is 27.9% faster than v3's, but its prediction quality is 67.3% better. If we compare v0 with v1, the enhancement of the model's symmetry improves prediction quality by 32% and also results in more apparent segmentation of the marker edges, as depicted in Figure 4.3. At last, v3 demonstrates an 85.5% improvement in prediction quality over the initial version v0, confirming the effectiveness of our optimization and design strategies for the final model version.

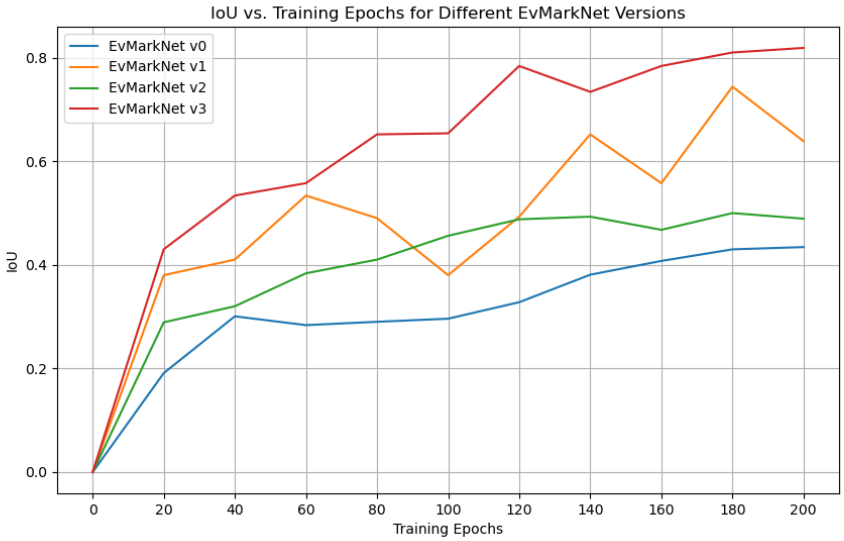


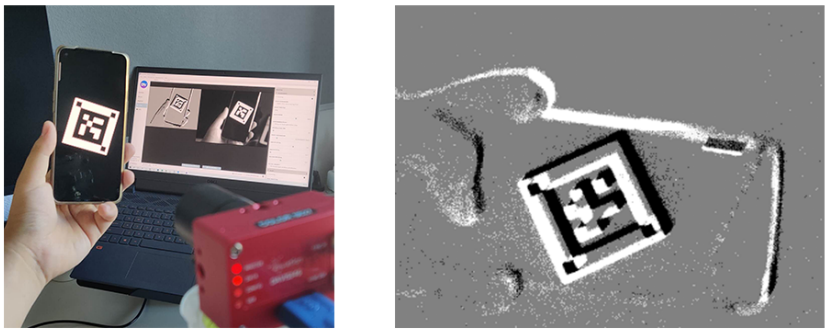
Figure 4.4: **IoU vs. Training Epochs for different *EvMarkNet* versions.**

Figure 4.4 illustrates how the Intersection over Union (IoU) of the model predictions related to training epochs. The training curves for models v0, v2, and v3, which contain fewer *ResBlocks*, are relatively smooth with minimal fluctuations. And they stabilize significantly after 160 epochs, indicating a trend toward convergence. In contrast, the training curve for v1 that uses more *ResBlocks* shows volatility. Notably, there is a sharp decline in IoU at 100 epochs. All those features show that a large number of *ResBlocks* can lead to training instability and difficulty in convergence.

4.2.2 *EvMarkNet* v3 Test Based on Independent Dataset

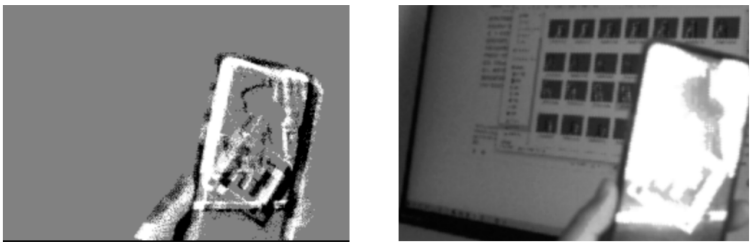
After evaluating the advantages of *EvMarkNet* v3 against earlier versions, we conducted a distinct test using an independent dataset to assess the model's capability with unseen data. We created 15 fiducial markers with random information. Instead of generating a video with randomly moving markers to display on a screen, we displayed the marker on a phone screen and manually swiped it at various speeds in front of the event camera. A total of 500 frames were captured for this test. Figure 4.5 illustrates both the data collection process and an example of a data sample. The final average IoU for the predictions on the new dataset is **65%**, which is lower than the results obtained with *EvMarkNet* v3 on the validation dataset. This reason can be attributed to two main factors:

- *Limitations of the Training Dataset*: The training dataset was generated from videos containing markers moving randomly, with some variation in speed as discussed in Section 3.3.2. Although videos take into account the random movement speeds of the markers, these movement speeds are generally slow and are not entirely the same as the faster movements in



a b

Figure 4.5: **A: collecting the independent data, B: a data sample example**



a b

Figure 4.6: **A: event frame containing noise, B: light reflection during collection**

practical scenarios. We mentioned the inconsistency of the event frame output of the event camera in Section 2.1.2. This difference between the slower speeds in the training dataset and the faster movements in the new, independent dataset contributed to differences in the event camera event frame outputs. As a result, the quality of the model's predictions was diminished.

- *Environmental Interferences*: During the training data collection, the screen material used to display the videos minimizes light reflection and ensures clean data capture. In contrast, the mobile phone screen used in collecting independent datasets sometimes produced reflections, introducing significant noise and interference. Figure 4.6 shows an example of an event frame output affected by the light reflection on the screen.

To address the first issue, in future work, incorporating new data samples that include faster marker movements and retraining the model can enhance the model's performance.

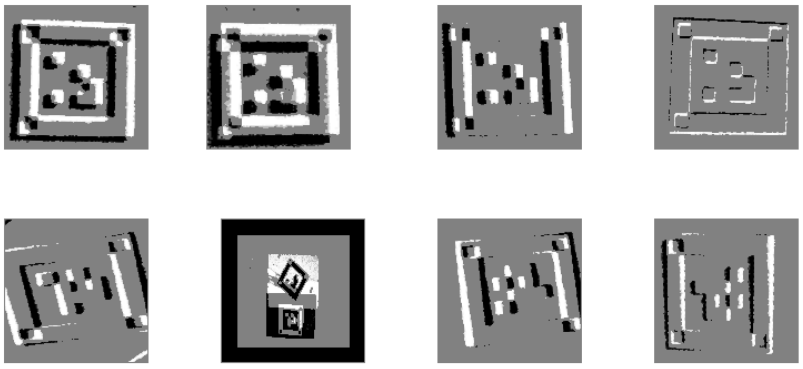


Figure 4.7: **The first line shows examples of correctly created marker candidates. The second line shows examples of incorrectly created marker candidates, which don't have complete fiducial markers displayed or are in the wrong orientations.**

4.3 Evaluation on Accuracy of Creating Marker Candidates

This section focuses on testing and evaluating the accuracy of creating marker candidates, utilizing the independent dataset referenced in Section 4.2.2 with the prediction results from *EvMarkNet*. To determine the successful establishment of a marker candidate, the following criteria must be met:

- The marker candidate, when extracted from the event frame and 1.3x rescaled, should clearly display the entire fiducial marker.
- Building on the first criterion, if three indicators are visible at the upper left corner, lower left corner, and upper right corner of the marker candidate, it confirms that the candidate has the correct orientation and is correctly established.

Figure 4.7 depicts a correctly created marker candidate.

After conducting tests on 500 data samples, the accuracy of the final marker candidate reaches **83.4%** and **417** correct candidates are created. While the prediction results from *EvMarkNet* only achieved 65% IoU due to occasional inaccuracies in edge prediction or undetected direction indicators, the algorithm we designed effectively compensates for these errors, as demonstrated in Figure 4.8.

However, for the same reason, incorrect candidate creations are caused by inaccuracies in edge prediction or undetected direction indicators. Although our method addresses most issues effectively, it struggles with a particular case, as illustrated in Figure 4.9. In this instance, although the white polygon has four vertices and is not clustered together, two of the vertices are actually offset significantly, and our method cannot detect this error. Furthermore, another situation is illustrated in Figure 4.10, the direction indicator which should be within the green circle by design doesn't appear. This issue arises because the

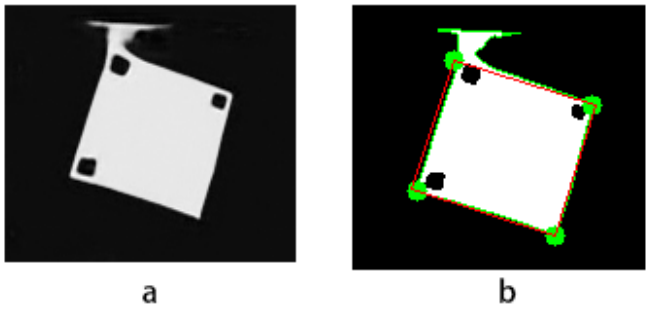


Figure 4.8: **There is an obvious error in the edge of the marker in image A, but the algorithm can still estimate the correct edge**

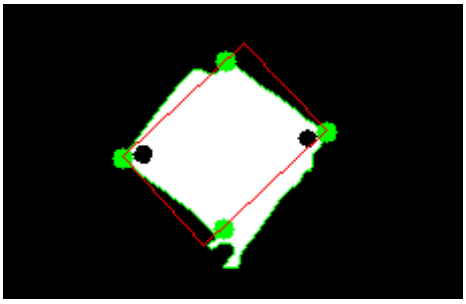


Figure 4.9: **An example of wrong quadrilateral estimation**

light intensity change in this area is not noticeable enough for the event camera to generate enough events. Consequently, *EvMarkNet* can't detect it. Our algorithm also produces identical results when calculating the sum of the color values in both the green and blue circles. This similarity in results makes it impossible to determine the location of the last indicator.

4.4 Evaluation on Accuracy of Decoding

This section evaluates our decoding algorithm using the correctly established marker candidates from the previous section. The test involves a simple comparison of the information decoded from the candidates against the information they are intended to carry. After testing, the decoding accuracy for the 446 correctly established marker candidates is found to be **93.4%**. Among the inaccurately decoded information, the error rate for individual bits is **12.2%**. This translates to approximately **43** incorrectly decoded bits out of a total of 6244 bits transmitted across 446 pieces of information, resulting in a bit error rate of about **6.8%**.

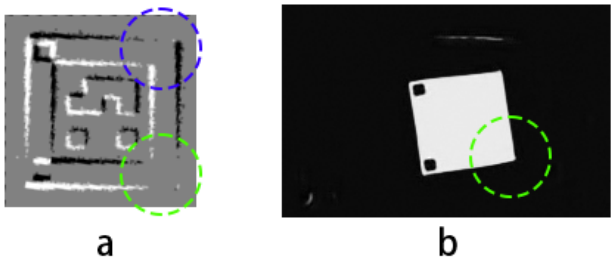


Figure 4.10: **An example of incorrect candidate caused by an undetected direction indicator**

When observing the test results, it is found that our method has a high error tolerance. Processing on-event and off-event separately enables complementary decoding results, enhancing overall accuracy. As depicted in Figure 4.11, according to the designated information for the marker, the encodable bit in cell (2, 3) should be encoded as '1'. In Figure 4.11a, the white pixels within the cell (2, 3) of the off-event are distributed both horizontally and vertically, suggesting that the arrangement is not due to pixel overflow. Thus, from our analysis, this cell should be decoded as '1'. However, due to the application of a dynamic threshold, this cell is mistakenly treated as '0'. In contrast, in Figure 4.11b, the white pixels in cells (2, 3) are sufficiently apparent to be correctly classified as '1'. By applying an OR operation between the results from both candidates, the error in the decoding of the off-event candidate is effectively neutralized. Consequently, the final decoding result for this cell is correctly determined as '1'.

However, the dynamic threshold can be a double-edged sword, it sometimes misclassifies cells. For instance, cell (2,2) in Figure 4.12 should be decoded as '0'. However, because the proportion of white pixels exceeds the dynamic threshold by 2%, it is mistakenly classified as '1'. This illustrates how sensitive the decoding process is to threshold settings, leading to potential inaccuracies.

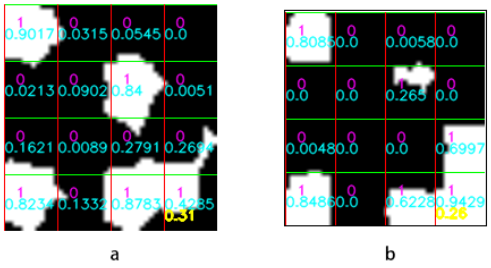


Figure 4.11: **A: the decoding result on an off-event candidate, B: the decoding result on an on-event candidate. The cell (2,3) in both candidates are complementary.**

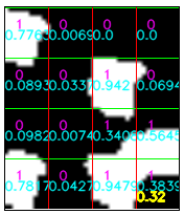


Figure 4.12: **Mistaken classification in cell (2,2).**

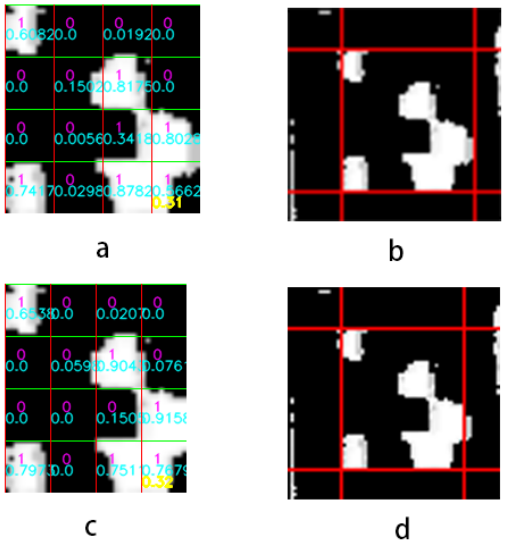


Figure 4.13: **A, B: the decoding result of an on-event candidate and its corresponding output of subsequent refinement. C, D: the decoding result after manually adjusting the right edge of the encodable areas and its corresponding output of subsequent refinement.**

Sometimes, inaccuracies in the dynamic threshold calculation can cause low-accuracy edge determinations in the encodable area by the second refinement. For example, Figure 19b shows the default output of our second refinement algorithm, which contributes to the decoding error of cell (2, 2) depicted in Figure 19a. Figure 19d illustrates the output after manually adjusting the edge to 3 pixels left. This adjustment leads to fewer white pixel leakage and correct decoding of the cell (2, 2) in Figure 19c. This demonstrates the critical role of precise edge alignment in reliable decoding.

Another source of decoding errors is the inaccurate calculation of the marker's white frame edges during the first refinement. As illustrated in Figure 4.14, the pink solid line correctly denotes the left edge of the white frame, but the red solid line, which is supposed to represent the right edge is wrong. The yellow dotted line is added to this figure manually, indicating the expected position for the right edge. This error occurs because the right edge of the frame within the candidate is uneven, causing the white pixels to be scattered. Each column

Figure 4.14: **An incorrect first refinement result. The purple solid line is the correct calculated position of the left edge of the white frame, the red line is the incorrect calculated position of the right edge of the frame, and the yellow dotted line is the manually added expected right edge position.**

Table 4.2: **Final evaluation result of the entire pipeline.**

Total Time	Decoding Accuracy	CPU RAM	Min GPU Video RAM
26.9 ms	77.9%	670 MB	750 MB

near the yellow dotted line contains significantly fewer white pixels compared to the column marked by the red solid line. Consequently, when the algorithm scans lines, the yellow dotted line does not meet the threshold conditions. In this scenario, the miscalculation during the first refinement disrupts subsequent stages, preventing normal execution of the second refinement and decoding processes. As a result, all encodable bits are considered undecidable. To address this issue, incorporating additional image processing algorithms to smooth the image edges is necessary.

4.5 Overall Evaluation

In this section, we evaluate the entire pipeline by processing 500 data samples from the independent dataset mentioned in Section 4.2.2. The evaluation involves running these samples through our pipeline and then comparing the final decoded output with the original information contained in the samples. The hardware configuration for this test is changed to Intel i7-11800H CPU, Nvidia RTX3060 GPU, and 8GB of available memory allocated to this task. Unlike the previous test of *EvMarkNet*, this setup represents a lower-performance configuration to illustrate the efficiency of our pipeline. Table 4.2 shows the result of the final evaluation.

The total operation time of the entire pipeline is only 26.9 ms. If the event camera collects data at an interval of 27 ms, theoretically up to 37 pieces of information can be transmitted per second. The final decoding accuracy of our method is around 77.9%. This is the same as the cumulative loss multiplication results of our previous tests. What surprised us most is that the entire pipeline runs much faster on a low-performance configuration than *EvMarkNet* on a high-performance configuration. In addition, the CPU RAM usage of the entire pipeline is only 670MB. It is even more worth mentioning that although we allocated 8GB of video memory to the entire task (mainly for running *EvMarkNet*), the minimum video memory actually required is only 700MB.

In order to have a deeper understanding of the performance bottleneck of the entire pipeline, we perform a more detailed evaluation. Each step in the pipeline is timed separately. The time distribution of each step is depicted in Figure 4.15. It can be clearly observed that the first step of using a neural network for detection is the most time-consuming part of the entire pipeline, 24.2 ms, taking up 90% of the total duration. This high time consumption is a trade-off when applying neural networks. As the payback for the extra computation time, the network gains versatility and enhanced stability across various application scenarios.

The decoding phase is the second most time-consuming part of the process, taking 2.3 ms and accounting for 8.6% of the total computing time. This duration is reasonable because the decoding part needs to decode both on-event and off-event candidates, which contain repetitive operations. In addition, the decoding process involves exhaustive traversal algorithms and may need to scan every row and column in the candidate. This becomes a potential factor in increasing the total computation time.

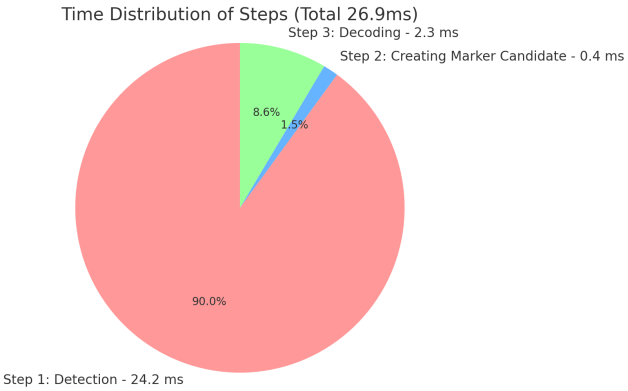


Figure 4.15: **A pie chart displaying the computation time distribution of each step of the pipeline.**

Chapter 5

Conclusions

In this thesis, a method for high-speed moving objects based on event cameras has been proposed. This method has the characteristics of being able to detect and identify moving objects based on event cameras and interact with recognized objects. In addition, this method is versatile for different application scenarios and is suitable for embedded platform devices.

The whole method is designed to revolve around a keyword and the common features between objects. We use optimized fiducial markers to introduce external consistent features to all objects. The detection of the marker is considered the detection of the objects themselves. This ensures our method's versatility and interactive capability. We first utilize a custom-developed neural network called *EvMarkNet* to detect the fiducial marker in the event frame output by the event camera, and create a dataset for training the network. The prediction result of *EvMarkNet* provides the approximate position of the fiducial marker in the event frame. From this, we extract the contents of fiducial markers from the event frame and apply a homography transformation to form marker candidates suitable for decoding. These candidates are split into on-event and off-event candidates, each undergoing two rounds of region segmentation refinements to define the encodable areas within markers. Finally, the refined encodable area is gridded, and each cell in the grid is binary classified with a dynamic threshold. The decoding results from the on-event and off-event are combined using an OR operation to produce the final result.

We conducted qualitative and quantitative tests and evaluations of our proposed method. First, we assess different versions of *EvMarkNet*. With version v3 showing the highest Intersection over Union (IoU), it proves its prediction quality and the rationality of our design approach. Next, we separately test each step in the pipeline. The high decoding accuracy shows the effectiveness of our method in analyzing fiducial markers and its tolerance to errors. We finally evaluated the entire pipeline and confirmed that it has low computation time, low storage and memory usage, and high accuracy, even on a lower-performance platform.

Based on these findings, we conclude that the method proposed in this thesis is well-suited for high-speed moving object recognition based on event cameras.

5.1 Limitations

Although our method has shown good performance, it still has certain limitations:

- The event camera we use is equipped with a manual focus lens. This constraint means if it's deployed on unmanned equipment, the distance between the camera and the object must remain constant to avoid defocusing, which might lead to serious performance degradation.

- Our approach relies heavily on a specific type of fiducial marker, AprilTag16. And the entire dataset used to train *EvMarkNet* is based on this kind of marker. As a result, introducing new types of markers or customizing existing ones (for instance, making structural modifications to our markers) could potentially reduce the accuracy of *EvMarkNet*'s predictions.

- Our method is limited to the case where there is only one fiducial marker in the event frame. For multiple markers, our method will only decode the marker with the largest area in the event frame.

- The fiducial marker originally contains content with a fixed shape and ID, which makes the traditional fiducial marker detection method have an error compensation mechanism. Even if some bits are not decoded correctly, this mechanism can be used to get close to the most similar shape and ID. This mechanism is particularly useful when the marker is partially obscured. However, we use fiducial markers to transmit completely random information, which makes the information decoding accuracy almost zero when the marker is obscured.

5.2 Future Works

Due to time constraints, many design details could not be optimized, and many tests could not be performed. The following lists possible future optimization directions and tests that can be used to demonstrate the advantages of this method.

5.2.1 Possible Optimization Directions

Neural Network with Better Prediction Quality

As the first step in the pipeline, high-quality prediction results from the *EvMarkNet* can make the subsequent candidate creation and decoding more efficient, with fewer candidate orientation problems, bias problems on the edges of the encodable area, and binary classification errors due to white pixel leakage. We believe that there are models with faster prediction speed and better output quality. Exploring the latest models could prove beneficial. For example, the recently released *MobileSam* [39] offers improvements in both model size and prediction speed compared to *EvMarkNet*, though its prediction quality remains to be evaluated.

Figure 5.1: **A: the event frame of a marker moving towards the top left, B: events are divided into two groups according to their timestamps, where red pixels are old events and green pixels are new events. The moving direction can bee seen from B.**

Fiducial Marker with More Encodable Bits

The fiducial marker used in our proposed method has only 16 encodable bits, and two of them are used as restriction bits. A marker with more bits might be feasible. More bits means more information combinations and effective encoding systems. Such as more restriction bits, or header and CRC checksum. This will significantly improve the transmission stability of information.

Dynamic thresholds for more flexibility

As mentioned in Section 4.4, sometimes dynamic thresholds can lead to errors in binary classification. In this case, a more flexible dynamic threshold is needed. For instance, we can consider the distribution of white pixels in each cell in the gridded encodable area. If most of the white pixels in some cells are distributed at the edge, more weights can be added to them when calculating the threshold. In addition, the white pixel density of each cell can be calculated, and more weights can be added to cells with high white density.

EvMarkNet Implement with TensorFlow Lite

TensorFlow Lite [11] was proposed by Google to optimize and compress models trained based on TensorFlow while maintaining the output quality of the model as much as possible and reducing the size of the model. To implement our model on embedded platforms, TensorFlow Lite is a good choice.

Utilization of Time Density of Event Camera

As mentioned in 2.1, in return for sacrificing spatial resolution, the event camera has a high temporal resolution. Each event output from the event camera has its own timestamp. An application of using their timestamp to estimate object moving direction is depicted in Figure 5.1. This feature, as the most significant advantage of event cameras, should be fully utilized. The processing of event frames is essentially the processing of two-dimensional images, which do not

contain any three-dimensional time information. It is necessary to develop a network designed to accept independent events as inputs. Such a network would analyze the spatiotemporal relationships of events, enabling it to make more accurate predictions.

5.2.2 Possible Tests to Conduct

Comparison with Currently Popular Image Segmentation Models

We want to compare our *EvMarkNet* with the current popular models for image segmentation to see what its advantages and disadvantages are. So we need to use the same network structure like *SegNet* [3] or DeepLab [6] to train for the task of fiducial marker detection, or use pre-trained models such as *SAM* [16] to transfer train or fine-tune. In this way, we can make a reasonable comparison.

Comparison with Traditional Fiducial Marker Detection Method

Some functions and features for traditional fiducial marker recognition should be added to our pipeline, such as a marker shape-matching mechanism and error compensation mechanism. Then it's possible to compare our method's advantages and disadvantages with those of traditional fiducial marker recognition methods.

Execution on embedded system platforms

One of the goals of this paper is to design a method suitable for embedded platforms. Although our method has been shown to have low storage and memory consumption suitable for embedded platforms, the specific performance on embedded platforms is still unknown. Although we tried to run our method on NVIDIA TX2, it was unsuccessful due to CUDA software compatibility issues. We hope to have the opportunity to verify our method on embedded platforms in the future.

Bibliography

[1] Markus Vincze Adam Loch, Germain Haessig. Event-based high-speed low-latency fiducial marker tracking. `https://arxiv.org/abs/2110.05819`, 2021.

[2] Amazon. Amazon drones can now fly farther following faa approval. `https://www.aboutamazon.com/news/transportation/amazon-drone-prime-air-expanded-delivery-faa-approval`, 2024.

[3] Vijay Badrinarayanan, Alex Kendall, and Roberto Cipolla. Segnet: A deep convolutional encoder-decoder architecture for image segmentation. url=https://arxiv.org/abs/1511.00561, 2016.

[4] Soumen Biswas and Ranjay Hazra. Robust edge detection based on modified moore-neighbor. *Optik*, 168:931–943, 2018.

[5] John Canny. A computational approach to edge detection. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, PAMI-8(6):679–698, 1986.

[6] Liang-Chieh Chen, Yukun Zhu, George Papandreou, Florian Schroff, and Hartwig Adam. Encoder-decoder with atrous separable convolution for semantic image segmentation. url=https://arxiv.org/abs/1802.02611, 2018.

[7] Sankeerth Durvasula, Yushi Guan, and Nandita Vijaykumar. Ev-conv: Fast cnn inference on event camera inputs for high-speed robot perception. *IEEE Robotics and Automation Letters*, 8(6):3174–3181, 2023.

[8] M. Fiala. Artag, a fiducial marker system using digital techniques. In *2005 IEEE Computer Society Conference on Computer Vision and Pattern Recognition (CVPR'05)*, volume 2, pages 590–596 vol. 2, 2005.

[9] Rafael Medina-Carnicer Francisco J. Romero-Ramirez, Rafael Muñoz-Salinas. Speeded up detection of squared fiducial markers. *Image and Vision Computing*, 76:38–47, 2018.

[10] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022.

[11] Google. Tensorflow lite. `https://www.tensorflow.org/lite/guide`, 2024.

[12] Kaiming He, Xiangyu Zhang, Shaoqing Ren, and Jian Sun. Identity mappings in deep residual networks. `https://arxiv.org/abs/1603.05027`, 2016.

[13] iniVation. Dv. `https://docs.inivation.com/software/dv/index.html`, 2024.

[14] Shaoqing Ren-Jian Sun Kaiming He, Xiangyu Zhang. Deep residual learning for image recognition. `https://arxiv.org/abs/1512.03385`, 2015.

[15] N. Kanopoulos, N. Vasanthavada, and R.L. Baker. Design of an image edge detection filter using the sobel operator. *IEEE Journal of Solid-State Circuits*, 23(2):358–367, 1988.

[16] Alexander Kirillov, Eric Mintun, Nikhila Ravi, Hanzi Mao, Chloe Rolland, Laura Gustafson, Tete Xiao, Spencer Whitehead, Alexander C. Berg, Wan-Yen Lo, Piotr Dollár, and Ross Girshick. Segment anything. *arXiv:2304.02643*, 2023.

[17] Jianing Li, Siwei Dong, Zhaofei Yu, Yonghong Tian, and Tiejun Huang. Event-based vision enhanced: A joint detection framework in autonomous driving. In *2019 IEEE International Conference on Multimedia and Expo (ICME)*, pages 1396–1401, 2019.

[18] Patrick Lichtsteiner, Christoph Posch, and Tobi Delbruck. A 128× 128 120 db 15 μs latency asynchronous temporal contrast vision sensor. *IEEE Journal of Solid-State Circuits*, 43(2):566–576, 2008.

[19] M. Litzenberger, C. Posch, D. Bauer, A.N. Belbachir, P. Schon, B. Kohn, and H. Garn. Embedded vision system for real-time object tracking using an asynchronous transient vision sensor. In *2006 IEEE 12th Digital Signal Processing Workshop 4th IEEE Signal Processing Education Workshop*, pages 173–178, 2006.

[20] Jonathan Long, Evan Shelhamer, and Trevor Darrell. Fully convolutional networks for semantic segmentation. =https://arxiv.org/abs/1505.04597, 2015.

[21] J. MacQueen. Some methods for classification and analysis of multivariate observations. 1967.

[22] Nico Messikommer*, Carter Fang*, Mathias Gehrig, and Davide Scaramuzza. Data-driven feature tracking for event cameras. *IEEE Conference on Computer Vision and Pattern Recognition*, 2023.

[23] Anton Mitrokhin, Cornelia Fermüller, Chethan Parameshwara, and Yiannis Aloimonos. Event-based moving object detection and tracking. In *2018 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS)*, pages 1–9, 2018.

[24] Feng Ning, D. Delhomme, Y. LeCun, F. Piano, L. Bottou, and P.E. Barbano. Toward automatic phenotyping of developing embryos from videos. *IEEE Transactions on Image Processing*, 14(9):1360–1371, 2005.

[25] Chethan M. Parameshwara Nitin J. Sanket, Chahat Deep Singh and Yiannis Aloimonos Cornelia Fermüller, Guido C. H. E. de Croon. Evpropnet: Detecting drones by finding propellers for mid-air landing and following. `https://arxiv.org/abs/2106.15045`, 2021.

[26] Edwin Olson. AprilTag: A robust and flexible visual fiducial system. In *Proceedings of the IEEE International Conference on Robotics and Automation (ICRA)*, pages 3400–3407, May 2011.

[27] OpenCV. Opencv modules. `https://docs.opencv.org/4.10.0/`, 2024.

[28] Nobuyuki Otsu. A threshold selection method from gray-level histograms. *IEEE Trans. Syst. Man Cybern.*, 9:62–66, 1979.

[29] Sergio Garrido Rafael Muñoz. Aruco in opencv. `https://www.uco.es/investiga/grupos/ava/portfolio/aruco/`, 2015.

[30] UZH Robotics and Perception Group. Event-based, 6-dof pose tracking for high-speed maneuvers using a dynamic vision sensor. url=https://www.youtube.com/watch?v=LauQ6LWTkxM, 2014.

[31] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation. =https://arxiv.org/abs/1411.4038, 2015.

[32] Alan Saalfeld. Topologically consistent line simplification with the douglas-peucker algorithm. *Cartography and Geographic Information Science*, 26(1):7–18, 1999.

[33] Nitin J. Sanket, Chethan M. Parameshwara, Chahat Deep Singh, Ashwin V. Kuruttukulam, Cornelia Fermüller, Davide Scaramuzza, and Yiannis Aloimonos. Evdodgenet: Deep dynamic obstacle dodging with event cameras. In *2020 IEEE International Conference on Robotics and Automation (ICRA)*, pages 10651–10657, 2020.

[34] Hamid Sarmadi, Rafael Muñoz-Salinas, Miguel A. Olivares-Mendez, and Rafael Medina-Carnicer. Detection of binary square fiducial markers using an event camera. *IEEE Access*, 9:27813–27826, 2021.

[35] Satoshi Suzuki and KeiichiA be. Topological structural analysis of digitized binary images by border following. *Computer Vision, Graphics, and Image Processing*, 30(1):32–46, 1985.

[36] PROPHESEE Metavision Technologies. Prophesee image deblurring frame-based + event-based vision. url=https://www.youtube.com/watch?v=1LXt4ScwfXA, 2023.

[37] uzh rpg. Event-based vision resources. `https://github.com/uzh-rpg/event-based_vision_resources`, 2024.

[38] Qing Wang, Chenren Xu, Supeng Leng, and Sofie Pollin. When autonomous drones meet driverless cars. In *Proceedings of the 16th Annual International Conference on Mobile Systems, Applications, and Services*, page 514, 2018.

[39] Chaoning Zhang, Dongshen Han, Yu Qiao, Jung Uk Kim, Sung-Ho Bae, Seungkyu Lee, and Choong Seon Hong. Faster segment anything: Towards lightweight sam for mobile applications. `https://arxiv.org/abs/2306.14289`, 2023.

[40] Chunhui Zhao, Yakun Li, and Yang Lyu. Event-based real-time moving object detection based on imu ego-motion compensation. In *2023 IEEE International Conference on Robotics and Automation (ICRA)*, pages 690–696, 2023.

[41] Xu Zhao, Wenchao Ding, Yongqi An, Yinglong Du, Tao Yu, Min Li, Ming Tang, and Jinqiao Wang. Fast segment anything. url=https://arxiv.org/abs/2306.12156, 2023.

[42] Xu Zheng, Yexin Liu, Yunfan Lu, Tongyan Hua, Tianbo Pan, Weiming Zhang, Dacheng Tao, and Lin Wang. Deep learning for event-based vision: A comprehensive survey and benchmarks. `https://arxiv.org/abs/2302.08890`, 2023.