LOOMING OBJECT DETECTION WITH EVENT-BASED CAMERAS

IFFATUR RIDWAN
Bachelor of Science, Military Institute of Science and Technology, 2015

A Thesis
Submitted to the School of Graduate Studies
of the University of Lethbridge
in Partial Fulfillment of the
Requirements for the Degree

MASTER OF SCIENCE

Department of Mathematics and Computer Science
University of Lethbridge
LETHBRIDGE, ALBERTA, CANADA

LOOMING OBJECT DETECTION WITH EVENT-BASED CAMERAS

IFFATUR RIDWAN

Date of Defence: December 7, 2017

Dr. Howard Cheng Supervisor	Associate Professor	Ph.D.
Dr. John Zhang Committee Member	Associate Professor	Ph.D.
Dr. Matthew Tata Committee Member	Associate Professor	Ph.D.
Dr. Amir Akbary Chair, Thesis Examination Committee	Professor	Ph.D.

Dedication

I dedicate this thesis to my parents and my sister for believing in me.

Abstract

We present a looming object detection method for event-based cameras. Event-based cameras detect events asynchronously which eliminates the unnecessary computation required for the conventional frame-based cameras. There are two main parts of this method. In the first part, we develop an event-based optical flow algorithm. The algorithm is based on Reichardt motion detectors inspired by the fly visual system and has a very low computational requirement for each event received from the event-based camera. In the second part, we develop an algorithm to detect looming objects using the output from the first algorithm. This proposed method is only sensitive to significant log-luminance changes, which results in low energy consumption. We have performed several experiments with our method using the Davis Dynamic Vision Sensor (DVS) which is an event-based camera. Experimental results show that our event-based looming detection algorithm accurately detects looming objects in all cases when there is a single object moving in the scene. It also does not report looming when no objects are looming. Our algorithm is fast and operates in real-time, requiring only microseconds to process each event.

Acknowledgments

First and foremost, I would like to thank Allah for bringing me where I am today. After that, my deep gratitude goes first to Dr. Howard Cheng, whom I cannot thank enough. Thank you for being patient when I was difficult to handle. Thank you for offering me this wonderful opportunity, for believing in me, even if it only was for just a fraction of seconds!

I am grateful to my committee members, Dr. John Zhang and Dr. Matthew Tata, for their guidance and support throughout my research. I would also like to thank the Tata lab for providing me research equipment. Thanks to Scott Stone and Cody Barnson for all their help.

My appreciation also extends to the people I met in my home far away from home, Lethbridge. Thank you everyone who kept supporting me during my mood-swings. Special note of thanks to those awesome people who never showed any sign of resentment even when I was wrong.

Finally, my parents, along with my sister and my brother-in-law. Without the inspiration and the drive they have given me, I might not be the person I am today. My sister has been my defender, counsellor and motivator by all means. Thank you for being there. Ammu, Abbu and Suappi - I love you.

“Allah does not burden a soul beyond that it can bear...” (Qur’an, 2:286)—Thanks to them who made me believe this.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Computer vision is a field in which algorithms are designed to analyze images or videos, in order to extract useful information. Video processing is characterized by the extraction of information from a video stream to generate some useful information and decisions. There are many applications of video processing including object tracking, video enhancement, motion detection, information retrieval, object identification, etc. In robotics, there are a number of problems that require extracting particular information from a video stream obtained from cameras on the robot. One such problem is looming object detection.

Looming is an optical phenomenon of rapid expansion of any object. Looming detection is a particular type of motion detection which detects whether an object is approaching towards the viewer or vice versa [25]. Traditionally frame-based cameras are used for such tasks. However, they have a relatively low frame rate. Thus, the speed of looming detection on systems based on these cameras is limited by the frame rate. These cameras capture data regularly even when there is no motion in the scene, wasting both computational resources and electrical power. Conventional algorithms often need to perform processing to identify regions of interest, and to ignore the static background. This processing requires extra computation and storage.

Neuromorphic engineering is a concept in which biological nervous systems are simulated by Very Large Scale Integration (VLSI) systems with electronic analog circuits [19]. This particular field works to replace conventional transistor-based circuits with neuron-like architecture inspired by the human brain. The Davis Dynamic Vision Sensor (DVS) is an example of such a system. The DVS is a different type of camera designed to simulate the human retina [14]. It only sends data when there is a significant change in the log-luminance in any individual pixel. Log-luminance value is the logarithmic value of luminance. Logarithmic value is used to represent very high variations that could not be represented efficiently on the linear curves. These significant changes in log-luminance are called events, and they may be reported by the camera asynchronously at any time as soon as the event is detected by the camera. The DVS is an event-based camera. Only significant changes are reported and need to be processed, resulting in lower computational power requirement, as well as faster reaction times. In contrast, frame-based cameras sample every pixel for each frame and report the luminance of each pixel at regular intervals. However, algorithms for frame-based cameras cannot be used directly for these neuromorphic cameras. New event-based algorithms have to be developed.

In the modern world, we need faster reaction times in applications such as robotics or vehicle navigation, or even for people with special needs to use in their regular life [27]. Our goal is to propose a system which can react significantly faster than systems based on conventional frame-based cameras. In this thesis, we propose a combination of event-based algorithms to detect looming objects. We show by experiments that the proposed algorithms are accurate and fast.

To detect motion occurring in a scene, we design a modified version of Reichardt detector [23]. The Reichardt detector is a non-linear mechanism which uses correlation computation to find out the difference between two adjacent input signals in order to detect motion. This model is inspired by the visual system of flies. We have designed our version of Reichardt detector to compute optical flow, indicating detected motion in a scene.

For looming object detection, our approach is to detect the edges of the object first. In order to identify the edges we group neighbour pixels with motion in similar directions. Only the boundary pixels in the group are selected to be considered. Next the directions of the edge or boundary pixels of the object are analyzed. If the direction is pointing away from the center of the object, our algorithm reports that a looming object is detected in the scene.

Experimental results show that our optical flow algorithm is accurate and can be performed in real time. Each event can be processed in about 2 microseconds on average. Our looming detection algorithm is accurate in all cases tested in which there is a single object in the scene, and it can also process each event in under 3 microseconds.

1.1 Contribution

There are several works already done on looming detection using conventional cameras. The Davis DVS produces outputs that are very different from conventional frame-based cameras. We have proposed a method to use the output of the DVS to compute optical flow using our variation of Reichardt Detector. This optical flow algorithm detects motion in any pixel of the scene. This information is then used to detect a looming object. To the best of our knowledge, this is the first algorithm for looming object detection for event-based cameras. Our approach takes advantage of the properties of the DVS in order to obtain an efficient real-time algorithm that accurately detects looming objects.

Our proposed event-based optical flow algorithm has been published in the International Conference Image Analysis and Recognition (ICIAR) 2017 [24].

1.2 Organization of Thesis

In this thesis, we will discuss the background and the motivations of our research in Chapter 2. In Chapter 3, we will develop our event-based optical flow algorithm and demonstrate its performance. In Chapter 4, we will explain our method of detecting looming objects as well as the algorithm with experiments and results. Finally in Chapter 5, conclusion and possible future works of the thesis will be discussed.

Chapter 2

Background

In this chapter we discuss some backgrounds of the methods and terms we have used in this research. We also explain the definitions and previous works on those topics.

2.1 Reichardt Detector

In 1956, Hassenstein and Reichardt proposed a correlation based movement detection model which was originally inspired by the visual system of flies [23]. The mechanism underlying this method of motion detection is non-linear. In this proposed method, movement is detected by computing the correlation of the appropriately filtered signals of two neighbouring retinal input channels.

The Reichardt detector consists of two mirror symmetric sub-units (Figure 2.1). In each sub-unit, the luminance values as measured in two adjacent image locations (one of them is delayed by a low pass filter) are multiplied together. The product can be viewed as the correlation of the two locations at different times. The resulting output signal is the difference between the multipliers of the two sub-units, and indicates the direction of movement [6].

More formally, let $f(x,y,t)$ be the luminance value at location (x,y) at time t. We also let Δx and Δy be the offset between the locations of two adjacent sub-units, and Δt be the time delay. Then the output of the Reichardt Detector is

$$RD(f,x,y,t,t') = f(x',y',t)\cdot f(x,y,t') - f(x,y,t)\cdot f(x',y',t'), \tag{2.1}$$

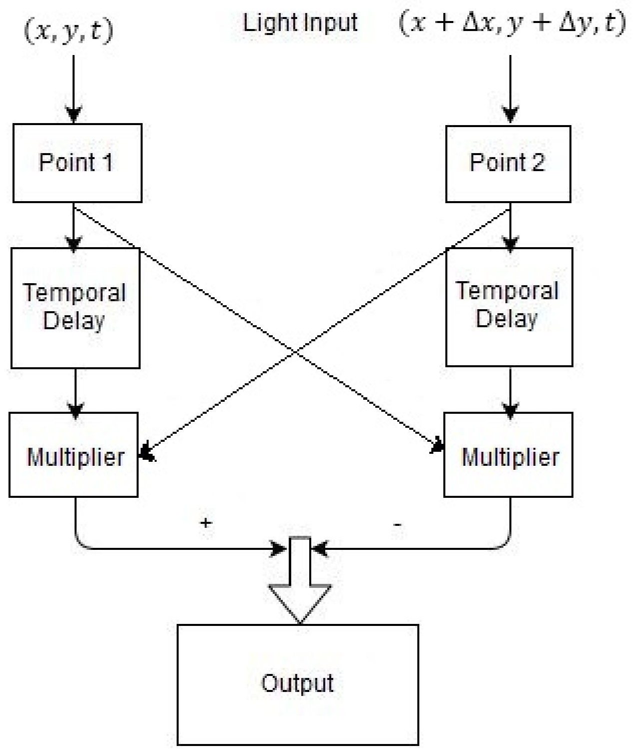


Figure 2.1: A model of Reichardt Detector.

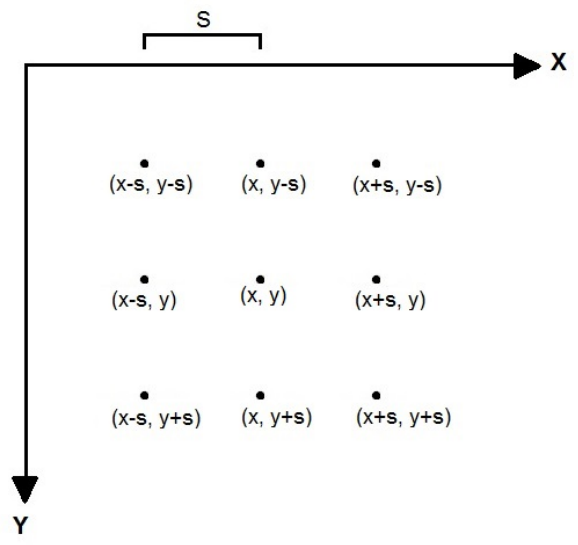


Figure 2.2: Neighbourhood of (x, y). Here *s* is the spatial displacement.

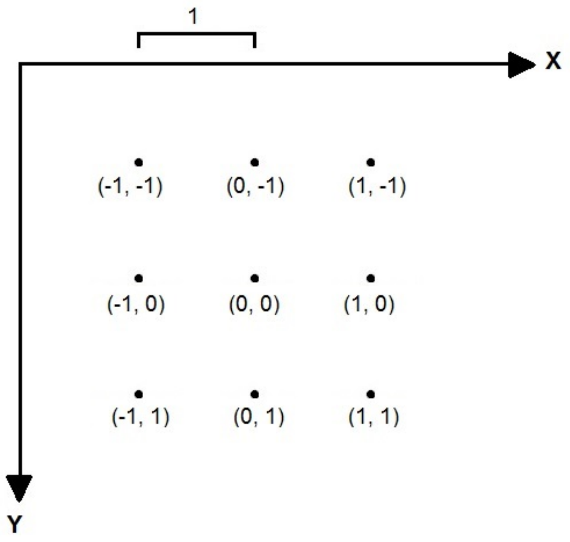


Figure 2.3: $(\Delta x, \Delta y)$ for each pixel in the neighbourhood of (x, y), where $s = 1$.

where $x' = x + \Delta x$, $y' = y + \Delta y$, and $t' = t + \Delta t$. If $|RD(f, x, y, t, t')|$ exceeds a threshold T_{RD}, motion is detected along the direction $(\Delta x, \Delta y)$ or $-(\Delta x, \Delta y)$, and the sign of $RD(f, x, y, t, t')$ indicates the actual direction. Adjacent pixels inside stationary objects will not be detected because the difference is close to zero.

Each Reichardt detector can only detect motion at a particular location with velocities of $\pm(\Delta x/\Delta t, \Delta y/\Delta t)$. Different velocities can be detected by varying Δx, Δy, and Δt. In practice, this is accomplished by a grid of Reichardt detectors on the pixels of successive frames, so that motion can be detected at all locations in a number of pre-defined directions (e.g. the 8 compass directions).

A Reichardt detector detects motion at a specific location in a specific direction. The proposed system by Hassenstein and Reichardt [23] is based on using multiple Reichardt detectors in a grid to detect motion in multiple directions. The neighbourhood of a location in the grid is shown in Figure 2.2 and Figure 2.3. The input to these sub-units are the signals at (x, y) and its neighbours at time t and t'. The vector $(\Delta x, \Delta y)$ defines the direction in which motion detection is done. The photoreceptors are placed at a distance of $\sqrt{(\triangle x)^2 + (\triangle y)^2}$

apart.

There are some cases in which detection of motion by the Reichardt detector can be ambiguous. For example in a pattern such as a checkerboard, in which the pattern is repeated symmetrically, the grid of detectors cannot identify the motion as the same pattern is found throughout the whole frame (Figure 2.4). As it is seen in the figure, the centre cell of the pattern strongly correlates with all four neighbourhood cells after horizontal movement. As a result, it is very difficult to distinguish the new position of the center cell. The new position can be any one of the four neighbourhood cells, and the grid of Reichardt detector reports motion in all four directions. It is not possible to identify which of the four movements actually took place. As for diagonal movement, no motion is detected at all. This is because even after the movement the neighbourhood cells retain the same pattern after Δt. This problem can be solved by choosing smaller values of Δt, Δx and Δy. If Δt, Δx and Δy are decreased accordingly, the changes in the pattern can be detected. because the new pattern would not align with the original grid.

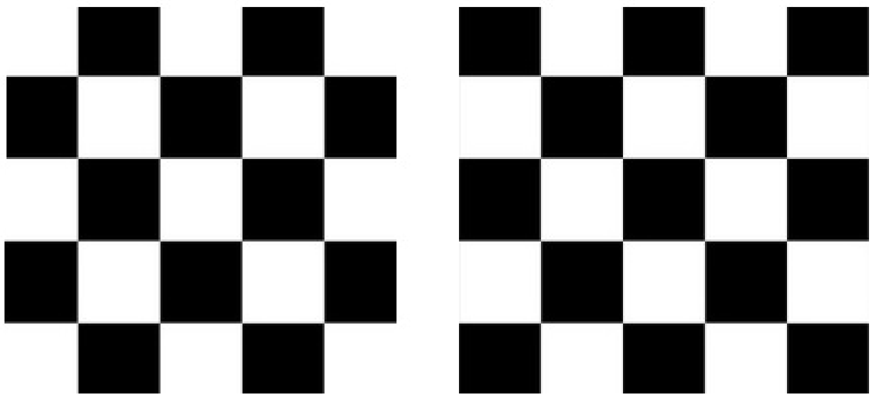


Figure 2.4: An example of checkerboard pattern before and after horizontal movement.

2.2 Optical Flow

Optical flow has different definitions in image processing and neuroscience. Though this research is an interdisciplinary project, we are drawing definitions and using the terminologies from image processing. In image processing, optical flow is a vector field indicating the apparent motion of each pixel in a scene. When objects in the scene move relative to the viewer, a pattern of motion emerges.

It is of particular interest in visual neuroscience because there are optical flow sensitive cells in certain parts of the visual system. That is, there are neurons that become excited if and only if there is optical flow visible. This is interesting because such neurons must integrate local texture and motion information across a relatively large field, yet they do so very fast. These are some of the fastest responses in the visual cortex [29].

Optical flow is a typical solution for many types of movement detection. There are many optical flow algorithms designed for conventional cameras [2, 26, 13]. These algorithms are often based on the principles in calculus such as partial derivatives. SpikeNet Technology [26] has developed an efficient processing algorithm [5], which is based on neurobiological principles. This algorithm is capable of efficient real-time calculation of optical flow. At each point in the image, the system can provide a direction of motion in 8 or 16 different possible directions together with a value for the distance (in pixels) of movement between two consecutive frames. Two threshold parameters are used: one is to determine the minimum amount of edge energy required to trigger a response, and the second one is used to calculate the minimum luminance change between two frames for signal motion. In case of very low values of thresholds, a dense motion map can be generated where all pixels are generating a motion value. The noise can be controlled by changing the threshold value. Even though the most significant advantage of this algorithm is its speed, this system is designed only for conventional cameras.

2.3 Looming Object Detection

According to the theory of human visual system, a compelling illusion of motion in depth is given by a retinal image that changes in size [25]. Examples are shown in Figure 2.5 and Figure 2.6. When the object increases in size, this phenomenon can be defined as “looming.” Looming is referred to only those events where the object is coming towards or approaching the viewer, or vice versa. After thousands of years of evolution, very sophisticated systems are used in nature. For example, the locusts visual system has a neural structure which selectively responds to looming objects [34].

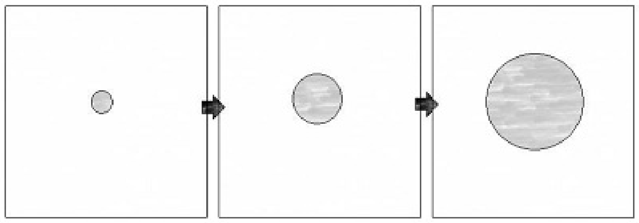


Figure 2.5: A simulated example of looming.

Regan and Beverly [22] presented a psychological evidence that there are some information-processing channels in the human visual pathway which deals with visual looming detection. In this case, they have defined looming as a "change in the image size."

Various algorithms have been developed for detecting looming [8, 20]. Subbarao [28] proposed a method to calculate the time-to-crash of an object and a viewer by using the first-order spatial derivatives of image motion. Park and Sowmya [21] followed three steps to analyze and understand looming behavior. According to them, looming is an optical phenomenon which defines rapid expansion of any object in an image, as it is getting larger on the perceivers retina. For the understanding of looming, first a real time optical flow algorithm is designed. Then, the retinal pixels in 2D space are clustered in order to segment objects. Finally, the looming aspects of the objects are computed. To compute the optical flow in the first step, each pixel has to be matched against other pixels in its neighbourhood

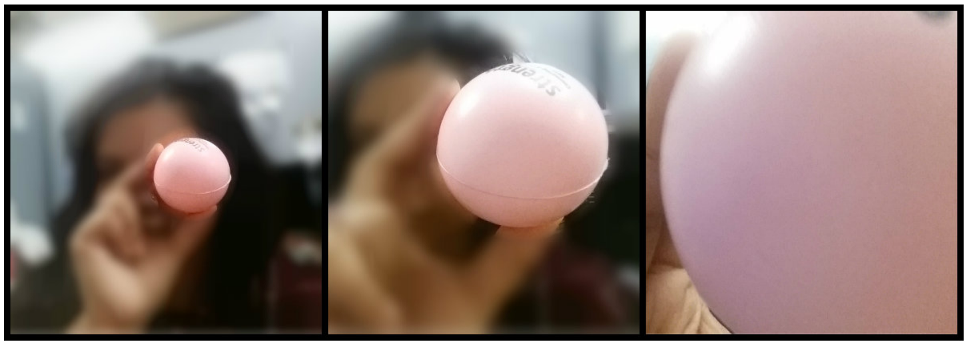


Figure 2.6: A ball approaching towards the viewer.

using correlation computations. Instead of computing the correlation of each pixel to all the pixels in the neighbourhood, Park and Sowmya proposed to use "photo-signatures" as an efficient way to eliminate irrelevant pairs of pixels that need to be matched. These photo-signatures are based on the gradient of the video in spatial and temporal directions. For the segmentation, Self-Organising Feature Map was used to cluster optical flow vectors. For the last step which is the computation of the looming aspects, the focus of expansion is computed for objects. In such a system, a robot will observe any possible obstacle or looming objects by turning its head around. If some possible collision detected, the robot will find a safer path.

Pedro and others [10] proposed an algorithm for the computation of the focus of expansion in a video sequence. Their main objective is to calculate the vanishing point of multi-frame interest point trajectories in consecutive frames. The method consists of two main parts: extracting the point trajectories simultaneously and projecting them into their vanishing point by means of the cross ratio property. The cross ratio is a geometric invariant for any perspective viewpoint such as translation, rotation, and scaling changes [12]. For the first part, they calculate the intersection points and the motion flow of the object(s). After that, for each point they link the consecutive segments using the optical flow as long as the points are visible to the viewer. Thus the point trajectories are collected. Later, they use the point trajectories to determine the focus of expansion.

2.4 The Davis Dynamic Vision Sensor (DVS)

The human brain is composed of several slow asynchronous neural components [17] which show exquisite performance in terms of cognitive behaviour in an uncontrolled environment. In neuromorphic engineering, the main concern is to design a system which resembles the human brain architecture with neural components such as the retina. In most situations where only a few objects are moving in the scene, conventional imagers such as digital cameras capture many frames with redundant data. This is a waste of computational power and time. It is found that biological retina is one of the most developed sensing devices which asynchronously transmit relevant information [11]. If such sensory systems can be built with circuits, it will reduce redundancy and will increase efficiency in power dissipation and information transfer. The Davis Dynamic Vision Sensor (DVS) is a camera that is modeled upon the human retina [14]. It uses a technology that works like the human retina.

Unlike conventional image sensors where image data are transmitted synchronously from the device, the DVS is an asynchronous device that only transmits events indicating the sign of a significant change when a large enough change in log-luminance occurs in any pixel. The output from the DVS camera is an event stream which is a packet container of pixel information. The polarity of the event, the coordinates of the pixel location and the timestamp of that event is carried by the event stream. There might be other information contained in the event stream such as the motion of the camera, but only the polarity, coordinate and the timestamp are important for this thesis. The DVS detects if there is any significant change in any pixel. If there is no significant change, this system does not produce any output. This particular difference from conventional video cameras makes it extremely useful in robotic navigation and other applications. It provides lower power requirements, lower computational requirements and faster reaction times. Since the DVS only compares log-luminance levels instead of reporting the absolute log-luminance levels, it has a higher dynamic range than conventional cameras.

On the other hand, working with this technology is sometimes difficult because only the polarity of the relative changes in pixel values are reported. Only the outline can be seen in this process, not the whole object. As a result the overall image is very hard to visualize if needed (Figure 2.7). Recently there have been some works done on the reconstruction of the original video from the data generated by the DVS. These methods [1, 15] allow reconstruction of the original video. However, the video obtained is only a rough approximation, requiring more computations and many movements in the scene.

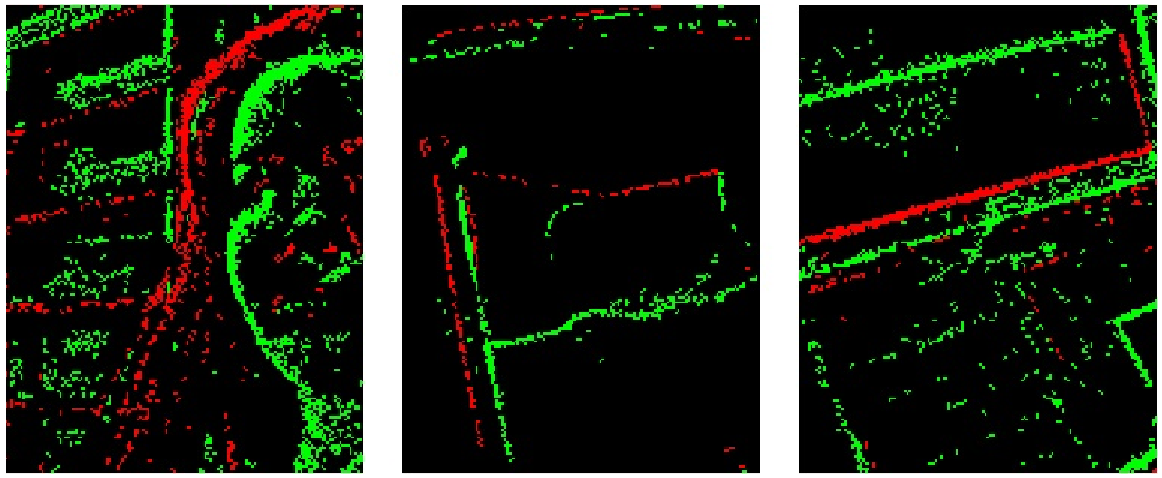


Figure 2.7: Example of DVS output. Here, positive illumination change in a pixel is shown as a green dot and negative illumination change is shown as a red dot.

The events are generated independently by each pixel. The continuous-time photoreceptor output, which encodes luminance logarithmically, is constantly monitored for changes since the last event was emitted by the pixel. A detected change in log-luminance exceeding a threshold value results in the emission of an ON or OFF event. Communication of the event to the periphery resets the pixel, which causes the pixel to memorize the new log-luminance value. The pixels are arranged in an array and fabricated in a standard chip-making process.

2.5 Address Event Representation

The Address Event Representation (AER) is a protocol for the DVS to transmit events generated to the receiver. This technology uses a high speed multiplexor and a narrow communication bus to transport event data by taking advantage of the low frequency of the events and the high speed of the bus [7]. A bus is a type of communication channel in computer architecture that transfers data between components. The pixel output from the DVS consists of asynchronous + and - address-events that signal significant log-luminance changes. Using AER, a structure similar to human brain architecture is provided to the neuromorphic chips to transmit data for a long-range communication [30].

A frame-based process gathers all events in one frame and processes them all at once. In an event-based system whenever an event occurs, it is transmitted and processed. The DVS generates events containing information including the pixel coordinates whose log-luminance has been significantly changed, the sign of the change and the timestamp. For example: (x coordinate, y coordinate, +/-, timestamp) is a typical event packet that is extracted from the event stream in AER. There are two software libraries to access the DVS output in AER format: Java Address-Event Representation (jAER) [32] and C Address-Event Representation (cAER) [31]. jAER is based on Java platform and cAER is based on C. Both libraries are still under development by iniLabs. Though jAER is the standard one, it tends to be slower than cAER because of the platform it uses. Also, cAER uses the low-level libcaer library API [33] which is easy to manipulate as libcaer is intended to give direct low-level access to iniLabs devices. For this reason we have chosen to use cAER in our research.

Chapter 3

Event-based Optical Flow Algorithm

In this section, an event-based optical flow algorithm is developed. The input to the algorithm is an event stream reported by Davis Dynamic Vision Sensor (DVS) in Address Event Representation (AER) format. The output is also an event stream indicating when motion is detected along with its location and direction. Our algorithm is described, and experiments are performed to evaluate its effectiveness in this chapter. We also show that our proposed algorithm can be considered a variation of the Reichardt detector (Section 2.1) on the original scene.

3.1 Motion Detection by Dynamic Vision Sensor

The DVS only reports an event when there is a significant change in the log-luminance at a particular pixel. Objects in motion will cover and uncover background pixels. For example, in Figure 3.1, a dark square object is moving horizontally in a lighter background. As the object moves, the leading edge of the object covers up the background while the trailing edge exposes the background. In this case, the background has an log-luminance higher than the object log-luminance, so the leading edge will trigger changes with negative polarity while the trailing edge will trigger changes with positive polarity.

Generally, when movement occurs in a scene, significant log-luminance changes will occur only near the edges (boundaries) of objects. The background and the object must have significantly different intensities for the motion to be detected. When an object moves in the same direction for a period of time, the pixels along the leading edge will likely

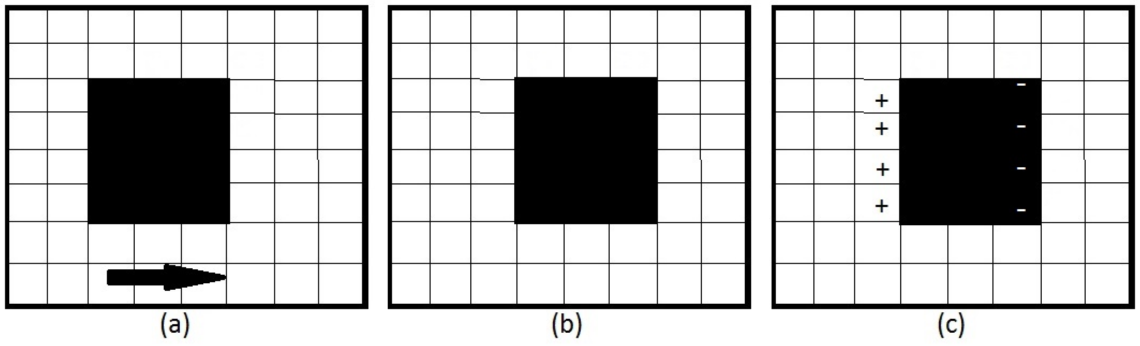


Figure 3.1: DVS events generated from simple horizontal movement. (a) Object at time t = 0. (b) Object after moving one unit of time. (c) Polarity changes.

produce the same polarity of changes along the direction of motion from one instant of time to the next. This is true if we assume that the background surrounding the object is relatively constant, which is a reasonable assumption if the detected changes occur close in time. This is illustrated in Figure 3.2.

3.2 Our Approach

Based on the previous discussion, we propose to detect motion from DVS events by finding events that are close to each other both spatially and temporally. If these events have the same polarity, it may indicate motion. Other factors such as noise may also cause the DVS to report events that will subsequently be detected as motion. By considering only events with the same polarity, as well as changing the thresholds on what should be considered "close" spatially and temporally, the effect of noise can be reduced. A spurious polarity change will not result in motion being detected. Motion will only be detected when there are two matching events detected close together.

Since the DVS generates events in the AER format, our optical flow algorithm will be event-based—the algorithm performs operations for each event received from the DVS, and generates motion events only when motion is detected. Our algorithm will also be online, processing the input as they arrive instead of waiting until enough input for a "frame" to be collected. To maintain the various advantages of the DVS, it is important that the proposed

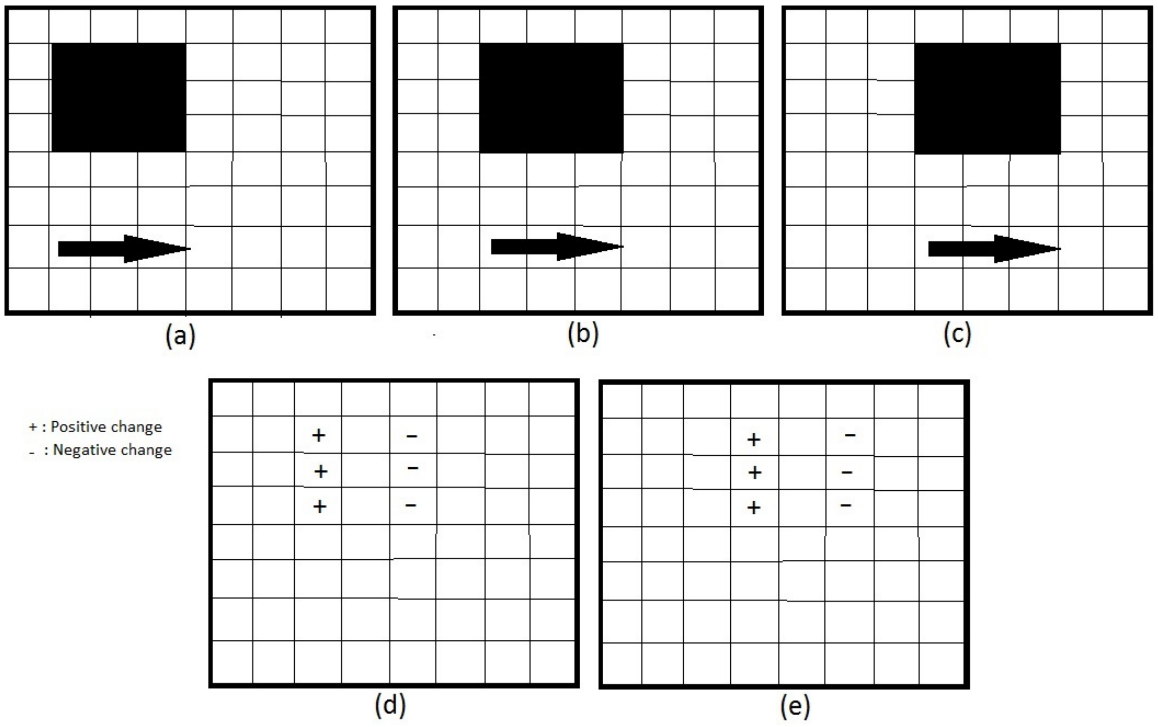


Figure 3.2: DVS events generated from simple movement. (a) A square object at time t = 0. (b) Object is moving, at time t = 1. (c) Object moved two columns, at time t = 2. (d) Showing the polarity change at time t = 1. (e) Showing the polarity change at time t = 2. Note that the edge of the object has significant changes in polarity along the direction of the movement.

algorithm does not convert the event stream into frames to apply conventional frame-based algorithms. If no motion is detected, our algorithm produces no output. This reduces processing and power requirement when there are no significant changes in the scene.

DVS events are generated as pixel changes are detected in time—they are ordered by their timestamps but they may arrive in any order in terms of spatial coordinates. As each event arrives, the algorithm needs to find recent events with the same polarity that are close by. It is necessary to develop a method of storing recent events so that our algorithm can efficiently locate recent events close to a particular location, as well as efficiently update the stored data as events arrive.

3.3 Data Structure

The choice of data structure to maintain the received events can have a big impact on the efficiency of the algorithm. The data structure used to maintain the received events needs to support the following operations:

Insert: insert a new event received with a given timestamp t, location (x,y) and polarity $\pm$.

Find: find recent events located close to a target event at (x,y) with a specific polarity.

Expire: remove events that are no longer recent.

If each event is considered as a 4-tuple $(x,y,t,\pm)$, we may consider this problem as a version of the dynamic nearest neighbour problem. Nearest neighbour algorithms search around a specific point, and find the closest neighbour points which meet some criteria. Along with the retrieval of possible nearest neighbours, this algorithm allows to insert and delete new points [3, 16]. For example, the find operation can find the nearest neighbour event of the target and report if it is close enough. Many data structures have been proposed in the literature for the dynamic nearest neighbour problem, but generally solving this problem is computationally challenging [16].

In our application, we have a restricted range of possible values in spatial coordinates as they are limited by the resolution of the camera. This restriction allows us to simplify our data structure in the same way sorting can be done more efficiently by bucket sort when the range of values is restricted [4]. In our case, a deque[1] can be used at each pixel location. Each deque maintains the recent events corresponding to the location. Events are added to the corresponding queues when they arrive, and are removed when they are too old and expire. Searching for an event that is spatially close can be done by examining only the queues corresponding to the locations close to the target location. Recent events at these locations will be at the back of the corresponding queues.

The idea of using separate queues at each location can be extended even further as there are only two possible values for polarity. As a result, we may use two queues at each location—one for positive polarity and one for negative polarity. To search for a recent event close to target, we simply have to examine the queues close to the targets with the correct polarity. However, we do not use this extension in our algorithm because experimental results show that it is not necessary to obtain good performance.

3.4 Algorithm

The input to our algorithm is an event stream in the AER format. Each event contains the timestamp, the coordinates of the pixel and the polarity. We are also using a timestamp threshold T and a low timestamp threshold T_{low} to specify how recent we want the events to be in order to be considered as a match.

The output of our algorithm is also an event stream, which contains information about detected motion. An event is generated only when motion is detected and no output is generated when no motion is detected. When motion is detected at a particular pixel, the algorithm generates an event specifying the timestamp, the location (x, y), as well as the direction of the motion. In our algorithm, only the eight compass directions are detected

[1]We are using "deque" to mean double-ended queue.

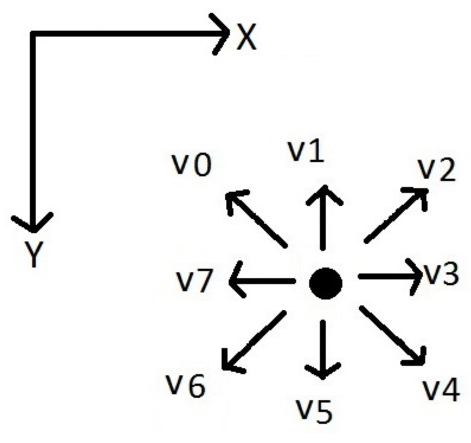


Figure 3.3: Vectors in eight compass directions.

(Figure 3.3), the directions are denoted by the vectors $\vec{v}_0, = (-1,-1), \vec{v}_1 = (0,-1), \vec{v}_2 = (1,-1), \vec{v}_3 = (1,0), \vec{v}_4 = (1,1), \vec{v}_5 = (0,1), \vec{v}_6 = (-1,1), \vec{v}_7 = (-1,0)$. Thus, the output of our algorithm is an event stream indicating the nonzero vectors in the optical flow at specific times and location. The direction vectors are fixed and only these 8 directions can be reported, but it is possible that multiple directions are reported at the same location and time. If desired, the multiple directions at a location can be combined (e.g. by taking the average of the detected directions).

For each pixel location (x,y), we maintain a deque $Q_{(x,y)}$ containing recent events received for that particular location. As each event arrives, we examine the eight neighbourhood pixels and search for a recent event that has the same polarity in the deque. If a match is found, an event indicating detected motion from the neighbour to the current pixel is reported. This is described in Algorithm 1. The parameter T is used to control how "recent" a neighbouring event is to be considered a match to the current event received. And the parameter T_{low} is used to make sure that the events which occurred very recently or at the same time, are not considered as a match.

When a new event occurs, we examine the elements of the deque of neighbouring pixels and check whether there is any event in the deque with the matching polarity. If there is a match, motion is detected. Otherwise, no motion is reported. In the output stream, we have

Algorithm 1: Optical Flow Computation with DVS for a single event received.

```
1  Input: an event from the DVS consisting of timestamp $t$, location $(x,y)$, and polarity
          $p \in \{+,-\}$; a timestamp threshold $T$ and a low timestamp threshold $T_{low}$.
2  Output: if motion is detected, the event $e = (x,y,t,\vec{v}_i)$ is reported.
3  while $Q_{(x,y)}$ is not empty do
4      $(t',p') \leftarrow$ event at the front of $Q_{(x,y)}$;
5      if $t - t' > T$ then
6          remove $(t',p')$ from the front of $Q_{(x,y)}$;
7      else
8          exit the loop ;
   end
9  Add $(t,p)$ to the back of the deque $Q_{(x,y)}$;
10 for $i \leftarrow 0$ to 7 do
11     $(x',y') = (x,y) - \vec{v}_i$ ;
12     found $\leftarrow$ false ;
13     $j \leftarrow 1$ ;
14     while $j \leq$ size($Q_{(x',y')}$) AND NOT found do
15         $(t',p') \leftarrow$ $j$th most recent event in $Q_{(x',y')}$ ;
16         if $T_{low} < t - t' \leq T$ and $p = p'$ then
17             Output event $(x,y,t,\vec{v}_i)$;
18             found $\leftarrow$ true ;
19         $j \leftarrow j+1$ ;
       end
   end
```

the coordinates of the matching pixel, the timestamp and the polarity of the event.

In Algorithm 1, expired events are removed from $Q_{(x,y)}$ at line 3 to line 8. The search for a match is done in the second while loop which begins at line 14 and ends at line 19.

In terms of computational complexity, each input event requires a number of operations proportional to the number of directions (fixed at 8) and the deque size. If a fixed limit on the deque size is imposed, then the number of operations for each event is constant. The number of pixels in the image is irrelevant. This is important because the output of the DVS (the input of our algorithm) is generally sparse and the complexity of our algorithm is directly proportional to the number of events in the input. Thus, the advantage of the DVS is preserved by our algorithm.

Our method is quite similar to the work of Park and Sowmya [21]. They have used a gradient match to determine the neighbourhood, while we are using a polarity match. The events generated from the DVS are considered as derivatives, or changes in log-luminance. Though the concepts are similar, their method is designed only for conventional cameras. Their frame-based algorithm is not applicable to the DVS output.

3.5 Relationship to Reichardt Detectors

Although the main approach in Algorithm 1 can be considered as "event matching," the algorithm is in fact closely related to the Reichardt detector. A single Reichardt detector along the direction $\vec{v} = (\Delta x, \Delta y)$ for a conventional frame-based camera is described by (2.1). Recall that the output of (2.1) is compared to the threshold T_{RD} to determine if motion is detected along the direction $\pm\vec{v}$.

We first show that Algorithm 1 can be considered an application of the Reichardt detector on the output of the DVS (instead of the original scene). An event is generated by the DVS at location (x, y) when the change in log-luminance is greater than T_{DVS}. If we denote this change $\Delta f(x,y,t)$, then

$$|\Delta f(x,y,t)| = |\log f(x,y,t') - \log f(x,y,t)| > T_{DVS}, \tag{3.1}$$

where $t' = t + \Delta t$. When two events are matched in Algorithm 1, each of these events corresponds to a log-luminance change exceeding T_{DVS} at two times t_1 and t_2 with $t_1 < t_2$. In the algorithm, the current event at (x,y) at time t_2 is matched with a previous neighbouring event of the same polarity at time t_1. To simplify notation, we let $t_2 = t_1 + \Delta t_1$, and let $t_3 = t_2 + \Delta t_2$.

A match in polarity of two events at (x,y,t_3) and (x',y',t_2) means that the product $\Delta f(x,y,t_2) \cdot \Delta f(x',y',t_1)$ is positive and greater than $(T_{DVS})^2$, which can be considered as the threshold T_{RD} that is used in (3.11) for detecting motion. The second term of (3.11) may be assumed to be 0 as there are no events at (x',y') at time t_3. Thus, our algorithm can be viewed as applying the Reichardt detector to DVS events.

Next, we describe the relationship between the application of the Reichardt detector to the DVS output and the application of the Reichardt detector to the original scene.

Theorem 3.1.

$$\begin{aligned} RD(\Delta f,x,y,t_1,t_2) = RD(\log f,x,y,t_1,t_2) + RD(\log f,x,y,t_2,t_3) \\ - RD(\log f,x,y,t_1,t_3). \end{aligned} \tag{3.2}$$

Proof. We have

$$\Delta f(x,y,t_1) = \log f(x,y,t_2) - \log f(x,y,t_1), \tag{3.3}$$

and

$$\Delta f(x,y,t_2) = \log f(x,y,t_3) - \log f(x,y,t_2). \tag{3.4}$$

Applying (2.1) from time t_1 to t_3 , we have

$$RD(\Delta f,x,y,t_1,t_2) = \Delta f(x',y',t_1)\cdot\Delta f(x,y,t_2) - \Delta f(x,y,t_1)\cdot\Delta f(x',y',t_2). \tag{3.5}$$

Now, from (3.3), (3.4) and (3.5) we get,

$$\begin{aligned}
RD(\Delta f,x,y,t_1,t_2) &= [\log f(x',y',t_2) - \log f(x',y',t_1)]\cdot[\log f(x,y,t_3) - \log f(x,y,t_2)]\\
&\quad - [\log f(x,y,t_2) - \log f(x,y,t_1)]\cdot[\log f(x',y',t_3) - \log f(x',y',t_2)]\\
&= \log f(x',y',t_2)\cdot\log f(x,y,t_3) - \log f(x',y',t_2)\cdot\log f(x,y,t_2)\\
&\quad - \log f(x',y',t_1)\cdot\log f(x,y,t_3) + \log f(x',y',t_1)\cdot\log f(x,y,t_2)\\
&\quad - \log f(x,y,t_2)\cdot\log f(x',y',t_3) + \log f(x,y,t_2)\cdot\log f(x',y',t_2)\\
&\quad + \log f(x,y,t_1)\cdot\log f(x',y',t_3) - \log f(x,y,t_1)\cdot\log f(x',y',t_2)\\
&= \log f(x',y',t_2)\cdot\log f(x,y,t_3) - \log f(x',y',t_1)\cdot\log f(x,y,t_3)\\
&\quad + \log f(x',y',t_1)\cdot\log f(x,y,t_2) - \log f(x,y,t_2)\cdot\log f(x',y',t_3)\\
&\quad + \log f(x,y,t_1)\cdot\log f(x',y',t_3) - \log f(x,y,t_1)\cdot\log f(x',y',t_2).
\end{aligned} \tag{3.6}$$

From (2.1), (3.3) and (3.4) we get,

$$RD(\log f,x,y,t_2,t_3) = \Delta f(x',y',t_2)\cdot\Delta f(x,y,t_3) - \Delta f(x,y,t_2)\cdot\Delta f(x',y',t_3) \tag{3.7}$$

$$RD(\log f,x,y,t_1,t_3) = \Delta f(x',y',t_1)\cdot\Delta f(x,y,t_3) - \Delta f(x,y,t_1)\cdot\Delta f(x',y',t_3) \tag{3.8}$$

$$RD(\log f,x,y,t_1,t_2) = \Delta f(x',y',t_1)\cdot\Delta f(x,y,t_2) - \Delta f(x,y,t_1)\cdot\Delta f(x',y',t_2) \tag{3.9}$$

Therefore, by combining (3.6), (3.7), (3.8), (3.9) we get

$$\begin{aligned} RD(\Delta f,x,y,t_1,t_2) &= \Delta f(x',y',t_1)\cdot\Delta f(x,y,t_2)-\Delta f(x,y,t_1)\cdot\Delta f(x',y',t_2) \\ &= RD(\log f,x,y,t_1,t_2)+RD(\log f,x,y,t_2,t_3) \\ &\quad -RD(\log f,x,y,t_1,t_3). \end{aligned} \tag{3.10}$$

□

Applying (2.1) to the output of the DVS, we have

$$RD(\Delta f,x,y,t,t') = \Delta f(x',y',t)\cdot\Delta f(x,y,t')-\Delta f(x,y,t)\cdot\Delta f(x',y',t'). \tag{3.11}$$

Theorem 3.1 shows that, the application of Reichardt detector to the DVS output can be thought of as a combination of the outputs of different Reichardt detectors on the original scene at closely related times. Thus, we have shown that Algorithm 1 can be considered to be a variation of the Reichardt detector on the original scene.

3.6 Experiments

In this section, some experiments performed with our method are discussed. The system specification and the thresholds used for the experiments are also presented.

3.6.1 Thresholds and Parameters

For the experiments, the properties of all constants and thresholds we have used are shown in Table 3.1. The thresholds were chosen experimentally to achieve good results. Also, the system specifications used for the experiments are described in Table 3.2 and the camera specifications are described in Table 3.3. For illustration, the arrows with green line and red head show the direction of movement (Figure 3.4). We illustrate the optical flow

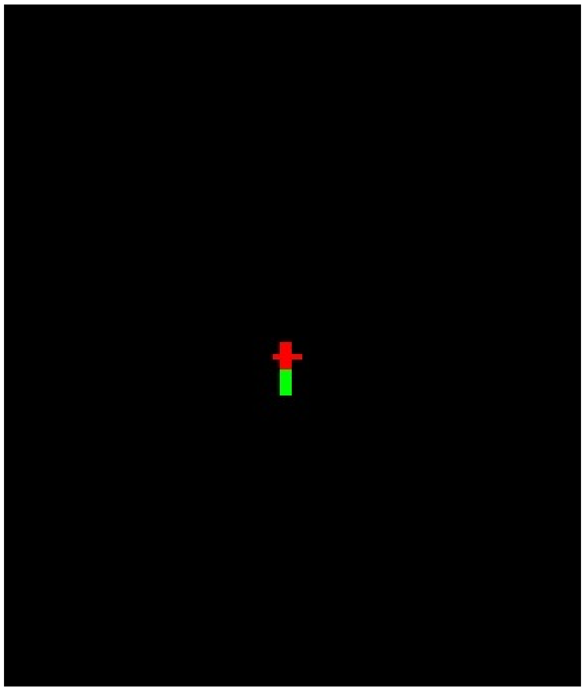


Figure 3.4: An arrow with green line and red head used to visualize the direction of detected movement.

Table 3.1: Properties of all constants and thresholds for all experiments.

Name	Value	Unit
COLS	180	pixels
ROWS	190	pixels
Timestamp Threshold (T)	25,000	microseconds
Low Timestamp Threshold (T_{low})	100	microseconds

detected at particular times for this thesis, but the optical flow is computed continuously in real-time.

3.6.2 Experiments on simple objects

Simple objects have been used with DVS camera for this part of experiments.

Round looming object In this experiment, a round object is moving towards the viewer. We can see in Figure 3.5 that most of the arrows are pointing away from the center of the object. The number of arrows pointing outside is larger than the number of arrows pointing inside, which indicated that the object is coming towards the viewer.

Table 3.2: The system specifications used for experiments.

Name	Value
CPU	Intel(R) Core(TM) i7-3630QM CPU @2.40GHz
RAM	8 GB
Programming Language	C++
Libraries	libopencv, libstdc++, libGL, libcaer

Table 3.3: The Davis Dynamic Vision Sensor camera specifications [14] used for experiments.

Name	Value
Model	DVS128
Optics	CS-mount
I/O	USB2.0
Software	cAER
Power source	USB Type B
Power consumption	Low/high activity: 30/60 mA @ 5 VDC

Here the total length of the video is 3.92 seconds, and the total execution time of Algorithm 1 is 1.35739 seconds over all DVS events. The total number of events is 668530. That means it takes 2.03 microseconds to process each event on average. The results are shown in Table 3.4.

Round object moving sideways In Figure 3.6 a round object is moved from right to left. The number of arrows pointing to the left of the frame is significantly higher than the number of arrows pointing to other directions. The results are shown in Table 3.5.

Table 3.4: Results of the experiment with round looming object.

Name	Value	Unit
Total number of events	668530	-
Runtime	1.35739	seconds
Length of video	3.92	seconds
Runtime per event	2.03	microseconds

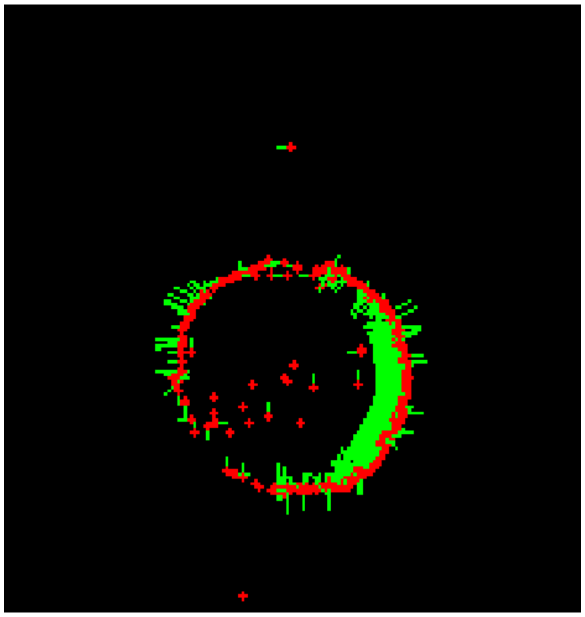


Figure 3.5: A round looming object.

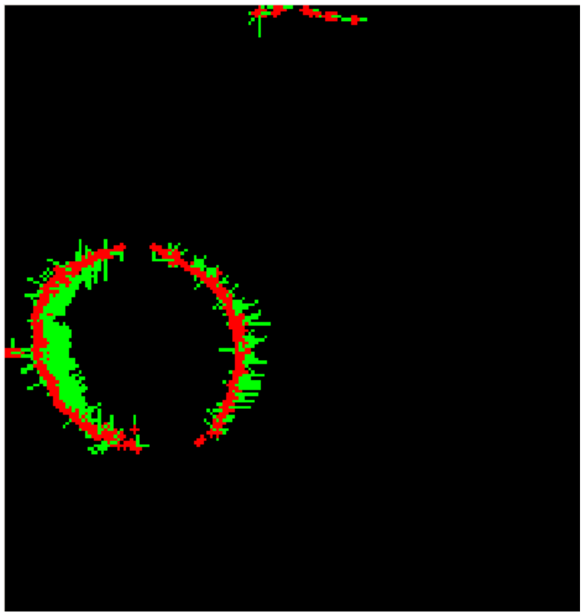


Figure 3.6: A round object moving sideways.

Table 3.5: Results of the experiment with round object moving sideways.

Name	Value	Unit
Total number of events	672201	-
Runtime	1.41999	seconds
Length of video	2.81	seconds
Runtime per event	2.11	microseconds

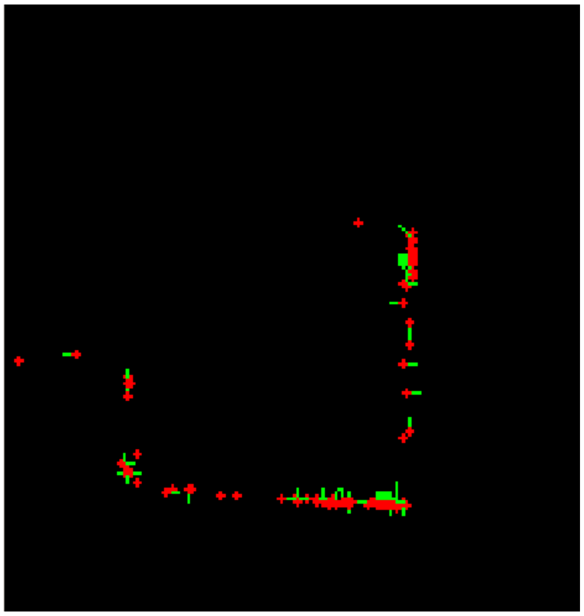


Figure 3.7: A square object with little movement.

Table 3.6: Results of the experiment with square object with little movement.

Name	Value	Unit
Total number of events	655203	-
Runtime	1.32696	seconds
Length of video	5.90	seconds
Runtime per event	2.02	microseconds

Square object with little movement In this experiment, the square object did not have a lot of movement. This explains the significantly low number of arrows in Figure 3.7. The results are shown in Table 3.6. We can see that the length of the video is 5.90 seconds but the number of events generated by the DVS is only 655203.

Square object moving from top to bottom The result of the experiment in Figure 3.8 is visually understandable. This square object had a movement from top to bottom of the frame. The results are shown in Table 3.7.

Square object moving sideways From the result of the experiment in Figure 3.9, we can say that the square object is moving from left to right by the direction of most of the arrows. The results are shown in Table 3.8.

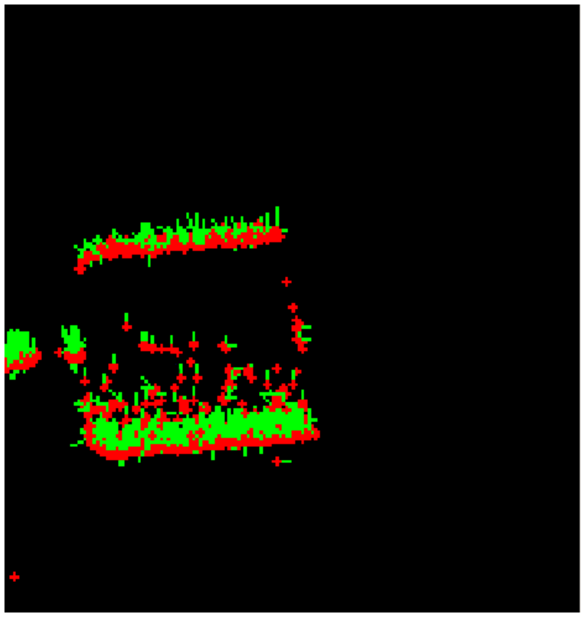


Figure 3.8: A square object moving from top to bottom of the scene.

Table 3.7: Results of the experiment with square object moving from top to bottom.

Name	Value	Unit
Total number of events	765584	-
Runtime	1.67583	seconds
Length of video	4.45	seconds
Runtime per event	2.19	microseconds

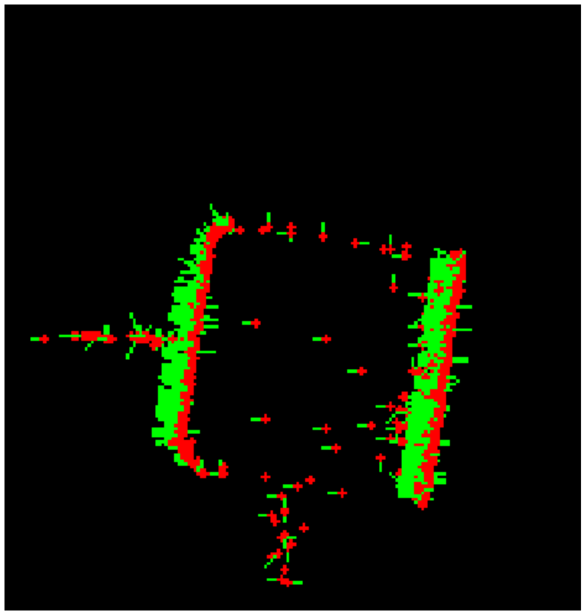


Figure 3.9: A square object moving from left to right.

Table 3.8: Results of the experiment square object moving sideways.

Name	Value	Unit
Total number of events	792649	-
Runtime	1.59001	seconds
Length of video	10.07	seconds
Runtime per event	2.00	microseconds

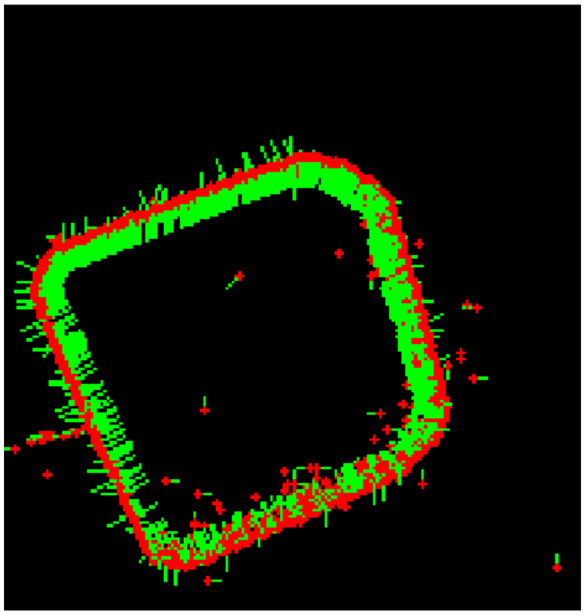


Figure 3.10: A looming square object.

Looming square object This experiment has a clear visualization of looming. The square object is coming towards the viewer, so the number of arrows pointing towards outside of the object is very large (Figure 3.10). The results are shown in Table 3.9.

Round object with looming In Figure 3.11, a round object is coming towards the viewer. We can observe that a large number of arrows are pointing outside of the edge of the object.

Table 3.9: Results of the experiment with looming square object.

Name	Value	Unit
Total number of events	850175	-
Runtime	1.60829	seconds
Length of video	5.10	seconds
Runtime per event	1.89	microseconds

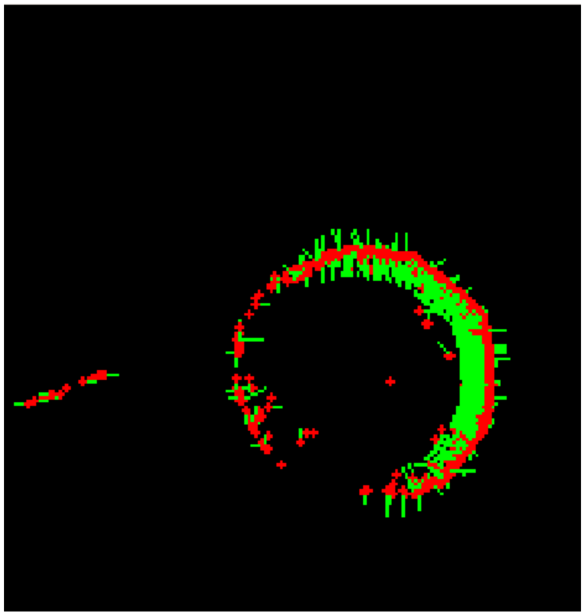


Figure 3.11: A looming round object.

Table 3.10: Results of the experiment with round object with looming.

Name	Value	Unit
Total number of events	657771	-
Runtime	1.25298	seconds
Length of video	4.88	seconds
Runtime per event	1.90	microseconds

The results are shown in Table 3.10.

Summary As shown by these experiments, Algorithm 1 produces results that agree with the input scene accurately. The total execution time depends only on the number of DVS events processed and not the length of the videos. On average, the time to process each event ranges from 1.89 microseconds to 2.19 microseconds. Notice that for a conventional camera with a resolution of 180×190, at 24 frames per second (fps) there will be 820800 pixels transmitted in one second of video. Our experimental results show the drastic reduction of data transmitted when the DVS is used.

Table 3.11: Properties of the test videos for simulated DVS.

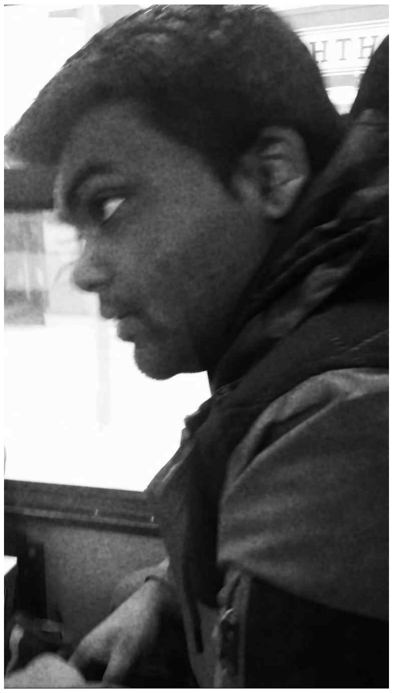
	Size	Number of Frames	Number of DVS events
Video 1	1280×720	60	32,940
Video 2	1920×1080	45	8,953,097

3.6.3 Experiments on simulated DVS

For the next experiments, we have designed a simulated DVS which uses captured videos from conventional frame-based cameras. Our simulated DVS compares intensity values from consecutive frames, and generate events when a significant change is detected. A simulated DVS was used because conventional frame-based videos were more available when the algorithm was first designed and implemented.

We have implemented and tested our optical flow algorithm on various kinds of videos with real objects such as human, vehicles etc. using our simulated DVS. For this experiment, two test videos are used on Algorithm 1. The properties of the videos are shown in Table 3.11. The first video consists of a single circular object that moves around in a dark background while the object is brighter. The second video contains a person moving his head and body. The camera is not steady so both the foreground and the background of the scene are moving. In the first video, events are generated by the DVS only on the boundary of the circular object, resulting in a significant reduction in the amount of data sent compared to conventional frame-based cameras. This is true even in the second video—the number of events generated is less than a tenth of the total number of pixels among all frames. Two example frames from the second video are shown in Figure 3.12, and the DVS events generated corresponding to these frames are shown in Figure 3.13.

Figures 3.14 and 3.15 show some of the optical flow computed by Algorithm 1 on Video 1 and Video 2, respectively. To visualize the results, motion events generated are collected and those occurring at the same time are displayed as individual images. Red arrows have been used to show the pixel movement. To make it easier to visualize, not all vectors reported by our algorithm are shown, only one vector from a group of closely


(a) Frame 26

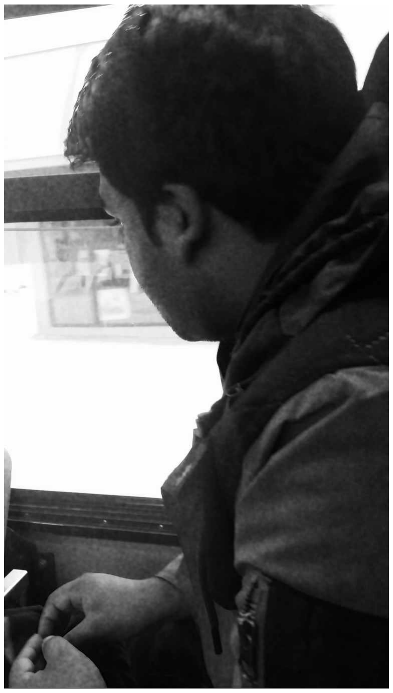
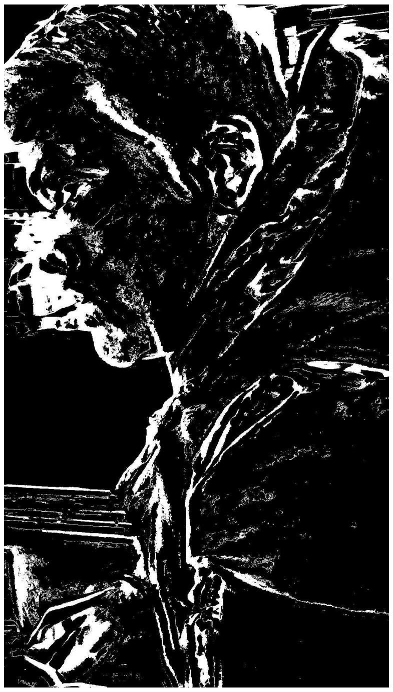

(b) Frame 44

Figure 3.12: Two example frames from the second test video.


(a) Frame 26

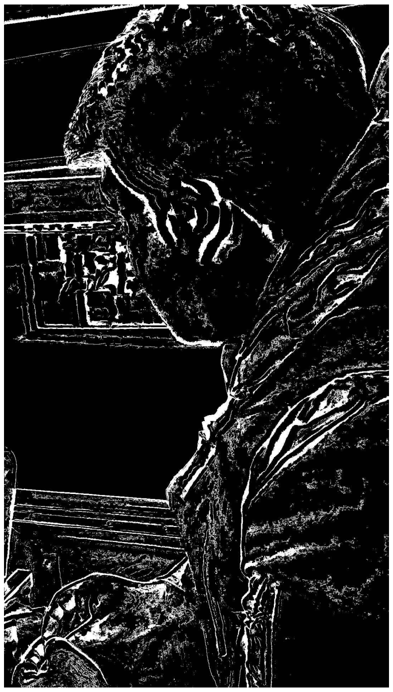

(b) Frame 44

Figure 3.13: DVS events corresponding to the example frames in Figure 3.12. Pixels with DVS events are shown in white.

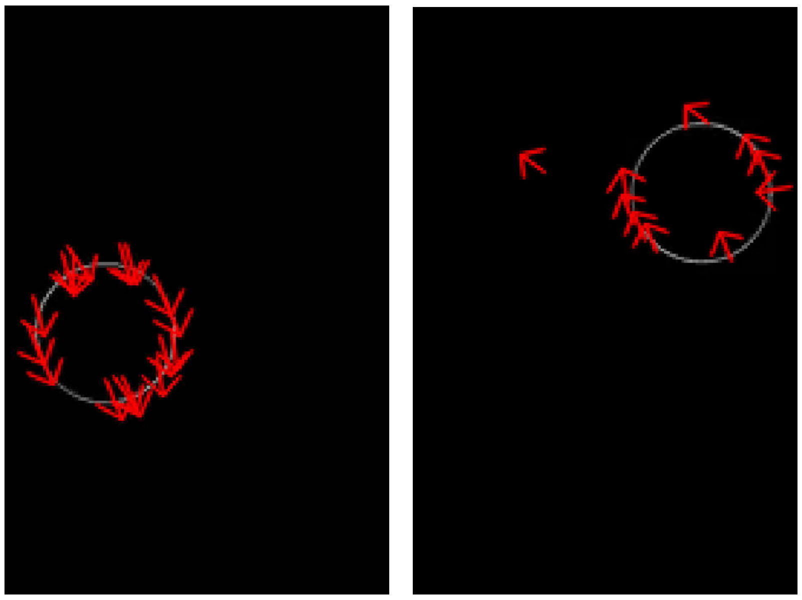

(a) Frame 3 (b) Frame 30

Figure 3.14: A visualization of the output of Algorithm 1 on Video 1.

located vectors are shown. We can visually observe that the optical flow computed reflect the actual motion present in the videos, though there are very rarely extraneous motion detected due to noise (e.g. the motion vector detected in Frame 30 in Figure 3.14).

3.6.4 Summary

In these experiments, we have only performed qualitative evaluation by visually inspecting the output. In order to evaluate accuracy quantitatively, one must have access to the "ground truth" which is time consuming to produce. Also, one must determine a measure to quantify accuracy in this case. Instead, we will evaluate Algorithm 1 in conjunction with the looming detection algorithm in Chapter 4, because the correct output in looming detection is easier to determine.

3.7 Variations

There are different variations of our algorithm that can be considered. We list some variations below and the motivations for these variations. A thorough examination of these

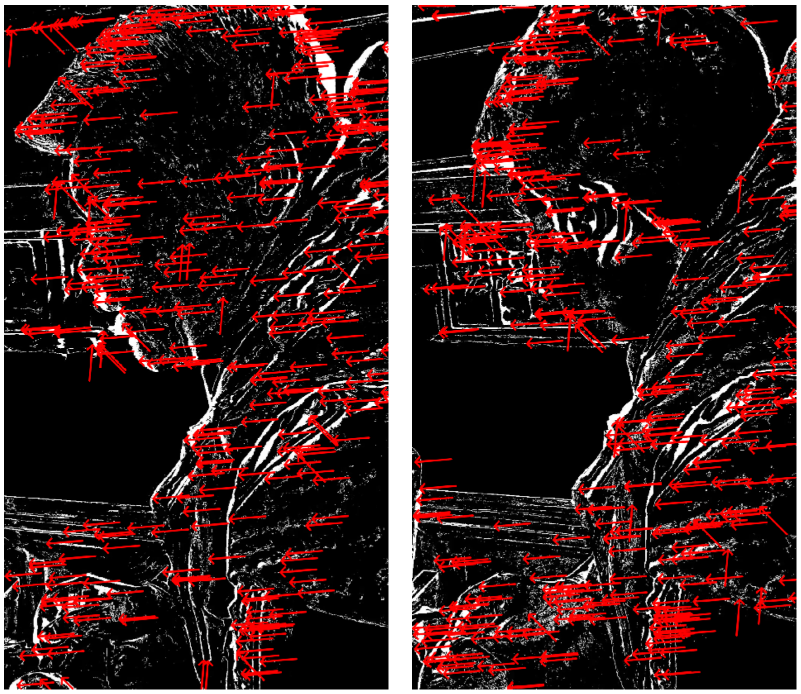

(a) Frame 11 (b) Frame 43

Figure 3.15: A visualization of the output of Algorithm 1 on Video 2.

variations beyond the scope of this thesis can be explored in the future.

3.7.1 Definition of the "most recent event" and the timestamp threshold

Our goal is to process the recent events only. Events that are too old are not our concern. If we match some events that are too old, it means that the matched events can be noise, or not useful. Also, since we are detecting motions that are fast enough, slow moving objects are not our concern right now. We can vary the interpretation of the "most recent" events by changing a timestamp threshold. The timestamp threshold T is a value which is used to determine the maximum time difference allowed between the two pixel events to be considered as a match. To be more specific, when we get an event at a particular pixel and search for another "recent" event near this pixel, we take the time differences between that event and all the other neighbouring events and compare it with the threshold. If the difference is within the threshold, we consider it as a possible match for our event. A very active scene may cause many events to be generated by the DVS, and will increase the size of the deque. The deque may get exceptionally large in case of a scene that has many active objects in it. This leads to very large space requirement. The appropriate timestamp threshold depends on the speed of the objects in a scene.

If the object is moving fast, a large timestamp threshold may cause the deque to grow too large and may cause several matches leading to ambiguity in motion detection. On the other hand, in a scene with a slow moving object, some of the movements may not be detected when a small timestamp threshold is used. Different threshold values are required for different velocities of object to be detected in our system. We can adjust the threshold to detect motion of objects with different velocity.

3.7.2 Size of the data structure

Varying the deque size will allow us to examine different number of events as required. Also, we can examine older events rather than just the recent ones, which may allow us to calculate the velocities of the objects. If our timestamp threshold is too large then our deque might get too large as well. A limit on deque size can be used to control space usage in our system. Also, in case of a very active scene, the deque can grow very large as well. On the other hand, if the timestamp threshold is smaller or the scene is quite idle, the deque might be very small which reduces space usage for every pixel.

3.7.3 Reporting the direction of detected motion

There can be multiple matches in a pixel neighbourhood. For example, in Figure 3.16 we can see that when the square object moves, there are polarity changes at the edges of the object. The same polarity change occurs in a specific direction alongside the edge of the object. This creates multiple vectors originating from a single pixel. In matching events in Figure 3.16 (b), there are some negative polarities alongside the edge and they are within the same neighbourhood in some cases. In this situation, multiple vectors are detected from a single center pixel. We can report all vectors separately, or simply compute and report the average of the vectors detected. This choice will depend on the use of the output. If we

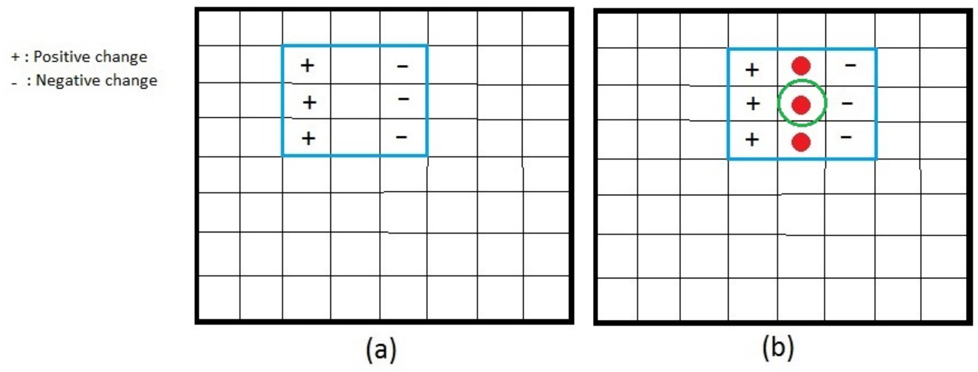


Figure 3.16: (a) Showing the polarity change of a moving square object at time t = 0. (b) Showing the polarity change at time t = 1. Note that the edge of the object has significant changes in polarity along the direction of the movement. The circled pixel will be matched with any of the three pixels with – polarity. This will create multiple vectors originating from a single pixel.

report the vectors separately, it will allow us to discretize the directions. On the other hand, taking an average of the vectors is more convenient in case of visualization. There are also various optical flow algorithms which report a single detection, and averaging the detected vectors will produce comparable results. Also, taking an average may reduce the effect of noise in the output.

Chapter 4

Looming Object Detection

After obtaining the optical flow from Algorithm 1, our next task is to determine if the detected motion represents looming objects. There are various ways to identify looming objects from movement. In this section, our algorithm and the implementation of looming object detection are discussed. Experimental results are also shown.

4.1 Our Approach

The output of Algorithm 1 is the event stream with detected motion. For looming detection, Algorithm 1 reports the average of directions of the movement (3.7.3) detected at a pixel. The events can represent zero or more objects. We have to identify if there is any object in the scene or not. After we find an object, we need to determine the type of movement of the object.

4.1.1 Object Detection

In order to classify the movement of the object(s) in the scene, we need to identify the boundary or the edge of the object(s) first. An efficient edge-linking method is required. Our approach for this thesis is to examine the direction of the closely located pixels. From Algorithm 1 we obtain the properties of the events describing the detected movement. The information we get from the events are: the location and the direction of movement of the particular pixel.

After we get an event as the output from Algorithm 1, we conduct a search in its neighbourhood for another event with a similar direction. If the search is successful, we link the

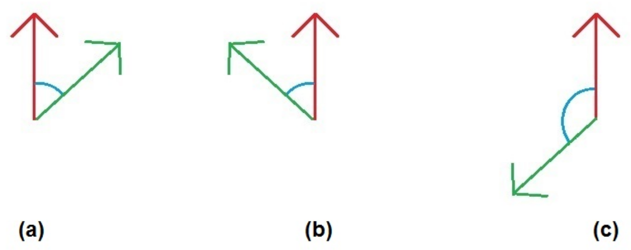


Figure 4.1: (a) and (b) The angle between the arrows is 45 degree which can be considered as similar direction (c) Angle between the arrows is more than 45 and they are not considered as similar direction.

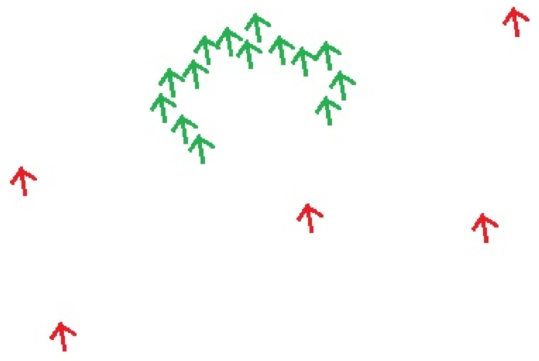


Figure 4.2: Linking the pixels.

origin pixel and the neighbour pixel together. These linked pixels are considered as a part of an edge of an object. We define two directions as “similar” if the angle between them is 45 degree or less (Figure 4.1).

By this method, we are grouping all similar events located close to each other and ignoring all other events which are located sporadically. These can be considered as isolated events which possibly are noise. Only large enough groups are considered as edges. For example, in Figure 4.2 the green arrows are placed closely with each other and the red arrows are scattered throughout the frame. The green arrows can be linked together and considered as an edge of a possible object. Red arrows can be ignored as noise.

The final output of this method may not form a complete edge; some parts of the original

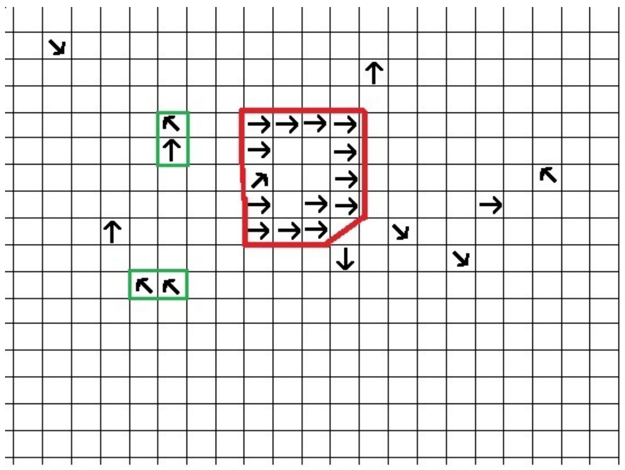


Figure 4.3: Group of pixels forming a square region (red area). The areas in green boundary are too isolated to be considered.

object may be ignored by this method. There can be two reasons for that. Either the changes were not significant enough (below the threshold) to be detected by the DVS, or the events were too isolated to be considered by our method. But the edges we obtain from this edge-linking method mostly consist of large segments of the edge of the object. These segments contain useful information for motion analysis. This linked edge can be a group of pixels having a particular shape such as a circle, polygon, ellipse or any random shape. For example, in Figure 4.3 the arrows with similar direction form a square object, though some part of the object is missing.

In our approach, we are not linking all the events with same direction. We are actually trying to form an edge, not a region. We are only concerned about the edge of the object because for looming detection, observing only the pixels on the boundary of the object is required. An event is considered to be on the boundary if at least one of its eight neighbourhood does not have a recent event. Otherwise it is considered to be in the interior. For this we check if an event is on the boundary of the possible object or not. If so, we link it with other selected events (Figure 4.4). By ignoring the events in the interior of the object, we are reducing the amount of data that need to be processed. Also, we are reducing possible ambiguity which was explained in Figure 3.16. Taking only the edge pixels into consideration will reduce the problem of reporting multiple vectors unnecessarily.

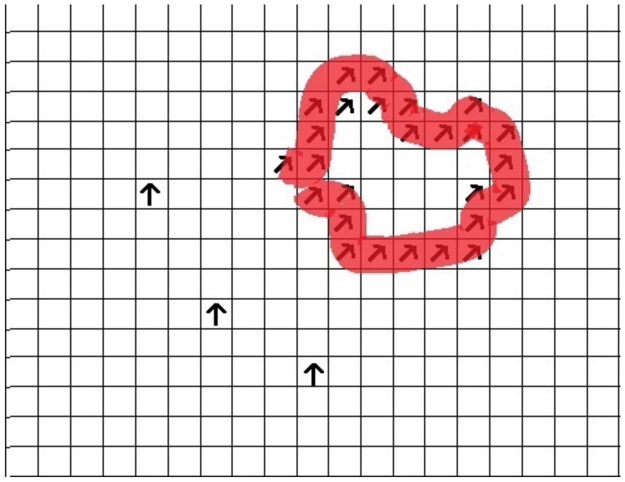


Figure 4.4: Linking pixels to form an edge.

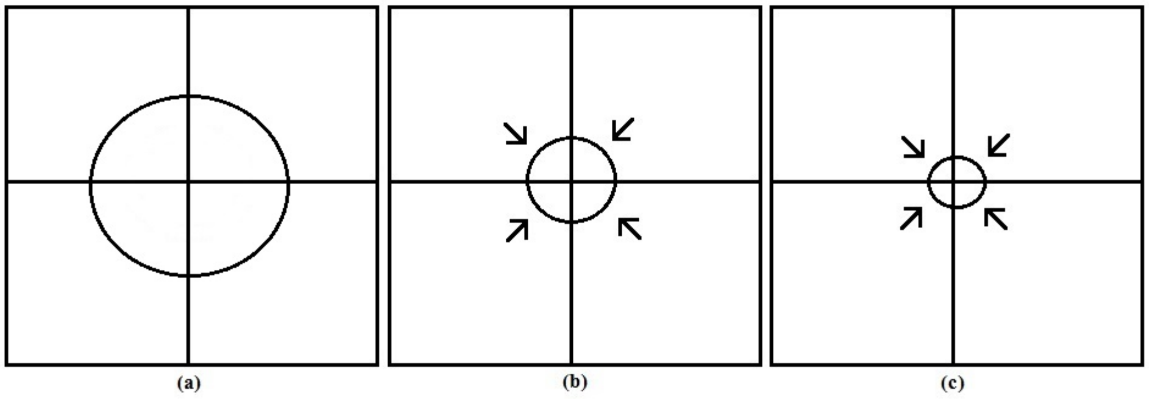


Figure 4.5: (a) A round object at time t = 0. (b) Object is moving away, at time t = 1. (c) Object is moving away, at time t = 2.

4.1.2 Classifying the type of Movement

Objects can move in various ways. We can classify object movement into three types: moving away from the viewer, moving towards the viewer (looming) and moving sideways.

When an object moves away from the viewer, the object appears to be getting smaller. The edge of the object moves towards the center as we can see in Figure 4.5. In case of the object moving towards the viewer, or looming, we can see the objects getting bigger. The edge of the object moves away from the center (Figure 4.6). When an object moves sideways in the frame (Figure 4.7), the object size remains the same. The leading edge of the object moves away from the center and the trailing edge moves towards the center.

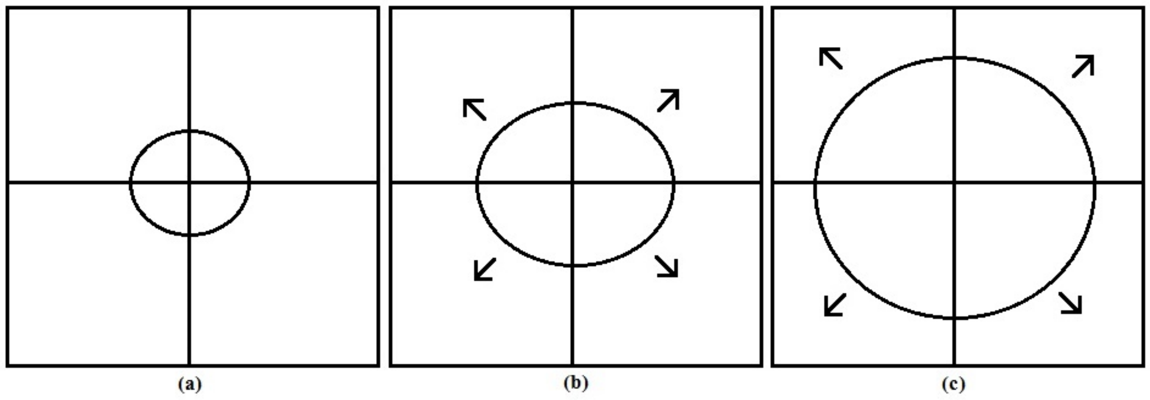


Figure 4.6: (a) A round object at time t = 0. (b) Object is moving towards, at time t = 1. (c) Object is moving towards, at time t = 2.

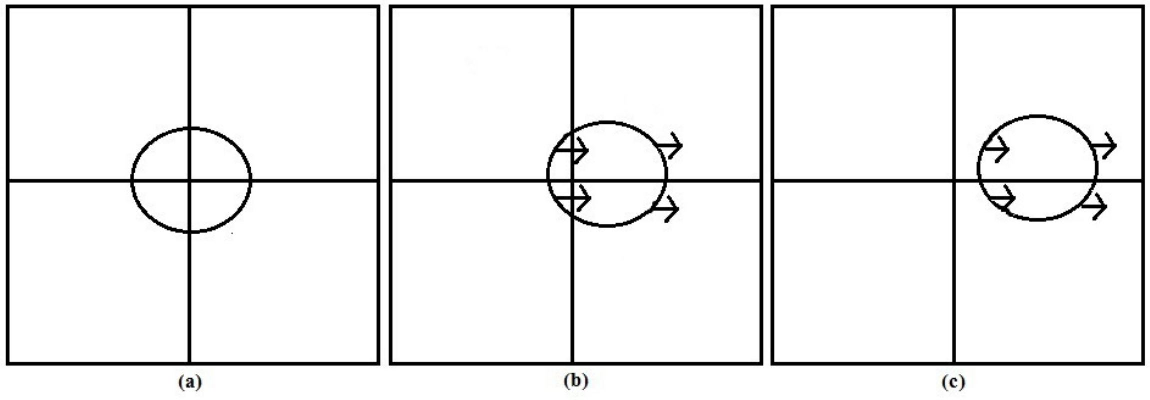


Figure 4.7: (a) A round object at time t = 0. (b) Object is moving horizontally, at time t = 1. (c) Object is moving sideways, at time t = 2.

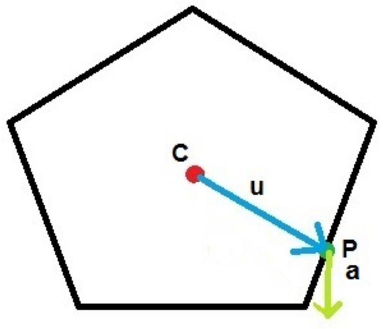


Figure 4.8: $\vec{u}$ is the vector from interior point C to pixel P. $\vec{a}$ is the direction vector at pixel P.

In order to classify the movement of the object, we need to find an interior point of the object and analyze all boundary events to determine if they are moving away from the interior. We have chosen to use the arithmetic mean of all boundary pixels in the group to obtain an interior point. This point is in the interior of the region provided that the region has a convex shape.

After we obtained an interior point of the object, we define $\vec{v}$ to be the vector associated with the direction in the reported motion event, and $\vec{u}$ to be the vector from the interior point to the event on the boundary. We need to determine if $\vec{v}$ is pointing away from the interior point. In other words, we need to determine if the angle between $\vec{u}$ and $\vec{v}$ is less than 90 degrees.

We use a dot product calculation to find out whether the event is pointing towards or pointing away from the interior. The dot product can be described as the projection of one vector onto another. For example, suppose we have two vectors $\vec{a}$ and $\vec{u}$ (Figure 4.8), and we want to calculate if vector $\vec{a}$ is pointing in the same direction as the vector $\vec{u}$. The dot product of these vectors will be positive if the two vectors are pointing in generally the same directions and negative if the two vectors are pointing in opposite directions. The value will be zero if they are perpendicular. Therefore, we compute $\vec{u} \cdot \vec{v}$, and conclude that $\vec{v}$ is pointing away from the interior if $\vec{u} \cdot \vec{v} > 0$.

After we reach a certain amount of linked events in a group, we examine the number of vectors pointing towards the interior and pointing away from the interior. If the number of vectors pointing away from the interior is greater than twice the number of vectors pointing towards the interior, our algorithm reports that a looming object is detected in the scene and resets the group. The ratio of two between the two types of vectors has been chosen experimentally. In addition, looming should only be reported if there is a significant number of vectors pointing away from the interior. Otherwise, the vectors may be noise and do not represent edges of an object. Experimentally, we have chosen to report looming only if the number of vectors pointing away from the interior is at least 0.5% of the total number of pixels in the scene.

Suppose we have $N_{\vec{pos}}$ as the number of vectors pointing away from the interior and $N_{\vec{neg}}$ as the number of vectors pointing towards the interior. Then, a looming event is reported if $N_{\vec{pos}} > 2 \times N_{\vec{neg}}$ and $N_{\vec{pos}} \geq 0.005 \times ROWS \times COLS$. Otherwise, no looming event is reported.

4.2 Data Structure

To store the events we have used a deque for each pixel location in a similar way as in Algorithm 1. The number of possible events required to store for each pixel is restricted to a constant in our algorithm. Thus the operations such as finding the most recent event is constant.

In Algorithm 2, we have to store and link the events in order to form an edge. The efficiency of our algorithm immensely depends on the data structure we use to maintain connectivity information. We have to choose a data structure which will reduce the time of execution and the memory requirement. A disjoint-set data structure maintains a collection or a set of data [9]. This data structure is also called union-find data structure. In our algorithm, we are using the union-find data structure to link (union) the pixels in order to identify a possible edge of an object.

An union-find data structure performs two useful operations: Union and Find. Union joins two subsets into a single subset. Find typically returns an item from this set that serves as its “representative”; by comparing the results of two Find operations, one can determine whether two elements are in the same subset. This data structure can perform both of these operations efficiently, and the complexity of each operation is practically constant time independent of the number of elements maintained[2].

4.3 Algorithm

The input of our Looming Detection algorithm (Algorithm 2) is the output events from our Optical Flow algorithm (Algorithm 1). The features in the output events are: the x and y coordinate of the event, the timestamp t of the event and the direction vector $\vec{v}$ of the event. We are using a set of thresholds in this algorithm. The timestamp threshold T and the low timestamp threshold T_{low} are the same as the previous algorithm. We are introducing two new thresholds in this algorithm which are a threshold S to define the size of each group of pixels and a threshold I to define the length of intervals we are using to stop collecting data and start the analysis of the object movement.

In Algorithm 2, our first task is to remove the old events that are already expired (line 1 to line 6). To do so, we are using a while loop to calculate the time difference of current time and recent time. If the difference is less than the timestamp threshold T, we remove the event from the deque. After we are done, we have recent events only to consider in the deque. Now in line 7 we add the event to the back of the deque. Following that, from line 9 to 13 we are using another algorithm (Algorithm 3) to decide if the current event is a boundary or not. In Algorithm 3, a loop is used to search for matching events in the deques in the neighbourhood.

Starting from line 16 to line 20, we link the events which are on the boundary of the object. Here we use the union-find data structure to link the events. We have a time threshold I to control when linked events are analyzed for looming objects (Algorithm 4).

[2]While this is not entirely accurate, it is sufficient for our application. Please see reference for a more precise complexity analysis.

Algorithm 2: Looming detection algorithm from the events collected from Algorithm 1.

```
Input: An event reported by Algorithm 1: $e = (x, y, t, \vec{v})$.
Thresholds: a timestamp threshold $T$, a low timestamp threshold $T_{low}$, a threshold for
the size of the group of pixels $S$ and a threshold for the length of regular interval to
analyze object detected $I$.
Output: If looming is detected, an event $t$ is reported.
1  while $Q_{(x,y)}$ is not empty do
2      $(x', y', t', \vec{v'}) \leftarrow$ event at the front of $Q_{(x,y)}$;
3      if $t - t' > T$ then
4          remove $(x', y', t', \vec{v'})$ from the front of $Q_{(x,y)}$;
5      else
6          exit the loop ;
   end
7  Add $(t, \vec{v})$ to the back of the deque $Q_{(x,y)}$;
8  boundary $\leftarrow$ false ;
9  for $i \leftarrow 0$ to $7$ do
10     $(x', y') \leftarrow (x, y) - \vec{v}_i$ ;
11     found $\leftarrow$ result of Algorithm 3 with parameters $Q_{(x',y')}$, $T$, $T_{low}$, $t$, $\vec{v}$ ;
12     if NOT found then
13         boundary $\leftarrow$ true ;
   end
14 if NOT boundary then
15     return ;
16 for $i \leftarrow 0$ to $7$ do
17     $(x', y') \leftarrow (x, y) - \vec{v}_i$ ;
18     found $\leftarrow$ result of Algorithm 3 with parameters $Q_{(x',y')}$, $T$, $T_{low}$, $t$, $\vec{v}$ ;
19     if found then
20         dir$_{(x,y)} \leftarrow \vec{v}$ ;
21         dir$_{(x',y')} \leftarrow \vec{v'}$ ;
22         union$((x, y), (x', y'))$ ;
   end
23 $last \leftarrow 0$ ;
24 $elapsed \leftarrow t - last$ ;
25 if $elapsed > I$ then
26     Call Algorithm 4 with current union-find structure ;
27     $last \leftarrow t$ ;
```

Algorithm 3: Find recent event with similar directions.

```
Input: Event queue $Q$, thresholds $T$ and $T_{low}$ as in Algorithm 2, a time $t$, and a
       direction $\vec{v}$.
Output: Returns true if there is a recent event in the queue with a similar direction
        as $\vec{v}$, false otherwise.
1  $j \leftarrow 1$ ;
2  while $j \leq size(Q)$ do
3      $(t', \vec{v'}) \leftarrow$ $j$th most recent event in $Q$ ;
4      if $T_{low} < t - t' \leq T$ AND $\vec{v}$ and $\vec{v'}$ are similar then
5          return true ;
6      $j \leftarrow j + 1$ ;
   end
   return false ;
```

Algorithm 4: Analyzing the group of pixels collected from Algorithm 2.

```
Input: Union-find data structure from Algorithm 2, and a threshold for the size of
       the group of pixels $S$.
Output: If looming is detected, an event $t$ is reported.
1  pos $\leftarrow 0$ ;
2  neg $\leftarrow 0$ ;
3  for each groups of pixels in union-find do
4      $C \leftarrow$ arithmetic mean of the group ;
5      $d \leftarrow ((x, y) - C) \cdot dir_{(x,y)}$ ;
6      if $d > 0$ then
7          pos $\leftarrow pos + 1$ ;
8      if $d < 0$ then
9          neg $\leftarrow neg + 1$ ;
   end
10 if $pos > 2 \times neg$ AND $pos \geq S \times ROWS \times COLS$ then
11     report looming event at time $t$ ;
12 reset union-find data structure for pixels in the group
```

For the analysis in Algorithm 4, we have the union-find data structure from Algorithm 2. First we find an interior point of the object. Then we calculate the dot product between the direction of the event and the interior point. If the result is positive, we say this event is pointing away of the object. If the result is negative, we say the event is not pointing away of the object. Finally when the analysis is done, we reset the union-find data structure in order to perform object analyze in the next time interval.

In terms of computational complexity, we can calculate the overall computational requirement by examining each individual loop in our algorithm. In the first loop in Algorithm 2, expired events are removed. For this part the complexity is proportional on the deque size, which is constant for Algorithm 2. The next part of our algorithm is avoiding the boundary events, which requires a search in the eight-neighbourhood of each event in the deque. The complexity for this process is also proportional to the number of events in the deque, which is constant.

After checking the boundary we are searching for similar events and linking them using the union-find data structure. The complexity of each union and find operation is practically constant. Lastly, we are analyzing the existing events in the union-find data structure by examining each group in the data structure and some calculations. For this part the complexity depends on the size of the group. Although the sizes of these groups are different each time, the sizes are proportional to the number of events processed by the algorithm since the last reset. Therefore, processing these groups can be done in constant time per event processed amortized.

4.4 Experiment and Analysis

We have performed experiments on our looming object detection algorithm (Algorithm 2) on various videos. Here we discuss the results from the experiments. We have used the same DVS event streams on Algorithm 2 as the ones we used to test Algorithm 1 in Section 3.6.

Table 4.1: Values of all thresholds and constants for all experiments on Algorithm 2.

Name	Value	Unit
COLS	180	pixels
ROWS	190	pixels
Timestamp Threshold (T)	25,000	microseconds
Low Timestamp Threshold (T_{low})	100	microseconds
Size of Groups (S)	0.005	-
Length of Regular Interval (I)	25,000	microseconds

4.4.1 Thresholds and Parameters

The system specification and camera properties are the same as those in Section 3.6.1 for the experiments on Algorithm 2. We have introduced two new thresholds for Algorithm 2, while others remain the same as before. The constant values and the threshold values we have chosen experimentally are shown in Table 4.1.

4.4.2 Experiments on Simple Objects

We have used simple objects with DVS camera for these experiments. Note that for all experiments we have used the output reported by Algorithm 1 in Section 3.6. Green dots are used to represent events that are moving away from the interior, while magenta dots are used to represent events moving towards the interior.

Round looming object A round object is moving towards the viewer in this experiment. We can see in Figure 4.9 that most of the dots are green which indicates that those events are pointing away from the interior of the object. Though there are few magenta dots which represents events moving towards the viewer, overall the number of green dots is significantly larger than the number of magenta dots. The final output of Algorithm 2 is looming.

Here the total length of the video is 3.92 seconds, and the total execution time of Algorithm 2 is 1.13 seconds over all DVS events. The total number of events is 668530. That means it takes 1.69 microseconds to process each event on average. The results are shown

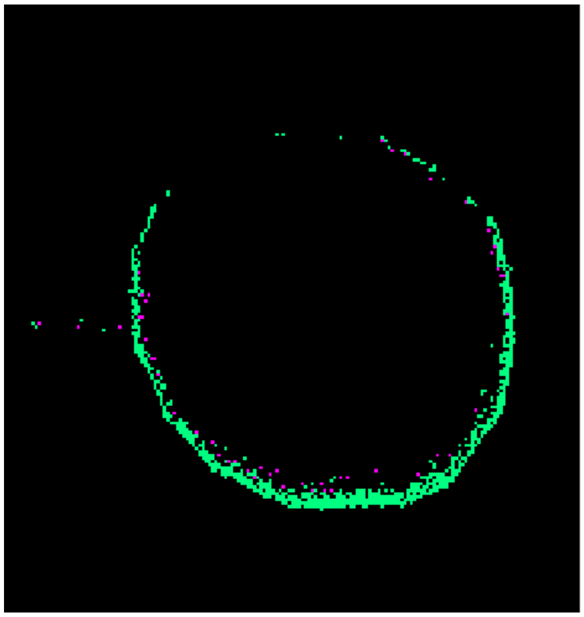


Figure 4.9: A round looming object.

Table 4.2: Results of the experiment with round looming object.

Name	Value	Unit
Total number of events	668530	-
Runtime	1.13	seconds
Length of video	3.92	seconds
Runtime per event	1.69	microseconds
Number of positive events	431	-
Number of negative events	151	-
Final output	Looming	-

in Table 4.2.

Round object moving sideways In Figure 4.10 a round object is moved from right to left. We can see the majority of green dots are on the left side of the object, while majority of magenta dots are on the right side of the edge of the object. The results are shown in Table 4.3.

Square object with little movement In this experiment, we have a square object which has very few movement (Figure 4.11). The results are shown in Table 4.4.

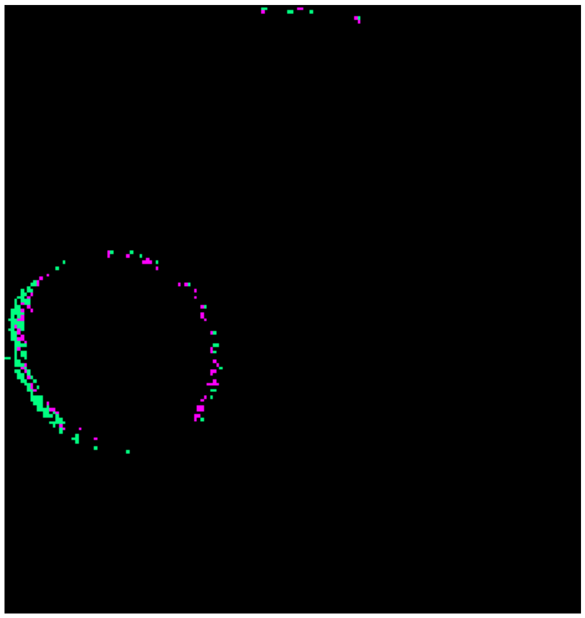


Figure 4.10: A round object moving sideways.

Table 4.3: Results of the experiment with round object moving sideways.

Name	Value	Unit
Total number of events	672201	-
Runtime	1.9	seconds
Length of video	2.81	seconds
Runtime per event	2.82	microseconds
Number of positive events	228	-
Number of negative events	120	-
Final output	Not looming	-

Table 4.4: Results of the experiment with square object with little movement.

Name	Value	Unit
Total number of events	655203	-
Runtime	1.00	seconds
Length of video	5.90	seconds
Runtime per event	1.52	microseconds
Number of positive events	26	-
Number of negative events	27	-
Final output	Not looming	-

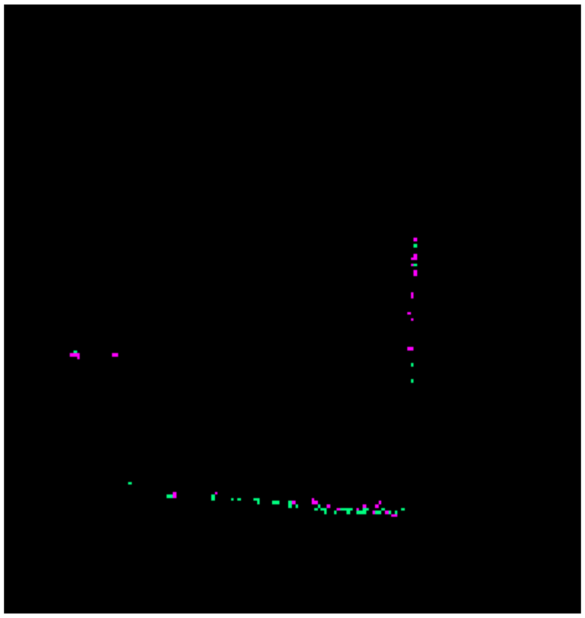


Figure 4.11: A square object with little movement.

Table 4.5: Results of the experiment with square object moving from top to bottom.

Name	Value	Unit
Total number of events	765584	-
Runtime	1.03	seconds
Length of video	4.45	seconds
Runtime per event	1.34	microseconds
Number of positive events	1084	-
Number of negative events	965	-
Final output	Not looming	-

Square object moving from top to bottom We can easily understand the output of this experiment just by seeing Figure 4.12. Here a square object is moving from top to bottom of the frame. The results are shown in Table 4.5.

Square object moving sideways Here in Figure 4.13 we can see that the left edge of the object is mostly magenta, which states that the events on that edge are moving towards the center. On the other hand, the right edge is mostly green which means the events on that edge are moving away from the center. Thus, the object is moving from left to right. The results are shown in Table 4.6.

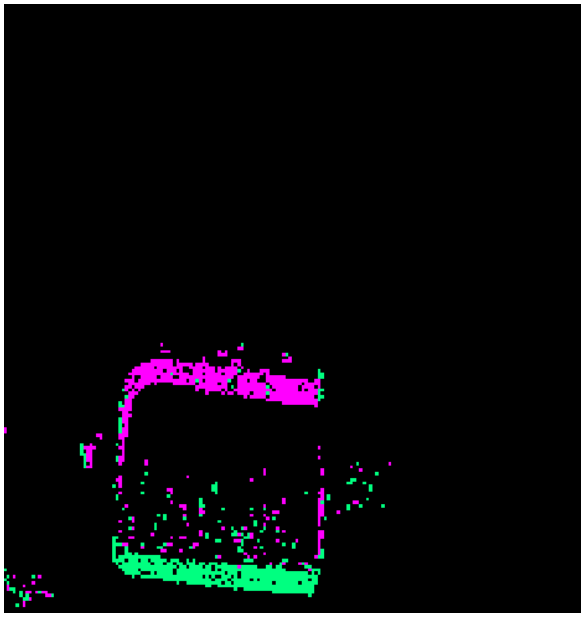


Figure 4.12: A square object moving from top to bottom of the scene.

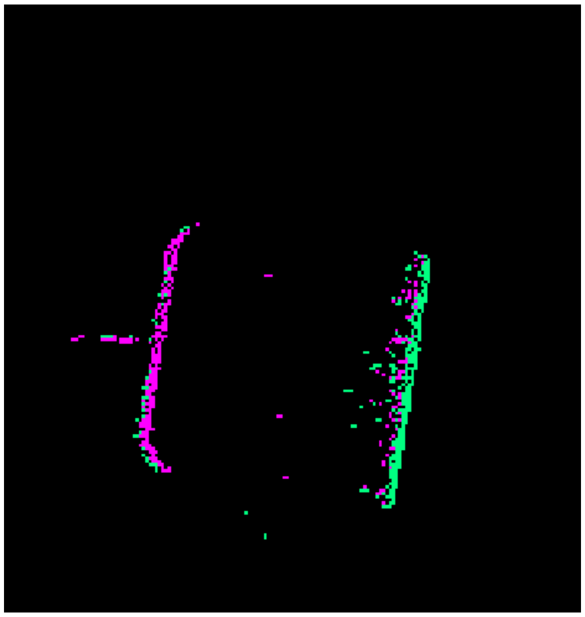


Figure 4.13: A square object moving from left to right.

Table 4.6: Results of the experiment square object moving sideways.

Name	Value	Unit
Total number of events	792649	-
Runtime	2.03	seconds
Length of video	10.07	seconds
Time per event	2.59	microseconds
Number of positive events	371	-
Number of negative events	388	-
Final output	Not looming	-

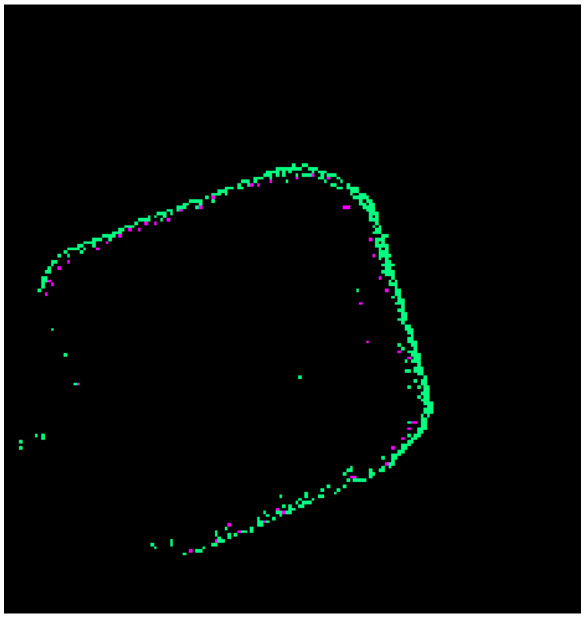


Figure 4.14: A looming square object.

Looming square object This experiment has a clear visualization of looming. The square object is coming towards the viewer (Figure 4.14). We have a large number of green dots on the overall edge comparing to the number of magenta dots. Note that the left edge of the object is not entirely detected by our algorithm, but it does not affect the output. The results are shown in Table 4.7.

Round object with looming Our last experiment is with a round object coming towards the viewer (Figure 4.15). Most of the dots on the edge of the object is green, which states that the events are pointing outside. Though the entire edge is not detected, the algorithm

Table 4.7: Results of the experiment with looming square object.

Name	Value	Unit
Total number of events	850175	-
Runtime	1.93	seconds
Length of video	5.10	seconds
Runtime per event	2.27	microseconds
Number of positive events	445	-
Number of negative events	95	-
Final output	Looming	-

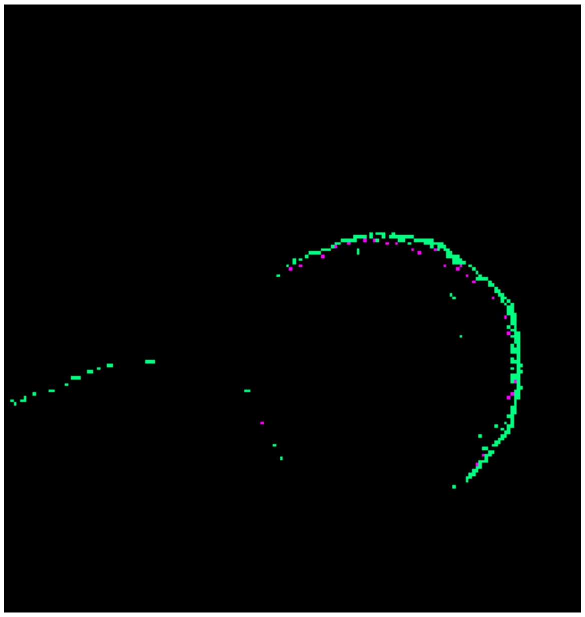


Figure 4.15: A looming round object.

provides acceptable output. The results are shown in Table 4.8.

4.4.3 Summary

These experiments produce results which accurately detects whether there is a looming object in the scene. We can see that the time to process each event in Algorithm 2 ranges from 1.34 microseconds to 2.82 microseconds on average depending on the size of the input.

Table 4.8: Results of the experiment with round object with looming.

Name	Value	Unit
Total number of events	657771	-
Runtime	1.09	seconds
Length of video	4.88	seconds
Runtime per event	1.65	microseconds
Number of positive events	228	-
Number of negative events	32	-
Final output	Looming	-

4.5 Limitations

Though our approach for computing the optical flow worked well on our experiments in Chapter 3, there are some limitations in our approach of detecting looming object. In this section we will discuss these limitations.

Multiple Moving Objects Our method gives accurate output when there is a single moving object in the scene. When there are multiple moving objects sharing the same scene, Algorithm 1 successfully detects the movement but Algorithm 2 for looming detection does not work well. The reason behind this is the calculation of the interior point (Algorithm 4) simply computes the average of all linked boundary events. If these events come from different objects, the interior point computed will be wrong. In that case, we get an average of all objects, which is not our goal. An example of this limitation is shown in Figure 4.16.

Shape of the Objects Another limitation of our algorithm is that the accuracy of the result depends on the shape of the object. If we have a convex-shaped object, our algorithm successfully detects the interior point and therefore detects the type of movement. But if the object shape is not convex, the interior point does not always lie inside the object. It may lie outside of the object. In that case our algorithm is unable to detect the true movement of the object. Also, the number of arrows pointing towards the interior and the number of arrows pointing away from the interior can be misleading when the object is not convex.

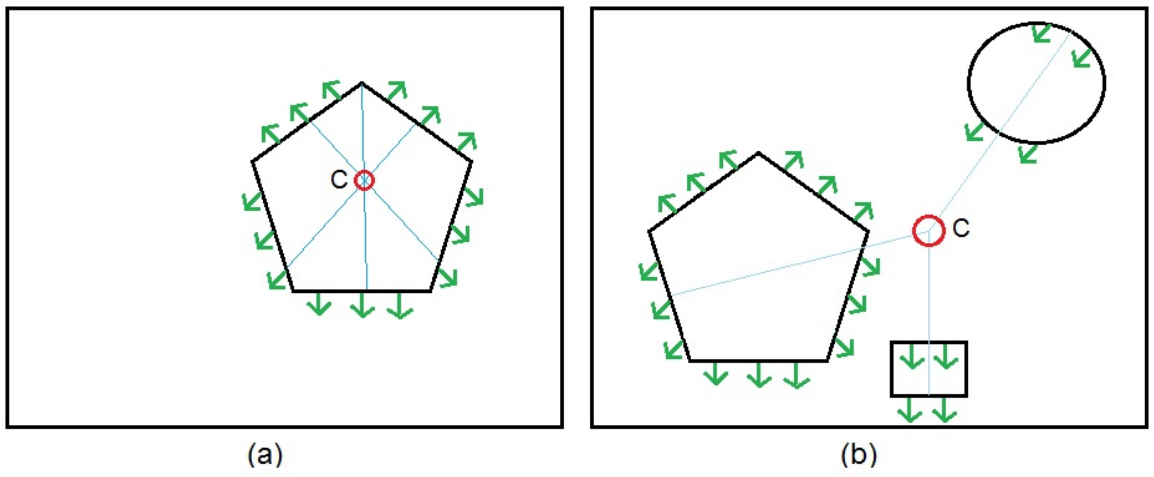


Figure 4.16: (a) A single object in a scene, the interior point computed lies inside the object. Therefore the direction of movement can be calculated accurately. (b) Multiple objects in a scene, the interior point is in the middle of the scene. The direction of different objects cannot be reported accurately.

An example of such limitation is shown in Figure 4.17.

Objects with Internal Patterns Our algorithm is not accurate with objects whose interior does not have constant luminance. Our algorithm can report events as edges for the internal patterns for objects such as those in Figure 4.18. When the object is in motion, the internal patterns can create significant changes in log-luminance and therefore our algorithm can detect the patterns as separate objects.

Isoluminant Colours If the object and the background have different colours that lead to the same luminance, the Davis DVS cannot distinguish between the foreground object and the background. Therefore no movement will be detected at all, and our algorithm will not even receive input. This is a limitation of the DVS. It should be noted that detecting this type of motion is also difficult for human [18], and the DVS is designed based on a model of the human retina.

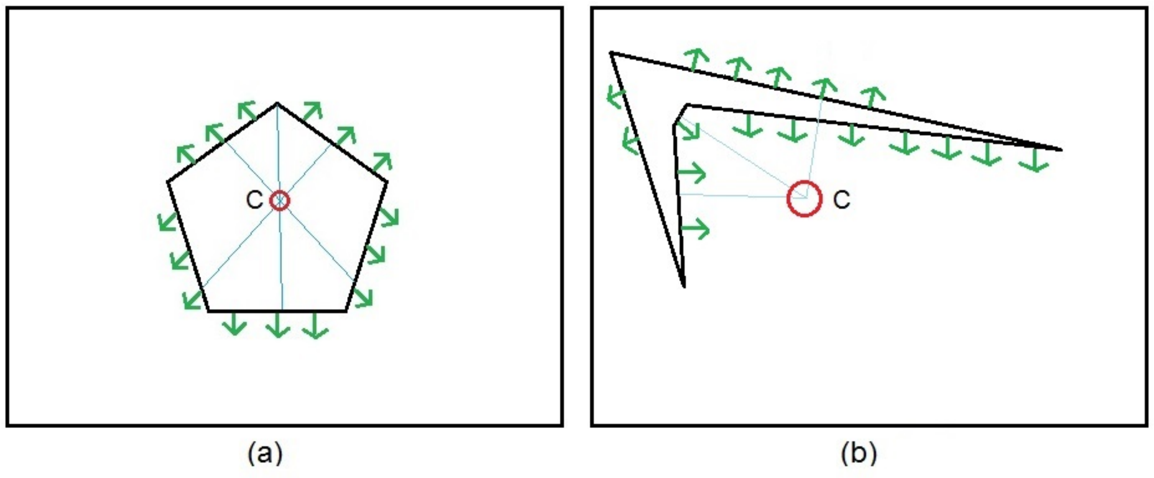


Figure 4.17: (a) A convex object in a scene, the interior point lies inside the object. Therefore the direction of movement can be calculated accurately. (b) A non-convex object in a scene, the interior point does not lie inside the object. Also, the number of arrows pointing towards the interior point is smaller than the number of arrows pointing away from the interior point, even though the object actually is looming.

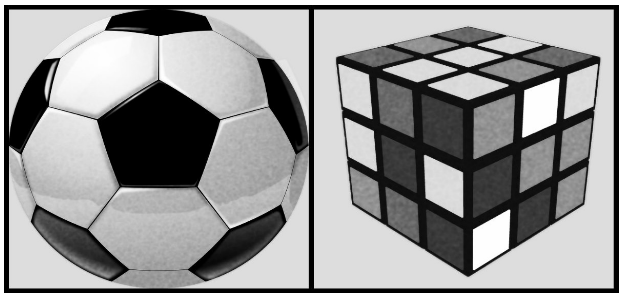


Figure 4.18: Objects with bold patterns.

Chapter 5

Conclusion

In this thesis we have proposed a non-conventional system which can detect looming objects. Our method introduces a variation of Reichardt Detector, which we employ on the event stream from the Davis Dynamic Vision Sensor (DVS) to compute optical flow. The optical flow algorithm reports the events with motion detected in the scene, delivering the events to our looming detection algorithm. Finally, our looming object detection algorithm reports if there is a looming object or not.

We have developed a series of algorithms to detect motion, analyze the motion and report whether the motion can be classified as looming or not. We have experimented our algorithms with various objects and backgrounds, with a variation on the length, brightness, etc. of the video. Our looming detection algorithm is accurate in all cases tested in which there is a single object in the scene. Each DVS event can be processed in under 5 microseconds, so that the advantages of using event-based cameras are preserved. Our algorithm can be used in real-time in applications such as robotics and autonomous vehicles.

5.1 Future Work

Our developed algorithms have delivered promising results, although it currently is restricted to situations with a single object in a scene. There are many directions for future works. Some of them are listed below.

Adaptive thresholds. The current algorithms proposed require a number of thresholds and parameters to be set experimentally by the user. It is desirable to have a system that can set these thresholds automatically and adapt to changing conditions when needed.

Multiple objects in a scene. A natural extension of our algorithm is analyze multiple objects in a scene. Instead of simply linking edge pixels and assuming that they are all part of the same object, more sophisticated clustering techniques can be used to group edges into multiple objects.

Handling self-motion. If the camera is put on a moving platform such as a robot, it may be necessary to distinguish between moving objects and stationary objects that the camera is moving towards. Information obtained from other sensors such as accelerometers may be used to make this distinction, if we can also obtain the velocity of the moving objects in the scene, then the sensor readings can be used to compensate for self-motion. The velocity of the moving objects may be determined by modifying our optical flow algorithm, so that a focus of expansion can be computed for each detected object.

Computing velocity and time-of-contact. Having velocity and time-of-contact of looming objects can be helpful in many real-world applications such as vehicle navigation and assistive devices for blind people.

Shape of objects. Currently our algorithm works well with objects having convex shapes, but may not be accurate if the object does not have a convex shape. A different approach to classify object movement may not have such a restriction.

Bibliography

[1] P. Bardow, A. J. Davison, and S. Leutenegger. Simultaneous optical flow and intensity estimation from an event camera. In *Proceedings of the IEEE Conference on Computer Vision and Pattern Recognition*, pages 884–892, 2016.

[2] S. S. Beauchemin and J. L. Barron. The computation of optical flow. *ACM computing surveys (CSUR)*, 27(3):433–466, 1995.

[3] K. Beyer, J. Goldstein, R. Ramakrishnan, and U. Shaft. When is nearest neighbor meaningful? In *International conference on database theory*, pages 217–235. Springer, 1999.

[4] A. Burnetas, D. Solow, and R. Agarwal. An analysis and implementation of an efficient in-place bucket sort. *Acta Informatica*, 34(9):687–700, 1997.

[5] F. Dramas, S. J. Thorpe, and C. Jouffrais. Artificial vision for the blind: a bio-inspired algorithm for objects and obstacles detection. *International Journal of Image and Graphics*, 10(04):531–544, 2010.

[6] M. Egelhaaf and W. Reichardt. Dynamic response properties of movement detectors: theoretical analysis and electrophysiological investigation in the visual system of the fly. *Biological Cybernetics*, 56(2-3):69–87, 1987.

[7] J. A. G. Franco, J. L. del Valle Padilla, and S. O. Cisneros. Event-based image processing using a neuromorphic vision sensor. *Power, Electronics and Computing (ROPEC), 2013 IEEE International Autumn Meeting on*, pages 1–6, 2013.

[8] T. Fülöp and A. Zarandy. Bio-inspired looming object detector algorithm on the eye-ris focal plane-processor system. *2010 12th International Workshop on Cellular Nanoscale Networks and their Applications (CNNA 2010)*, pages 1–5, 2010.

[9] B. A. Galler and M. J. Fisher. An improved equivalence algorithm. *Communications of the ACM*, 7(5):301–303, 1964.

[10] P. Gil-Jiménez, H. Gómez-Moreno, R. J. López-Sastre, and A. Bermejillo-Martín-Romo. Estimating the focus of expansion in a video sequence using the trajectories of interest points. *Image and Vision Computing*, 50:14–26, 2016.

[11] T. Gollisch and M. Meister. Eye smarter than scientists believed: neural computations in circuits of the retina. *Neuron*, 65(2):150–164, 2010.

[12] L. He, T. Chia, and C. Yang. A geometric invariant approach to human face verification. *Journal of information science and engineering*, 22(3):511, 2006.

[13] B. K. Horn and B. G. Schunck. Determining optical flow. *Artificial intelligence*, 17(1-3):185–203, 1981.

[14] iniLabs. Dynamic vision sensor. `http://inilabs.com/products/dynamic-vision-sensors/`. Accessed: 2016-11-1.

[15] H. Kim, S. Leutenegger, and A. J. Davison. Real-time 3d reconstruction and 6-dof tracking with an event camera. In *European Conference on Computer Vision*, pages 349–364. Springer, 2016.

[16] K. Li and J. Malik. Fast k-nearest neighbour search via dynamic continuous indexing. In *International Conference on Machine Learning*, pages 671–679, 2016.

[17] S. C. Liu and T. Delbruck. Neuromorphic sensory systems. *Current opinion in neurobiology*, 20(3):288–295, 2010.

[18] Zhong-Lin Lu, Luis A Lesmes, and George Sperling. The mechanism of isoluminant chromatic motion perception. *Proceedings of the National Academy of Sciences*, 96(14):8289–8294, 1999.

[19] A. K. Maan, D. A. Jayadevi, and A. P. James. A survey of memristive threshold logic circuits. *IEEE transactions on neural networks and learning systems*, 28(8):1734–1746, 2017.

[20] C.D. Pantilie and S. Nedevschi. Real-time obstacle detection in complex scenarios using dense stereo vision and optical flow. *13th International IEEE Conference on Intelligent Transportation Systems*, 2010.

[21] S. S. Park and A. Sowmya. Autonomous robot navigation by active visual motion analysis and understanding. *Proceedings of IAPR Workshop on Machine Vision Applications*, 1998.

[22] D. Regan and K.I. Beverley. Looming detectors in the human visual pathway. *Vision research*, 18(4):415–421, 1978.

[23] W. Reichardt and M. Egelhaaf. Properties of individual movement detectors as derived from behavioural experiments on the visual system of the fly. *Biological Cybernetics*, 58(5):287–294, 1988.

[24] I. Ridwan and H. Cheng. *An Event-Based Optical Flow Algorithm for Dynamic Vision Sensors*, pages 182–189. Springer International Publishing, Cham, 2017.

[25] F. C. Rind and P. J. Simmons. Seeing what is coming: building collision-sensitive neurones. *Trends Neurosci.*, 22:215220, 1999.

[26] Spikenet technology. `http://www.spikenet-technology.com/`. Accessed: 2016-11-01.

[27] S. Stone. Using Auditory Augmented Reality to Understand Visual Scenes. Master's thesis, University of Lethbridge, LEthbridge, Alberta, Canada, 2017.

[28] M. Subbarao. Bounds on time-to-collision and rotational component from first-order derivatives of image flow. *Computer Vision, Graphics, and Image Processing*, 50(3):329–341, 1990.

[29] M. S. Tata, N. Alam, A. L.O. Mason, G. Christie, and A. Butcher. Selective attention modulates electrical responses to reversals of optic-flow direction. *Vision research*, 50(8):750–760, 2010.

[30] User guide: Aer. `https://inilabs.com/support/software/jaer/#h.63b8cxspmdby/`. Accessed: 2017-09-09.

[31] User guide: caer. `https://inilabs.com/support/software/caer/`. Accessed: 2017-09-09.

[32] User guide: jaer. `https://inilabs.com/support/software/jaer/`. Accessed: 2017-09-09.

[33] User guide: libcaer. `https://inilabs.com/support/software/libcaer/`. Accessed: 2017-09-09.

[34] B. Webb. Robots in invertebrate neuroscience. *Nature*, 417(6886):359–363, 2002.