BIO-INSPIRED LOOMING OBJECT DETECTION USING EVENT-BASED CAMERAS

GUILLERMO SÁNCHEZ ATTOLINI
Bachelor of Science, Instituto Tecnológico y de Estudios Superiores de Monterrey, 2013

A thesis submitted
in partial fulfilment of the requirements for the degree of

MASTER OF SCIENCE

in

COMPUTER SCIENCE

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

© Guillermo Sánchez Attolini, 2019

BIO-INSPIRED LOOMING OBJECT DETECTION USING EVENT-BASED CAMERAS

GUILLERMO SÁNCHEZ ATTOLINI

Date of Defence: December 16, 2019

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. Habiba Kadiri Chair, Thesis Examination Committee	Associate Professor	Ph.D.

Dedication

In loving memory of my father, wherever he may be.

To the rest of my family, whose locations are known.

Abstract

We present a novel looming object detection method for event-based cameras. Event-based cameras capture a scene without generating redundant information thus reducing the needed transmission power, bandwidth, and computations to process the redundant information generated by conventional frame-based cameras. Regular computer vision algorithms cannot be directly applied to them. Conversely, existing event-based algorithms that detect looming objects have some limitations. In this thesis, an existing bio-inspired looming object detection algorithm for frame-based cameras was adapted for an event-based camera. The adapted algorithm was then used as part of a novel looming object detection algorithm that is fast and capable of detecting multiple looming objects in a scene. We tested our approach on the Davis Dynamic Vision Sensor event-based camera.

Acknowledgments

I wish to show my appreciation for Dr. Howard Cheng; for his support during my admission process; for always keeping an open door whenever I needed advice on my research, my courses, or any other aspect since I arrived to Lethbridge; for believing in me. I thank Dr. Cheng the most.

I thank Dr. John Zhang and Dr. Matthew Tata for their helpful advice and support throughout my research. I thank Dr. Cheng and Dr. Tata for providing the research equipment. I also thank Behnam Kamranian and Jihoon Og for their help in creating the test datasets.

I also wish to extend my appreciation to all the faculty and colleagues in the Mathematics and Computer Science department. They always allowed me to distract them with questions and provided many conversations that broadened my knowledge or gave me a blissful distraction whenever I needed it.

I am grateful for all the special people I have met in Lethbridge. Such as the Friesens who brought me to Lethbridge and welcomed me into their home, and my friends who gave me the reasons to see Lethbridge as my home.

Finally, I wish to show my appreciation for my parents, my sister and everyone who has inspired me to become a better version of myself and pursue this endeavour.

Contents

List of Tables

List of Figures

Chapter 1

Introduction

Looming is the apparent enlargement of an object in a scene, indicating that the object is approaching an observer. Detecting this optical effect is important in nature for the survival of animals. Being able to detect a looming object or another animal helps to avoid collisions and predators. For similar reasons, this task has also become important in the field of robotics. In an age where self-driving cars are beginning to become commonplace, it is important that these machines are able to avoid collisions for the safety of their passengers and others. A self-driving car should be not only as quick as a human in reacting to avoid hitting a pedestrian, they should also be better. A second example is unmanned aerial vehicles which some companies have gambled will become the way persons will receive purchased goods in the future. Applications can even help people with special needs navigate in their daily life [25].

It is not enough to only detect looming objects. A low response latency is also an important factor in the success of avoiding collisions. In order to reduce latency, it is important to consider the entire process from capturing the data representing a looming object to the point when the looming object is detected. A conventional frame-based video camera captures an image frame every few milliseconds. As a result, there is a minimum latency for those milliseconds between frames. Also, these data have to be transmitted to be processed, and the complexity of the processing algorithm is at least the same complexity as processing every pixel in each of the image frames. One obvious way to reduce latency is to reduce the time between frames. However it also increases the amount of generated data which in

turn increases the transmission and processing time. Another approach is to increase the bandwidth and processing power, but at the cost of increasing the electrical consumption of the device and hardware requirements. This approach would exclude applications where the processing power or battery storage is limited. Another solution is to use a camera that reduces the latency between frames, without increasing the amount of information. In Chapter 2, we will introduce event-based cameras, a type of cameras that respond quickly to changes in the scene but only generate relevant information. This approach reduces hardware and bandwidth requirements while reducing latency in the response time.

A disadvantage to using an event-based camera is that most of the existing state-of-the-art algorithms for computer vision tasks such as looming object detection cannot be directly used on the output of an event-based camera. For the problem of looming object detection, Ridwan and Cheng [23] designed an optical flow algorithm for event-based cameras. The optical flow algorithm generates optical flow events which are then used as the input for a looming object detector [22]. The approach solves the problems stated above with its own limitations. For example, the algorithm cannot detect multiple looming objects or concave objects.

In this thesis we analyze an existing looming object detection algorithm for frame-based cameras [5, 28]. This is a bio-inspired algorithm based on a type of mouse retina cells that react to looming motions. Adapting this algorithm for event-based cameras requires a careful examination of its operations as well as appropriate choices of data structures, so that the resulting algorithm behaves in a similar fashion but is much more efficient.

As with the original frame-based algorithm, the adapted event-based algorithm does not properly detect looming objects that are larger than these cells. Instead, the scene is partitioned into a number of overlapping cells of different sizes and the event-based algorithm is applied to each cell. The new approach allows looming objects of different sizes to be detected, even when multiple objects are looming at the same time.

1.1 Main Contributions

In this thesis we present an adaptation of a bio-inspired looming detection algorithm [5, 28] that was originally created for traditional frame-based cameras into event-based cameras. The accuracy of the event-based algorithm is evaluated by comparing its output against that from the frame-based counterpart. It is shown that the results are accurate qualitatively, and the event-based algorithm is significantly faster than the frame-based algorithm.

A new event-based optical flow algorithm is then obtained using the output of the bio-inspired looming detection algorithm. Finally, we proposed a new method of detecting multiple looming objects by partitioning the observed scene into cells of different sizes. It is shown that the final looming object detection algorithm is accurate, and has run time similar to the previous algorithm of Ridwan [22] despite being able to detect multiple looming objects in a scene. It should also be noted that the algorithms in this thesis are truly event-based—a decision can be made for each event received from the camera without any need to collect the events into frames to process in batches.

1.2 Organization of the Thesis

In Chapter 2, background concepts are introduced and previous approaches for both frame-based and event-based cameras to detect looming objects are described. Chapter 3 presents the adapted bio-inspired algorithm for event-based input, as well as experimental results on the effectiveness of the adapted algorithm. Chapter 4 shows how the adapted algorithm can be used to compute optical flow, and presents a novel approach to multiple looming object detection. Experimental results are shown to demonstrate the effectiveness of the final algorithm. Concluding remarks and possible directions for future work are given in Chapter 5.

Chapter 2

Background

2.1 Looming Object Detection

Looming is the effect perceived by an observer when an object is approaching it. The object appears to grow larger as it comes nearer [24]. There are neural circuits in animals that detect approaching objects [21, 27].

In computer vision, we can take advantage of the looming effect to automatically detect if an object is approaching a digital camera. There are several algorithms for conventional cameras [8, 18, 19, 26]. The problem lies in separating the object from the rest of the image, and determining if it is actually growing larger. It is in the former that problems arise. Objects can have irregular shapes, and we could mistake an object for the background of the image. Additionally, it is hard to distinguish multiple objects in the image. The problem becomes even harder if there are multiple objects of different sizes. This method of detection can be fooled if an object is actually growing, not approaching.

In robotics we usually need a way to avoid obstacles, or get out of the way of a moving object. In some applications (for instance a self-driving car), these reactions need to be as quick as possible. If we need to perform a significant amount of computations to detect looming objects, the reaction time may be high. Even worse, a robot may have limited hardware which is not dedicated solely to the camera.

Figure 2.1: Example of Dynamic Range.

2.2 Event-Based Cameras

Conventional digital video cameras present a succession of images to represent a video. As with digital cameras, each image is taken by an array of light sensors, in which each sensor represents a single pixel. A common frame rate is 24 frames per second (fps). The images, or frames, are usually taken every 41.67 milliseconds, enough to fool the human eye into thinking it is looking at a continuous feed.

We should note that every frame a camera takes is a complete new image of whatever is being filmed. Even if the actual image has not changed, a video camera will continue to take a new image every 41.67 milliseconds and either stores it or transmits it. There are compression algorithms that will remove redundant data between frames. These of course need extra computations to run.

A scene, in a single image, contains areas with different light intensities. A scene with a *high dynamic range* refers to an image where the difference between the lowest and the highest intensity is very large. Conventional cameras have to reduce or increase the general brightness of an image, to reveal the details of these areas. Favouring one extreme of the range obscures the other one. Figure 2.1 shows an example of a photograph with high dynamic range. The overall brightness in the image on the left is reduced allowing it to show the details of the sky. The brightness is increased on the right allowing it to show the details of the city, but the sky is too bright to appreciate its details.

In dim light a frame-based camera will need to reduce its frame rate to increase the exposure time. This is needed so that enough light reaches the camera's sensors. Each frame in a video is a sample of the scene being filmed, hence it is affected by sampling issues. When the frame rate is below the Nyquist Rate [9], the video will present temporal aliasing. In other words, the frame rate has to be twice as fast as the movement in the scene it is filming, or its representation will be incorrect. Some applications, such as self-driving cars, will need to react to quicker movements than the frame rate of a conventional camera. There are commercial cameras that can capture 1000 frames per second [1], rather than the usual 24, however these will generate proportionally a huge amount of redundant data.

Neuromorphic engineering is a concept that uses analog and digital electronic circuits to implement scalable architectures that emulate the architecture of biological brains [16]. These architectures use a threshold logic. Just as neurons, an element in a neuromorphic architecture gets activated when the weight of its input goes above a certain threshold. Event-based cameras, such as the Davis Dynamic Vision Sensor [12], are an example of a neuromorphic architecture.

Event-based cameras take a different approach to traditional cameras. As their name states, instead of sending frames they send events. An event is triggered when the log luminance detected by any pixel in the sensor of the camera changes beyond a particular threshold. This means, if the input becomes darker or brighter. Each pixel in the sensor is independent from each other, and each will report this change in log luminance asynchronously. An event consists of the coordinates of the pixel, the time stamp of the event, and the polarity of the change (from bright to dark or from dark to bright).

The event-based camera used in the experiments in this thesis is the Dynamic Vision Sensor [12] (DVS), specifically the DVS240 [11] model manufactured by Inivation. It is a low-power-consumption camera, with reduced data rate, high speed (reacts in under a millisecond) and a high dynamic range.

A sample comparison of a regular camera and the DVS can be found in Figure 2.2. On

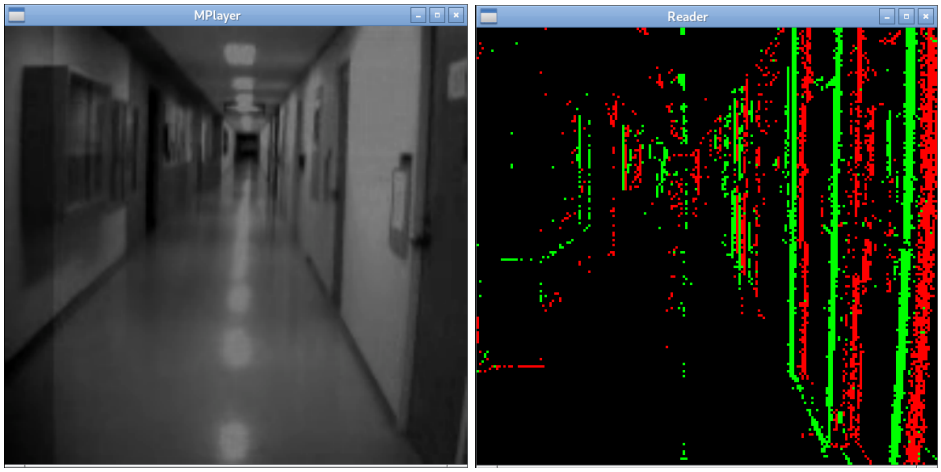


Figure 2.2: The Dynamic Vision Sensor (DVS) being carried down a hallway.

the left is a frame of a regular video output (which the DVS can also generate). On the right is a representation of the events in the DVS captured for a window of time. The green pixels are positive (from dark to bright) events, and the red are negative events.

There are several benefits when using an event-based camera. These cameras do not need the computations a frame-based camera requires to compress the information, because they do not generate redundant information. When the scene contains a high dynamic range, there is no need to change brightness settings, for every pixel in an event-based camera is independent of the others. The camera will not take the pixels' luminance into account, only when there was some significant change in log luminance. Because the sensor is constantly exposed, an event-based camera can still capture scenes in dim light. As long as the changes in log luminance are greater than the camera's threshold, they will be reported. Event-based cameras are less susceptible to aliasing. For instance, the DVS [11, 12] has a minimum latency of 12 microseconds, and a typical latency of 30 to 1000 microseconds. In comparison, a 1000 fps camera has a minimum latency of one millisecond.

There is a drawback with event-based cameras, though. Computer vision algorithms, such as those for detecting a falling person [17], that have been implemented in regular cameras cannot be used directly with the DVS. Our work is to implement new algorithms or adapt existing ones.

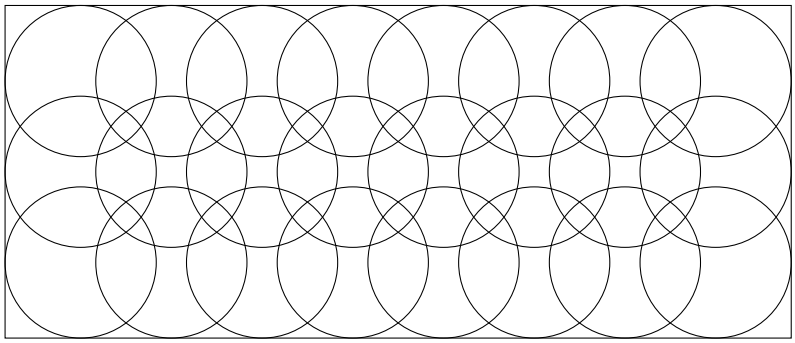


Figure 2.3: Sample receptive field.

The events in the DVS are represented using the Address-event Representation [13] (AER). This is inspired on how regular neurons communicate with each other. The maker of the DVS provides a library that uses an implementation of AER for both Java [13] and C [14, 15]. These libraries define a *Polarity Event* that contains information about a generated event, including the x and y coordinates, time stamp, and polarity of the pixel that changed.

2.3 Bio-Inspired Looming Detection

Fülöp and Zarándy [5, 28] implemented a mathematical model that describes one type of cells in the mouse retina. This cell gets activated when a dark object is looming. These cells (also called receptive fields) are arranged into a grid (Figure 2.3). The cells may overlap, and each cell contains an equal number of excitatory and inhibitory channels.

An inhibitory channel responds to changes from dark to light. An excitatory channel responds to changes from light to dark. If a dark object is approaching, several elements in the light sensor will turn from light to dark, firing the excitatory channels. At the same time, there will be little response from the inhibitory channels. The opposite happens when a dark object is moving away from the camera (recessing). When a dark object is moving sideways, the same number of elements will go from dark to light as from light to dark. In other words, the same amount of excitatory and inhibitory channels will fire.

The model just described can only detect dark looming objects in front of a bright

background. Fülöp and Zarándy [6, 28] improved their model to detect both dark and bright looming objects. The new model focuses only on whether there is a change at all. The excitatory channel is redefined to respond when the intensity changes in a pixel where up until now there has been no change. The inhibitory channel now responds to a pixel that had intensity changes recently, but is remaining constant now. As an object moves in the picture its edges obscure the background, provoking changes in intensity, and increasing the activity in the excitatory channel. After some time the pixels affected by the edge are now inside the object, their changes in intensity become very small, provoking activity in the inhibitory channel. When an object approaches, there will be an outline of excitatory pixels in the shape of the object, surrounding a similar outline of inhibitory pixels. There will be more excitatory than inhibitory pixels. This can be seen in Figure 2.4, where the edge of an approaching square generates a perimeter of excitatory pixels, with an trailing perimeter of inhibitory pixels. The opposite happens when the object is moving away, which is also called recessing. When an object is presenting lateral movement, the number of activated excitatory and inhibitory channels will be the same. When the difference in excitatory and inhibitory channels increases, there is a looming object. When the difference decreases there is a recessing object. When the difference does not change, there is a lateral movement or no movement at all.

The new model is achieved by preprocessing the image, through a series of functions. First, a temporal convolution is applied against past values:

$$u_{i,j}(t) = \sum_{n=0}^{s-1} pic_{i,j}(t-n)w_n \tag{2.1}$$

where $pic_{i,j}(t)$ is the intensity value of the image at positions i and j at time t, s is the number of frames to compare, t is the present time, and w is a vector of weights, one per frame. For example: $w = \left[1; -\frac{1}{2}; -\frac{1}{2}\right]$.

Equation (2.1) sets to zero all the pixels that have not changed in the last few frames. It is comparing the values of each pixel in the current frame against values in previous ones.

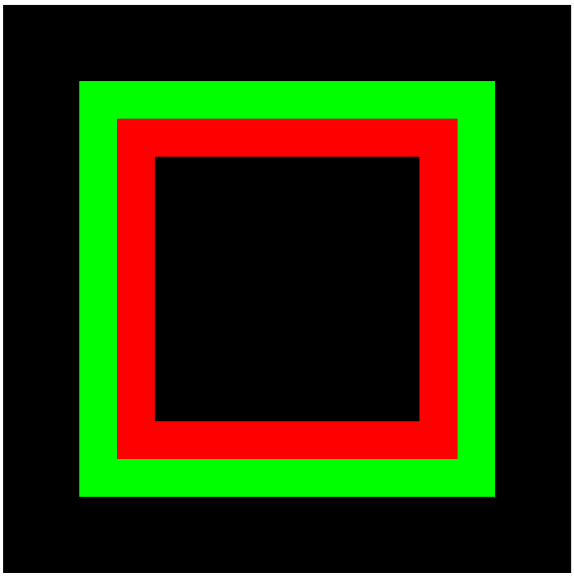


Figure 2.4: Excitatory (green) and Inhibitory (red) pixels.

Using the sample weights, if the pixel has not changed in intensity for 3 frames, the current intensity will be multiplied by 1, the previous two will be multiplied by $-\frac{1}{2}$, so the value of u will be 0. If the pixel has just changed intensity, the sum will be non-zero. An increase in luminance will give a positive u, while a decrease in luminance will give a negative u. The magnitude of u will be directly proportional to the magnitude of the difference in luminance.

To remove noise and small changes in luminance, the following functions are used:

$$g(u) = \begin{cases} (u_{i,j} + o_d)w_d & \text{if}(u_{i,j} + o_d) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.2}$$

$$h(u) = \begin{cases} (-u_{i,j} + o_l)w_l & \text{if}(-u_{i,j} + o_l) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.3}$$

$$pic_{diff_{i,j}}(l) = g(pic_{i,j}(l)) + h(pic_{i,j}(l)) \tag{2.4}$$

Note that $h(u)$ inverts a negative u, and $g(u)$ makes a negative u = 0. The direction of

the change in luminance has no effect in the computations. o_d and o_l functions as threshold values. $g(u)$ filters negative values below o_d and $h(u)$ filters positive values under o_l. The w_d and w_l constants exist as weights to the output of the filter.

To give an example of how the offsets work. Assume $s = 2$, $w = [1; -1]$, $o_d = 0$, $o_l = 0$, $w_d = 1$, and $w_l = 1$. With this configuration $u_{i,j}(t)$ compares only the current and previous frames, the offsets have no effect, and w_d and w_l have no effect on the computations. If the pixel at (i, j) became darker by a magnitude of 5, $u_{i,j}(t) = -5$. When placed in the filters, $g(-5) = 0$ and $h(-5) = 5$. If the same configuration is used, but $o_d = -10$ and $o_l = -10$, then $g(-5) = 0$ and $h(-5) = 0$. The o_l brings the $-u_{i,j}$ back into a negative number, so the function outputs 0.

Figure 2.5 shows the effect of these equations when applied to an approaching square. The square grows one pixel at a time in every direction. All pixels are either at maximum brightness or 0. Thus, the output of the equations generates a hollow square with one-pixel thickness. This video was created artificially, and within each frame the square grows exactly one pixel in each direction. The pixels always change from the minimum intensity value to the maximum one. Because of the nature of the video, only one frame needs to be compared at a time, and there are no small brightness transitions. So there is no need to correct any offset. The constants to generate the image were set to: $w = [1; -1]$, $o_d = o_l = 0$, $w_d = w_l = 1$.

Using the filtered changes, the following equations calculate the response in the excitatory and inhibitory channels:

$$E_{i,j}(t) = \sum_{n=0}^{s-1} pic_{diff_{i,j}}(t-n)we_n \tag{2.5}$$

$$I_{i,j}(t) = -\sum_{n=0}^{s-1} pic_{diff_{i,j}}(t-n)wi_n \tag{2.6}$$

where i and j are the spatial coordinates of a pixel, s is the number of frames that will be

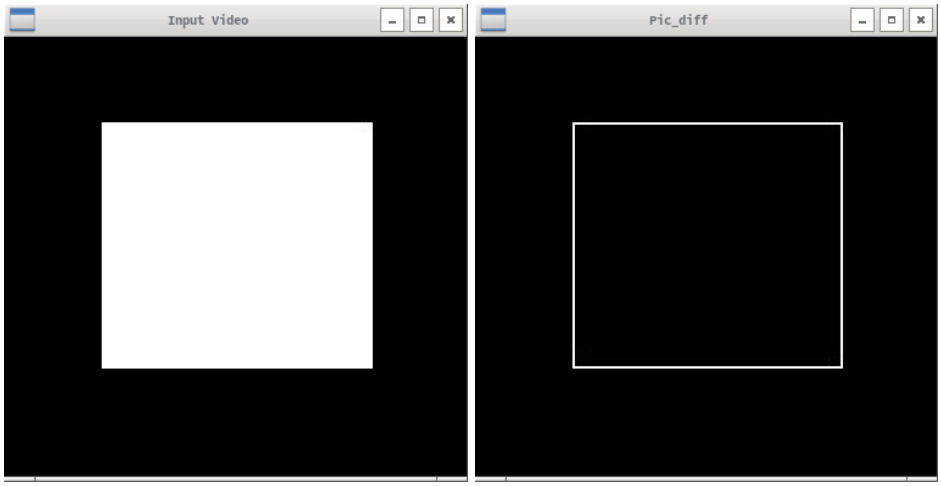


Figure 2.5: Sample output of change detection.

evaluated, and both *we* and *wi* are weights for the convolutions in the excitatory/inhibitory channels. Similar to *w*, *we* and *wi* may be set to $[1;-\frac{1}{2};-\frac{1}{2}]$, so two previous frames are compared.

Follow-up operations ensure that only large changes in luminance are taken into account, by adding a certain offset:

$$h_e(x) = \begin{cases} (x+o_e) & \text{if } (x+o_e) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.7}$$

$$h_i(x) = \begin{cases} (x+o_i) & \text{if } (x+o_i) > 0 \\ 0 & \text{otherwise} \end{cases} \tag{2.8}$$

where o_e and o_i are offset values for their respective channels.

After the individual pixel values have been calculated the method calculates the value for the whole receptive field.

$$res_{k,l} = r\left(\sum_{(i,j)\in N_r(k,l)} h_e(E_{i,j}) - h_i(I_{i,j})\right) \tag{2.9}$$

where $N_r(k,l)$ is the receptive field at position (k,l), the evaluated (i,j) takes values of

the coordinates covered by the receptive field, and r is a rectification function: $r(x) = x(sign(x)+1)/2$.

Finally, comparing the current and previous res values, the algorithm decides if the area is looming, recessing, or no-change/moving-sideways:

$$decision = \begin{cases} \text{"}recessing\text{"} & \text{if } res(t) < res(t-1) \\ \text{"}lateral/nothing\text{"} & \text{if } res(t) = res(t-1) \\ \text{"}looming\text{"} & \text{if } res(t) > res(t-1) \end{cases} \tag{2.10}$$

The problem with this approach is that either we have a receptive field that covers the whole image no matter where the centre is, or we have smaller fields that are unaware of the others. The former will have to process the entire "frame" for every pixel. However, the object may be too small, and the algorithm will spend time processing useless data. In contrast, a large object may step over the boundaries of the receptive fields. This may cause the individual receptive fields to give the wrong answer.

2.4 Optical Flow

In video processing, optical flow is the apparent motion of an object in video. It is the perceived shift of intensity from one pixel to another. The optical flow of an object can be represented as a collection of vectors at each pixel.

In the animal kingdom there are cells that get activated when a perceived object moves. These specialized neurons process a large amount of information coming from their field of view in a very short amount of time [21]. Estimating the optical flow is an important task because it is a key component in solving motion detection problems.

Though there are existing optical flow algorithms [2, 10, 4], they are mostly implemented for frame-based cameras. However, Ridwan and Cheng [22, 23] have designed an optical flow algorithm that takes the input of an event-based camera.

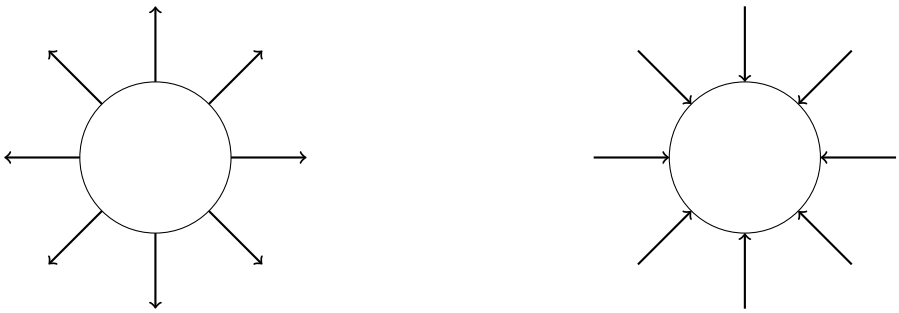
The optical flow algorithm of Ridwan and Cheng works under the assumption that if a

certain pixel had a log luminance change recently, and another pixel close to it has a change of the same polarity, the object generating that change has changed position. Using these two pixels, we can calculate a displacement vector. In an ideal example, when an object is moving only the pixels at its edges will be changing and the ones in its body will remain constant. Hence, an optical flow algorithm will output vectors at the edges of the object.

2.5 Dynamic Vision Sensor Looming Detection

Ridwan and Cheng [22, 23] developed an algorithm that uses the optical flow of an object in a video to determine if it is looming.

A looming object detection algorithm that is based on optical flow takes the vectors as input. When an object is looming, a significant portion of the vectors point away from its centre. When it is recessing, a significant portion will point towards its centre. When it is moving sideways, a significant portion of the vectors will point in the direction of motion. The algorithm computes the centre of the object as the mean of all of these vectors. Then, as explained above, if the vectors are pointing away from the centre, the object is looming. Figure 2.6 shows an example of the Optical Flow of an approaching and a recessing image.



(a) Example of optical flow in a looming figure. (b) Example of optical flow in a recessing figure.

Figure 2.6: Example of optical flow.

In a regular frame-based camera this approach involves eliminating redundant pixels, and computing frame by frame. The Dynamic Vision Sensor already does this for us. Its stream of events will only contain the changing pixels which form the contour of the

object. Ridwan's approach computes the optical flow from the input of the DVS, and uses it to detect looming objects towards the camera.

There are a few problems with this approach. First, if there are two objects in the field of view, they will be considered as one. The algorithm was meant for a single object. All similar vectors in a period of time are considered to come from the same object. The algorithm will compute the centre inside one of the objects, or outside of both, and it will produce an incorrect decision.

Second, if the object in the scene is concave, the output of the algorithm may be wrong. The centre may be computed outside of the object itself. For simplicity, Ridwan's algorithm was designed for only convex objects.

Third, the sensor is imperfect, and sometimes it shows an incomplete silhouette of the object. For example, perhaps only two-thirds of the boundary are reported by the DVS. The lack of information may be enough to compute an incorrect centre or an incorrect optical flow direction. The output would be incorrect.

Finally, an object may contain internal patterns. For example, the imperfections of a rock. These will cause the optical flow algorithm to generate extra vectors. The looming algorithm may decide that they are a part of the contour of the object, and may compute an incorrect looming decision.

Fülöp and Zarándy's algorithm, detailed in Section 2.3, has no problem detecting concave objects because it does not need to compute a centre. Unfortunately, it cannot be applied directly to an event-based camera. In the following chapter an adaptation of the algorithm will be presented, along with how it benefits from using an event-based camera.

Chapter 3

Event-based Bio-Inspired Looming Detection

3.1 Introduction

An event is a discrete log luminance change in a pixel. The changes are discrete because event-based cameras only generate events when the log luminance of an image pixel has changed above a certain threshold. Event-based algorithms refer to those that process pixel events separately as they arrive. This is opposed to frame-based algorithms in which a complete frame is generated, transmitted and processed.

Section 2.5 presented Ridwan's [22] algorithm to detect approaching objects using an event-based camera, and enumerated the situations where it would fail to give the correct decision. Section 2.3 presented Fülöp and Zarándy's [6] algorithm to detect approaching objects in frame-based cameras. This chapter will present an adapted algorithm for event-based cameras. We will discuss the adaptation's trade-offs, and the limitations of both approaches.

To adapt the algorithm, it is necessary to consider the following issues:

1. There are no frames. Any decision has to be reached by comparing incoming pixel events, against older recorded events.

2. Unlike the frame-based approach, there is not a fixed period of time between events. The present and previous events may be substantially distant in time. Hence, it is necessary to use thresholds. If the time difference between two events is greater than

the threshold the two events are not considered to affect one another.

3. The weight and offset constants in the equations are not necessary. Fülöp and Zarándy's algorithm deals with intensity levels, and these constants are used to remove small changes. The input from the event-based cameras consists of only the direction of the change. Small changes and the magnitude are not reported.

3.2 Picdiff and DVS Input

As explained in Section 2.3, event-based cameras only generate information on pixels where the log luminance has changed beyond a certain threshold. Fülöp and Zarándy's algorithm emulates this behaviour through the following steps:

1. Attenuate repeated light intensities between frames using equation (2.1).

2. Attenuate changes below the value of a threshold, and ignore the polarity of the change in intensity using equations (2.2) and (2.3).

When adapting the Fülöp and Zarándy's algorithm, it was not necessary to emulate the aforementioned steps. The output of an event-based camera is analogous to them.

To adapt the rest of the algorithm (described in equations (2.5-2.10)) the sections of the *picture* were also separated into overlapping receptive fields.

3.3 Event-based Excitatory and Inhibitory Channels

As described in Section 2.3, equations (2.5) and (2.6) measure a pixel's activity in the excitatory and inhibitory channels. Similarly, the adapted algorithm defines excitatory and inhibitory pixel-events:

Excitatory Event: an event generated by the camera. It is the first event to be received after a period of time longer than the one defined in an excitatory threshold.

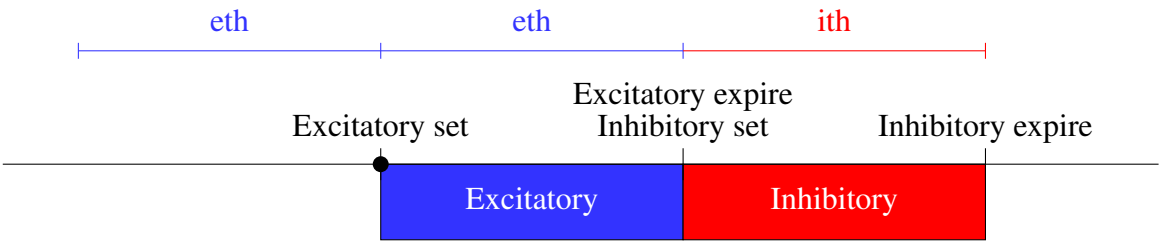


Figure 3.1: A single event timeline

Inhibitory Event: an event generated by the algorithm itself after there has been a period of inactivity (defined in an inhibitory threshold) since the last event from the camera was received at that pixel.

As mentioned above, event-based cameras generate binary information. Similarly the emulated approach does not quantify the activity in the excitatory and inhibitory channels, but only whether there is any activity at all. Hence, there is no need to emulate equations (2.7) and (2.8).

Figure 3.1 shows the lifespan of an event in a single pixel. An event is first received, such that there had not been any event for that pixel for the time specified in *eth* (Excitatory Threshold), the pixel becomes excitatory. Then, some time elapses, another period of *eth*, and the pixel is no longer excitatory. Now the pixel is considered inhibitory. After the time set in *ith* (Inhibitory Threshold) the pixel returns to a normal state.

Figure 3.2 shows the effect of receiving several consecutive events within the excitatory threshold. The first event sets the pixel in a excitatory state. Then more events continue to arrive, causing no effect, for none of their time stamps are more than one *eth* apart from the first event. The first event expires, returning the pixel to a normal status. Because the last event to arrive is newer that the expired one, the pixel is not set the pixel in an inhibitory state. Only after the last event has expired, and after the time specified in *exp_th* (expiration threshold) has elapsed, the pixel is set to the inhibitory state for the duration specified in *ith*.

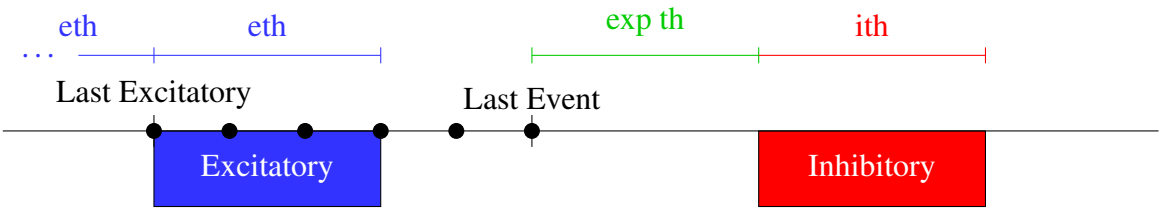


Figure 3.2: Multiple events timeline

3.4 Computation of Final Result (res)

Equation (2.9) is emulated by counting the excitatory and inhibitory events inside of the receptive field—subtracting the inhibitory count from the excitatory count and applying the same rectification function on the difference. Finally, equation (2.10) was modified so that $res(t-1)$ is a previous result of the adapted version of equation (2.9). If this previous result was above a period of time, defined in a resThreshold, the decision *Nothing* is returned. Otherwise the same path as the original equation is followed.

In total, four thresholds have been defined:

Excitatory threshold (eth): the time that must have elapsed for an event between two events for the second one to be considered excitatory. Also, the time the event remains being excitatory.

Inhibitory threshold (ith): the time an event remains being inhibitory.

Expiration threshold (exp_th): the time before a regular event expires.

res threshold (resThreshold): the maximum time between the previous res time stamp and the current res time stamp.

3.5 Supporting Data Structures

As explained in Section 2.2, an event consists of the coordinates of the pixel where it comes from, the time stamp when it occurred, and its polarity. The adapted approach keeps

track of the time stamp of the most recent event at each coordinate in a two-dimensional array. Any new event's time stamp is compared with the one stored here.

There is a difference between the required approach to process inhibitory events and the approach to process excitatory events. To decide if an event is excitatory, the algorithm checks if the received event is the first to be received after a period defined in an excitatory threshold. To decide if an event is inhibitory, the algorithm needs to find the pixels that have received an event recently. The difference on the event's time stamp and the present time must be greater than the excitatory threshold, but smaller than the inhibitory threshold. A solution would be to search through the most-recent-event array and count all of the events that are within the inhibitory threshold. However, the algorithm would have to perform this search every time a new event appears. Since the camera used for this thesis has 190×180 pixels, the adapted algorithm would perform $34,200$ operations per event. This method is inefficient, but there is a way to considerably reduce the number of operations:

1. The algorithm uses counters to keep track of how many excitatory and inhibitory events are being observed by a Receptive Field at any point in time.

2. The algorithm needs to keep track of all the events it has labelled as excitatory, inhibitory or neither.

3. When an event is labelled excitatory or inhibitory, the algorithm increases the respective counter.

4. When the algorithm stops tracking an old event, the algorithm decreases its respective counter.

A good data structure to keep track of events is a queue [3]. The adapted algorithm adds a new event at the back of the queue. The oldest events are at the front. To stop tracking, or *expire*, old events, the events at the front, as long as their time stamp is beyond the inhibitory thresholds are removed. There are three queues that are needed:

excitatory queue: for all events that are the first to come, after the time specified by the threshold eth.

regular queue: for all the events that were not classified as excitatory.

inhibitory queue: When events from the previous queues expire, they are inserted into this queue. To avoid repeated events (for instance, if an event is expired from the excitatory and regular queues at the same time) events are only added to the inhibitory queue if they have the same time stamp as the most recent event, at their location.

Figure 3.3 shows how a single event transitions between the queues. Here, $\Delta_1 = current\_ts - last\_ts$ and $\Delta_2 = current\_ts - event\_ts$; eth, ith, and exp_th are the excitatory, inhibitory and the expiration thresholds, and exQ, regQ and inhQ are the excitatory, regular and inhibitory queues. The following explains what happens after three iterations (three events enter at different times).

When an event evt_1 enters, $current\_ts$ is set to evt_1's time stamp ($last\_ts$ was set to some initialization value), and Δ_1 is calculated. If it is above the excitatory threshold, the event is classified as *excitatory* and sent to exQ. Otherwise, the event is classified as *regular* and sent to regQ. Now Δ_2 is calculated using $current\_ts$ and evt_1's (the only event in either exQ or $regQ$ at this point). Since at this point both time stamps are equal, Δ_2 is zero. Regardless of which of the two queues it was placed in, it will be below the threshold and evt_1 remains in place. There are no events in $inhQ$ yet, so last_ts is updated to $current\_ts$, and the function ends.

When another event evt_2 enters, $current\_ts$ is updated, and it is treated as evt_1. Now Δ_2 is calculated using $current\_ts$ and evt_1's ts. If the event was sent to $regQ$, it is compared to exp_Th; if was sent to exQ, it is compared to eth. If the value of Δ_2 is below or equal to the its corresponding threshold, it remains in the queue. If it is above, two things may happen. If evt_1's time stamp is different than $last\_ts$ it is only expired. Otherwise, evt_1's time stamp is updated to $current\_ts$ and sent to $inhQ$. For this example, it is supposed that the event

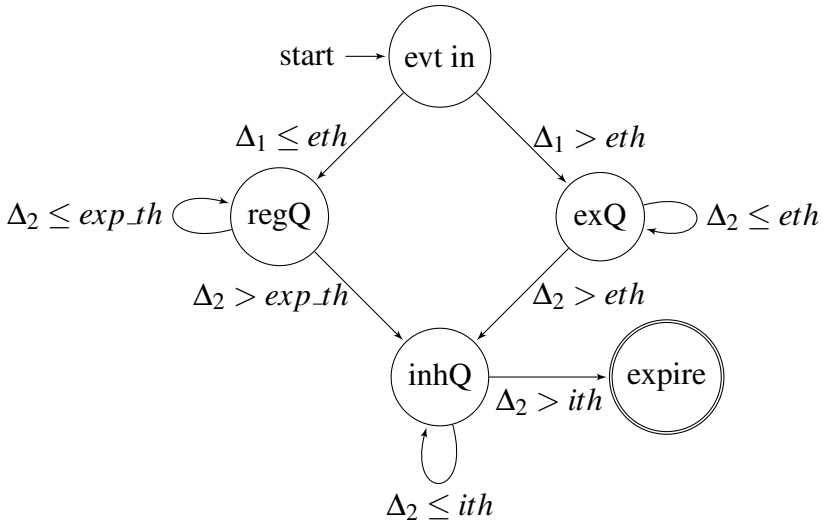


Figure 3.3: Event State Diagram

was placed in *inhQ*. Now, Δ_2 is recalculated, using *current_ts* and evt_2's (the only event in either *exQ* or *regQ* at this point), as before it will be zero, and the event remains in place. Again Δ_2 is recalculated, using *current_ts* and evt_1's (the only event in *inhQ* at this point), it is also zero and the event remains in place. Finally, last_ts is updated to *current_ts*, and the function ends.

When the third event evt_3 enters, *current_ts* is updated. The events flow in a similar way as the previous paragraph until the function reaches the inhQ again. Assume evt_2 also entered *inhQ*. Now Δ_2 is calculated using evt_1's time stamp (the event at the front of the queue). If it is below or equal to *ith* it remains in place. Otherwise, it is expired. Just as before, evt_2 would calculate a $\Delta_2 = 0$. So it remains in *inhQ*. Finally, last_ts is updated to *current_ts*, and the function ends.

3.6 Organization of Program Flow

The algorithm is implemented in an object-oriented language. There are three main classes where the algorithm is implemented. As it can be seen in Figure 3.4, the classes were implemented using the Observer Design Pattern [7], making them both subject and observer. The Receptive Field Class implements the emulated functions. It holds a count

each for excitatory and inhibitory events that gets modified through the *notify* function. When the counts are modified, the Receptive Field is triggered into making a decision. The Receptive Field is observed by any class that needs to learn about its decision—a graphical display, for instance—and it observes the Container Class.

The Container Class holds all of the Receptive Fields that cover the *image*, and it holds the three event queues explained above. Every time an event is added or expired from a queue, the Container notifies all of the Receptive Fields that cover its location. These receptive fields then update their counts and process a decision. The Container Class observes the Network Class.

The Network Class is responsible of creating and interconnecting the Container object, all of its Receptive Fields, and any observer of the latter. It then observes the camera or video input, and sends new events to the Container.

When a new event is generated in the camera, it is sent to the Network, which in turn sends it to the Container. The Container classifies the event, places it in its corresponding queue, and notifies the corresponding Receptive Fields. Then it processes the queues and further notifies the Receptive Fields. For each notification, these update their counts and compute a decision. For each decision, the Receptive Fields notify their observers.

3.7 Algorithm

This section contains the description of the adapted algorithm. It is divided into three algorithms.

Algorithm 1 describes how an event is processed in the Container class when the *notify* function is invoked. The conditional statement in lines 4 to 9 verify if the event is old enough to be considered an excitatory event. If so, the corresponding receptive fields are notified (Algorithm 2), else it is considered to be a regular event.

After classifying the input event, the algorithm proceeds to inspect the *excitatoryQueue*. Any event older than the excitatory threshold is extracted (expired), and the corresponding

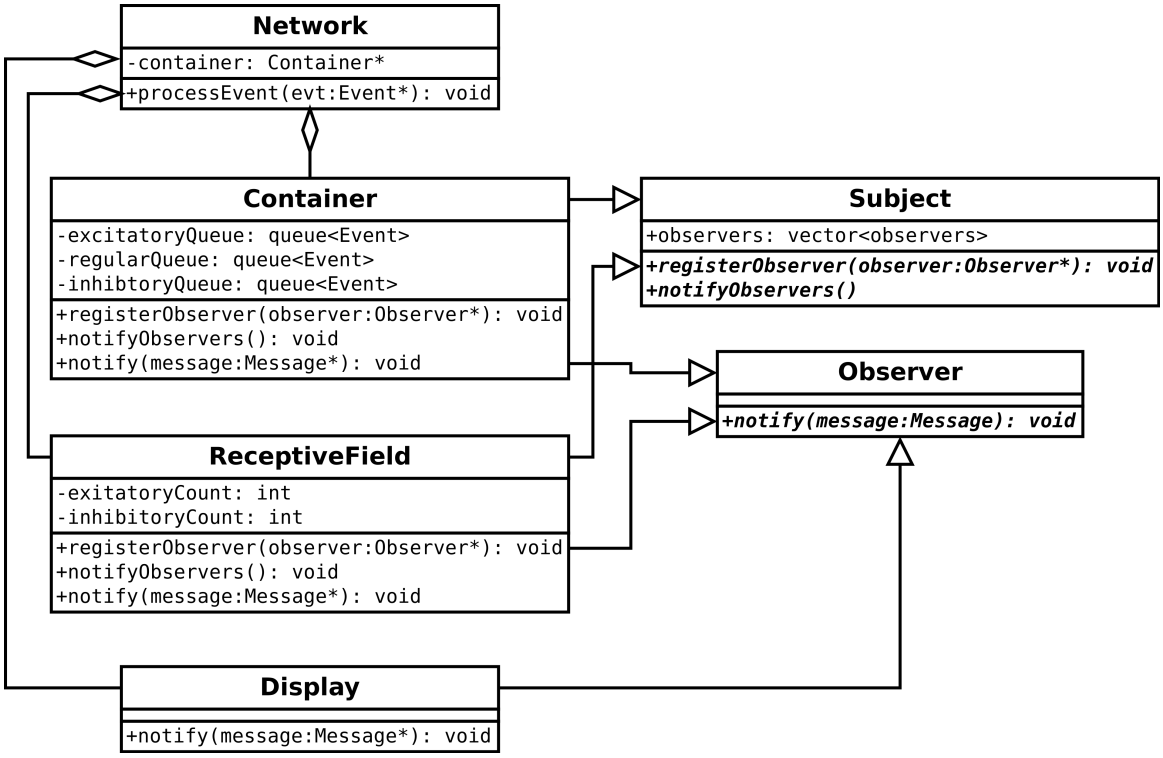


Figure 3.4: Class Diagram

receptive field is notified (Algorithm 2). As shown in line 15, if the expired event's time stamp matches the last event's time stamp, the event is added to the inhibitory queue. Also, its time stamp is updated to the current time, for it is now a *new* event, and the corresponding Receptive Field is notified (Algorithm 2).

Within lines 21 and 29 the algorithm proceeds to process the *eventQueue* in the same manner as the above, except the Receptive field is not notified (Algorithm 2) as the event is expired from the queue.

Finally, lines 30 through 34 show how the events are expired from the inhibitoryQueue. Again, the corresponding Receptive Field is notified (Algorithm 2).

Algorithm 2 describes the actions taken by a receptive field when its *notify* function is invoked.

The conditional between lines 4 and 7 is set in place so that a decision is only processed on a change of time stamp. This is meant to avoid firing several times when processing events with the same time stamp. If the event indeed contains a new time stamp, the time

Algorithm 1 An event enters a container

1: **Global Variables:** Excitatory threshold eth, Inhibitory threshold ith; Array with the last event time stamp for each location $lastEvtTs$; Queues of excitatory exQ, regular $evtQ$ and inhibitory $inhQ$
2: **Input:** Event e consisting of location x, y, time stamp ts.
3: **Output:** Notifications to observers (Receptive fields).
4: **if** $e.ts - lastEvtTs[e.y][e.x] > eth$ **then**
5: $exQ.push(e)$
6: notify observers to increase the excitatory count.
7: **else**
8: $evtQ.push(e)$
9: **end if**
10: $ts \leftarrow e.ts$
11: **while not** $exQ.empty()$ **and** $ts - exQ.front().ts > ith$ **do**
12: $exev \leftarrow exQ.front()$
13: $exQ.pop()$
14: notify observers to decrease the excitatory count.
15: **if** $exev.ts = lastEvtTs[exev.y][exev.x]$ **then**
16: notify observers to increase the inhibitory count.
17: $exev.ts \leftarrow ts$
18: $inhQ.push(exev)$
19: **end if**
20: **end while**
21: **while not** $evtQ.empty()$ **and** $ts - evtQ.front().ts > ith$ **do**
22: $ev \leftarrow evtQ.front()$
23: $evtQ.pop()$
24: **if** $ev.ts = lastEvtTs[ev.y][ev.x]$ **then**
25: notify observers to increase the inhibitory count.
26: $ev.ts \leftarrow ts$
27: $inhQ.push(ev)$
28: **end if**
29: **end while**
30: **while not** $inhQ.empty()$ **and** $ts - inhQ.front().ts > ith$ **do**
31: $Event ev \leftarrow inhQ.front()$
32: $inhQ.pop()$
33: notify observers to decrease the inhibitory count.
34: **end while**

stamp is updated. Algorithm 3 is run, and the observers (such as a display) are notified.

Lines 8 through 20 decode the input message and take the appropriate action of increasing or decreasing the excitatory or inhibitory counts.

Algorithm 3 runs the steps to make a decision in the Receptive Field, and notifies its

Algorithm 2 A Receptive field is notified

1: **Global Variables:** Time stamp of the previously received event $lastTs$; $inhibitoryCount$; $excitatoryCount$
2: **Input:** Event e consisting of location time stamp ts, event type $excitatory/inhibitory$ and action $increase/decrease$
3: **Output:** Notifications to observers (Displays or others).
4: **if** $e.ts > lastTs$ **then**
5: $lastTs \leftarrow e.ts$
6: Process the RF and notify observers.
7: **end if**
8: **if** $e.type = inhibitory$ **then**
9: **if** $e.action = increase$ **then**
10: $inhibitoryCount \leftarrow inhibitoryCount + 1$
11: **else**
12: $inhibitoryCount \leftarrow inhibitoryCount - 1$
13: **end if**
14: **else if** $e.type = excitatory$ **then**
15: **if** $e.action = increase$ **then**
16: $excitatoryCount \leftarrow excitatoryCount + 1$
17: **else**
18: $excitatoryCount \leftarrow excitatoryCount - 1$
19: **end if**
20: **end if**
21: $lastTs \leftarrow e.ts$;

observers. Lines 4 and 5 calculate the *res* value by obtaining the difference between the excitatory and inhibitory counts, apply the rectification function (same as in Section 2.3) to it.

The conditional in line 6 verifies that the time stamp of the previous *res* value is within the *res threshold*, and that both values are different. Otherwise, the decision is *NOTHING*. If both conditions are met and the current *res* is smaller than the previous, the decision is set to *RECESSING*. If the current *res* is larger, the decision is *LOOMING*.

Finally, the previous values are updated to the current, and the observers are notified.

Algorithm 3 Process Receptive Field

1: **Global Variables:** Time stamp of the previously received event $lastTs$; $inhibitoryCount$; $excitatoryCount$; Previous calculation of res $lastResVal$ and its time stamp $lastResTs$.
2: **Input:** Event e consisting of location time stamp ts, event type $excitatory/inhibitory$ and action $increase/decrease$
3: **Output:** Notifications to observers (Displays or others).
4: $x \leftarrow excitatoryCount - inhibitoryCount$
5: $res \leftarrow x * (sign(x) + 1)/2$
6: **if** $lastTs = lastResTs$ **or** $lastTs - lastResTs > resThreshold$ **or** $res = lastResVal$ **then**
7: $\quad decision \leftarrow NOTHING$
8: **else if** $res < lastResVal$ **then**
9: $\quad decision \leftarrow RECESSING$
10: **else**
11: $\quad decision \leftarrow LOOMING$
12: **end if**
13: $lastResTs \leftarrow lastTs$
14: $lastResVal \leftarrow res$
15: Notify observers about the $decision$

3.8 Analysis of complexity per event

Every time the algorithm is invoked by an incoming event, it adds the event to a queue, and starts expiring old events. A single invocation of the algorithm could take $O(n)$ time, where n is the number of old events in the queues. However, this ends up emptying the queues, and in the next invocation the algorithm would take constant time. This invocation would add only one event to the queue, hence the next invocation would also take constant time. The algorithm will do few operations most of the time, but may do a lot of operations at any one time.

To better explain the complexity analysis, amortized analysis is used instead [3]. In Algorithm 1, when an event is received, it is stored in the excitatory queue (and its receptive fields are notified using Algorithm 2) or into the regular queue. After some time, the event expires. If the event expired from the excitatory queue its receptive fields are notified using Algorithm 2. If the event's time stamp matches the one in the last_event array, it is inserted into the inhibitory queue, and its receptive field is notified using Algorithm 2. In the case

that it is an excitatory event, and the pixel generates no further events, the event will trigger the Receptive Field four times during its lifetime. In the case where the event is not excitatory and not the last event to be received, it will not trigger anything. In conclusion any event that enters the algorithm will trigger, at most, four receptive field decisions. All receptive field decisions are done in constant time. Hence, processing each event is done in constant time on average.

3.9 Experiments

3.9.1 Setup

To test the adapted algorithm, both the original frame-based algorithm and the adapted algorithm were implemented in two separate programs. The event-based camera used to test both algorithms is able to output both a stream of events and a regular video. Making it possible to compare both approaches simultaneously.

The excitatory, inhibitory and res thresholds were set to the period between frames. So all the events that generated a particular event are taken into account.

Each of the programs outputs its decisions into a file: the decision, its time stamp, the location of the receptive field that is activating, and the size of the receptive field.

A program was written to compare the output of both implementations. It compares the decisions made at a certain time between the two implementations and calculates the Precision and Recall rate. Because the frame-based implementation only generates information at precise intervals (every 50ms in a 20 fps video) and the event-based algorithm generates information every few microseconds, a threshold is used to bind the event-based decisions to the nearest decision of the frame-based approach.

Table 3.1 shows the configuration used in artificial and real captures.

Table 3.1: Configuration for tests.

	Excitatory Threshold	**Inhibitory Threshold**	**Res Threshold**	**Evaluation Threshold**
Artificial Captures	50000	50000	50000	49999
Regular Captures	33333	33333	33333	33332

3.9.2 Dataset

A set of videos was created to test the efficacy of the algorithms. There are a few videos that were created artificially. These were created using a program that simulated both each frame of the video and each event. A brief description of them is given in Appendix A. Table 3.2 shows the properties of the videos used in this chapter.

Table 3.2: Video Properties

Video	**Duration**	**Frames per second**	**Number of frames**	**Number of Events**	**Events per second**
MergingCards0	16.7s	30	502	752540	45062.3
MovingCard2	7.8s	30	234	891577	114304.7
MovingCard3	8.5s	30	254	616984	72586.4
MovingCard4	11.6s	30	348	998124	86045.2
MultipleRect0	10.0s	30	299	1664585	66458.5
MultipleRect1	14.8s	30	442	2734478	184762.0
NonConvex0	8.3s	30	249	643509	77531.2
NonConvex1	13.1s	30	394	863532	65918.5
NonConvex2	8.9s	30	266	403402	45326.1
RotatingCard0	7.9s	30	238	1000295	126619.6
RotatingCard1	8.6s	30	257	684831	79631.5
SpinningCard0	5.8s	30	173	395658	68216.9
SpinningCard1	8.0s	30	240	610202	6275.3
Testvideo	4.1s	30	122	494733	120666.6
Text0	6.7s	30	201	627565	93666.4
Text1	9.9s	30	298	1196079	120816.1
TwoCardsMoving0	7.9s	30	238	1006830	127446.8
TwoCardsMoving1	10.2s	30	308	375423	36806.2
TwoCardsMoving2	20.3s	30	609	861447	42435.8
Walking0	15.5s	30	466	4486973	289482.1

3.9.3 Evaluation Metrics

Since our goal is to emulate the frame-based algorithm, the output of the frame-based implementation is considered to be the ground truth. Precision [20] is defined as the number of decisions in the event-based approach that have a matching decision in the frame-based approach, over the total number of event-based decisions:

$$\text{precision} = \frac{\text{\# of matches}}{\text{\# event-based decisions}} \tag{3.1}$$

Recall [20] is defined as the number of decisions in the frame-based approach that have a match in the event-based approach, over the total number of frame-based decisions.

$$\text{recall} = \frac{\text{\# of matches}}{\text{\# frame-based decisions}} \tag{3.2}$$

For the decisions to match, they must come from the same receptive field, the absolute difference between their time stamps must not exceed the threshold, and they must have the same decision.

High precision means that most of the decisions made by the algorithm are correct. High recall means that most of the events that should be detected are being detected. A high precision but low recall would tell that the algorithm is making the correct decision, but it is not making many of them. A low precision but high recall would mean that the algorithm is detecting most of the events it should be detecting, but it is also making a large number of decisions it should not be making.

3.9.4 Machine

Table 3.3 shows the characteristics of the machine where the experiments were performed, and Table 3.4 shows the specification of the camera used to record the videos.

The excitatory, expiration and inhibitory thresholds were set to 50ms, and the res threshold was set to 100ms.

Table 3.3: System specification

Name	Value
CPU	AMD A12 7th Gen 2.7GHz
RAM	4 GiB

Table 3.4: The Dynamic Vision Sensor camera specifications [11]

Name	Value
Model	DVS240
Optics	CS-mount
I/O	USB2.0
Software	cAER
Power Source	USB Type B
Power Consumption	Low/high activity: 30/60 mA @ 5 VCD

3.9.5 Results

Three size and overlap configurations were tested on the dataset. As explained in Section 2.3 two or more receptive fields can cover the same region (Figure 2.3). In the case of the adapted algorithm the Receptive Fields are rectangular (Figure 3.5), and the overlap is the number of pixels, in one dimension, that one receptive field can be placed over the other.

The configurations tested were: a receptive field of 3×3 pixels with an overlap of 2 pixels, a receptive field of 89×89 pixels with an overlap of 45 pixels, and a receptive field of 179×179 pixels with an overlap of 0 pixels. These were chosen to see how the adapted

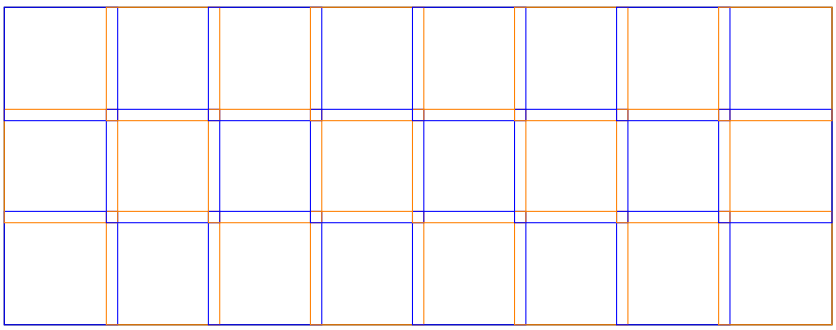


Figure 3.5: Rectangular receptive fields. Adjacent receptive fields are coloured differently for visualization.

algorithm behaved using a small receptive field, a receptive field a quarter of the image, and a receptive field covering the whole image. The latter does not have any overlap as there is nothing to gain from a second receptive field covering the whole picture, shifted one pixel. The results can be found in Tables 3.5, 3.6 and 3.7.

Table 3.5: Experiments with 3×3 receptive fields with an overlap of 2.

Video	**Total Frame-based Time**	**Total Event-based Time**	**Average Event-based Time**	**Worst Event-based Time**	**Precision**	**Recall**
MergingCards0	9.22s	1.67s	2.21μs	59.4ms	4.39%	18.10%
MovingCard2	4.33s	1.54s	1.73μs	63.3ms	12.48%	21.93%
MovingCard3	4.51s	1.31s	2.13μs	64.5ms	4.98%	15.67%
MovingCard4	6.20s	2.08s	2.08μs	116.8ms	3.95%	12.85%
MultipleRect0	5.65s	2.62s	1.57μs	116.2ms	19.59%	36.92%
MultipleRect1	8.33s	5.18s	1.89μs	222.2ms	16.94%	38.74%
NonConvex0	4.65s	1.10s	1.71μs	64.7ms	6.66%	19.10%
NonConvex1	7.54s	1.53s	1.78μs	63.5ms	7.45%	17.44%
NonConvex2	4.81s	0.90s	2.24μs	31.4ms	8.64%	25.21%
Walking0	9.18s	15.07s	3.36μs	429.4ms	25.48%	51.87%
RotatingCard0	4.58s	1.90s	1.90μs	63.9ms	8.61%	20.00%
RotatingCard1	4.70s	1.47s	2.14μs	57.7ms	4.17%	17.11%
SpinningCard0	3.05s	0.75s	1.90μs	36.8ms	9.73%	21.80%
SpinningCard1	4.33s	1.04s	1.71μs	61.1ms	3.66%	10.27%
Testvideo	2.22s	0.94s	1.89μs	31.7ms	7.03%	24.34%
Text0	3.93s	1.58s	2.51μs	67.9ms	38.04%	57.19%
Text1	5.16s	3.50s	2.93μs	117.8ms	18.98%	36.14%
TwoCardsMoving0	4.41s	1.69s	1.68μs	58.2ms	9.94%	22.90%
TwoCardsMoving1	5.45s	0.96s	2.55μs	32.1ms	5.21%	21.29%
TwoCardsMoving2	11.01s	1.46s	1.69μs	62.5ms	6.39%	15.24%
ArtificialOne	8.42s	0.33s	2.15μs	16.1ms	73.87%	69.60%
ArtificialTwo	3.54s	0.08s	1.87μs	5.1ms	72.00%	61.66%

The low precision and recall rates in Table 3.5 are mainly attributed to noise in the captured videos. Figure 3.6 shows a comparison of the output of the frame-based and event-based implementations. Figure 3.6(a) shows the output of the frame implementation, in which the offsets described in equation (2.2) and equation (2.3) are set to 20. Small

Table 3.6: Experiments with 89×89 receptive fields with an overlap of 45.

Video	**Total Frame-based Time**	**Total Event-based Time**	**Average Event-based Time**	**Worst Event-based Time**	**Precision**	**Recall**
MergingCards0	2.45s	0.47s	0.62μs	15.6ms	41.34%	90.00%
MovingCard2	1.17s	0.55s	0.62μs	16.9ms	49.51%	81.99%
MovingCard3	1.21s	0.40s	0.65μs	8.9ms	48.85%	88.46%
MovingCard4	1.72s	0.64s	0.64μs	15.8ms	34.60%	87.78%
MultipleRect0	1.49s	1.06s	0.64μs	31.5ms	37.46%	75.00%
MultipleRect1	2.15s	1.86s	0.68μs	61.4ms	41.69%	75.52%
NonConvex0	1.25s	0.40s	0.62μs	9.6ms	26.63%	79.90%
NonConvex1	1.95s	0.55s	0.64μs	16.6ms	29.43%	87.17%
NonConvex2	1.31s	0.28s	0.70μs	8.8ms	42.74%	86.56%
Walking0	2.42s	3.81s	0.85μs	59.9ms	52.78%	93.99%
RotatingCard0	1.21s	0.63s	0.63μs	17.4ms	33.04%	87.53%
RotatingCard1	1.19s	0.48s	0.70μs	9.6ms	30.10%	86.86%
SpinningCard0	0.89s	0.26s	0.65μs	9.4ms	51.87%	94.83%
SpinningCard1	1.18s	0.36s	0.58μs	8.8ms	26.95%	75.33%
Testvideo	0.58s	0.33s	0.67μs	9.6ms	33.61%	86.57%
Text0	1.03s	0.43s	0.69μs	17.8ms	57.22%	88.88%
Text1	1.48s	0.94s	0.78μs	30.2ms	43.81%	85.75%
TwoCardsMoving0	1.20s	0.63s	0.63μs	17.4ms	42.26%	78.81%
TwoCardsMoving1	1.58s	0.28s	0.75μs	9.0ms	44.22%	89.90%
TwoCardsMoving2	2.96s	0.55s	0.64μs	18.0ms	51.44%	91.15%
ArtificialOne	2.55s	0.10s	0.63μs	0.2ms	68.14%	16.33%
ArtificialTwo	1.06s	0.03s	0.64μs	0.2ms	60.24%	22.11%

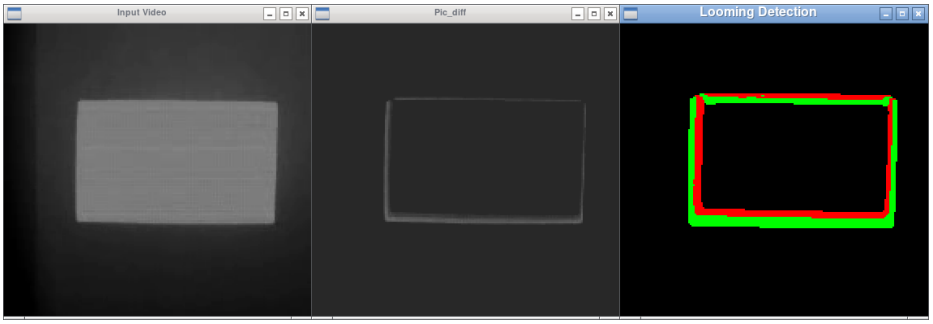
changes due to noise are effectively ignored by these offsets. As seen in Figure 3.6(c), the frame-based implementation output with the offsets set to 3 shows a considerable amount of noise. The videos used as input to the frame-based implementation are affected by noise resulting from the lossy video compression format used in storing these videos. The noise is visible in Figure 3.6(c) because the offset was lowered. On the other hand, the events recorded by the DVS used as input to the event-based implementation contain noise caused by smaller changes of log luminance if the threshold in the DVS was set too low (Figure 3.6(b)). The types and locations of noise contained in the input for the frame-based

Table 3.7: Experiments with a 179×179 receptive field with an overlap of 0.

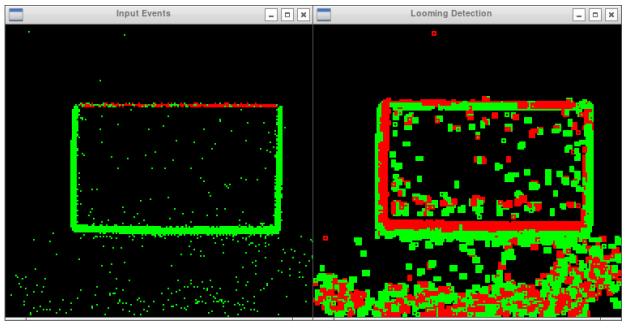
Video	**Total Frame-based Time**	**Total Event-based Time**	**Average Event-based Time**	**Worst Event-based Time**	**Precision**	**Recall**
MergingCards0	1.59s	0.39s	0.52μs	4.2ms	51.03%	92.56%
MovingCard2	0.73s	0.44s	0.49μs	4.4ms	51.73%	87.58%
MovingCard3	0.81s	0.30s	0.49μs	2.3ms	44.77%	86.67%
MovingCard4	1.14s	0.55s	0.56μs	4.8ms	44.88%	93.01%
MultipleRect0	0.95s	0.77s	0.46μs	8.7ms	37.70%	80.21%
MultipleRect1	1.41s	1.42s	0.52μs	9.7ms	42.20%	78.49%
NonConvex0	0.82s	0.34s	0.52μs	2.5ms	35.89%	85.71%
NonConvex1	1.29s	0.45s	0.53μs	4.7ms	35.69%	85.66%
NonConvex2	0.86s	0.20s	0.49μs	2.3ms	44.42%	86.29%
Walking0	1.55s	3.19s	0.71μs	17.8ms	52.55%	95.91%
RotatingCard0	0.73s	0.54s	0.54μs	4.7ms	27.83%	90.54%
RotatingCard1	0.82s	0.38s	0.55μs	4.6ms	41.54%	92.00%
SpinningCard0	0.56s	0.21s	0.53μs	2.2ms	45.03%	100.00%
SpinningCard1	0.77s	0.31s	0.51μs	2.2ms	46.97%	98.84%
Testvideo	0.40s	0.27s	0.56μs	2.5ms	24.77%	87.69%
Text0	0.67s	0.32s	0.51μs	4.3ms	59.16%	89.80%
Text1	0.96s	0.71s	0.59μs	4.5ms	47.00%	89.05%
TwoCardsMoving0	0.80s	0.51s	0.51μs	4.6ms	42.78%	78.26%
TwoCardsMoving1	0.93s	0.22s	0.58μs	2.4ms	43.50%	94.39%
TwoCardsMoving2	1.88s	0.52s	0.60μs	5.4ms	60.85%	95.75%
ArtificialOne	1.70s	0.09s	0.61μs	0.2ms	59.26%	7.34%
ArtificialTwo	0.70s	0.02s	0.52μs	0.1ms	59.26%	16.33%

and event-based are different, resulting in low precision and recall rates. In contrast, the artificial videos have higher precision and recall rates. The regular video still has some compression noise that has to be attenuated with o_d and o_l. However, the event stream has only the relevant events and no noise.

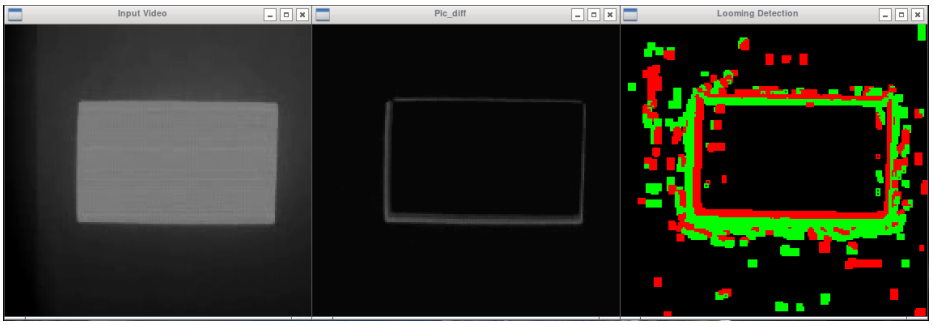
Figure 3.7 shows a visual comparison of both outputs. The configuration uses receptive fields of size 3×3 pixels and an overlap of 2 pixels. Note the similarity in the Looming Detection windows. There is an outer border of receptive fields with a looming decision followed by a trailing border of recessing decisions. This is what Fülöp and Zarándy [6]



(a) Output of the frame-based implementation setting $o_d = 20$ and $o_l = 20$. Looming receptive fields are green, while recessive receptive fields are red.



(b) Output of the event-based implementation.

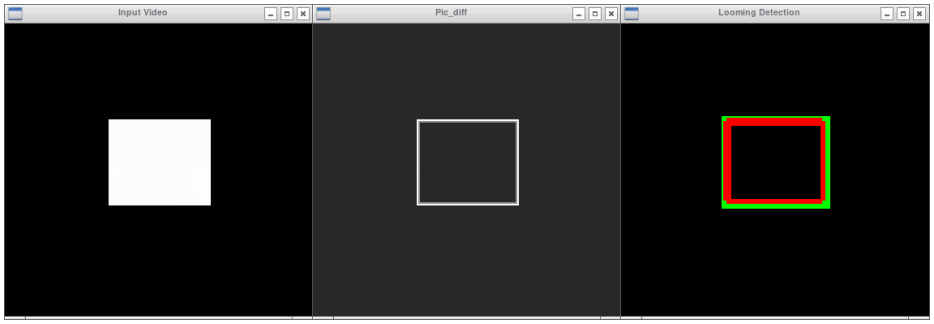


(c) Output of the frame-based implementation setting $o_d = 3$ and $o_l = 3$.

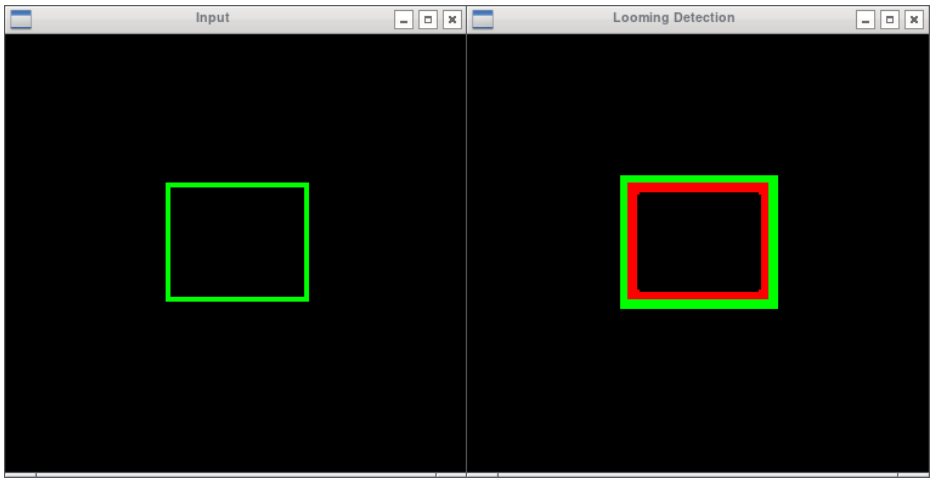
Figure 3.6: Noise in the input.

expected to see when part of a looming object moves through the receptive fields. As the object first enters there are more excitatory events and the receptive field generates a looming decision. Later, those events expire and become inhibitory events, provoking the recessing response. Although there is a low precision and recall rate for the event-based implementation, qualitatively both algorithms are yielding similar results especially if the noisy parts are ignored. In addition to noise, the precision and recall rates are low because the event-based implementation may produce the correct decision for a receptive field at a

slightly different time or location, despite the fact the output is qualitatively correct.



(a) Output of the frame-based implementation setting $o_d = 20$ and $o_l = 20$



(b) Output event implementation

Figure 3.7: Comparison using artificial videos.

This effect reverses in Tables 3.6 and 3.7. The recall of the captured videos rises above 70%, there even is a case that has a recall of 100%. Their precision is low, but significantly higher than the ones in Table 3.5. This is due to the fact that when the receptive fields are larger, isolated noise affects the receptive fields less significantly. In addition, it is less likely for a decision to be reported at an incorrect location because the receptive fields are larger. The artificial videos see a steep drop in recall. Their precision drops, but still remains above 50%. Figure 3.8 shows what happens using the ArtificialOne video with a receptive field size of 179×179 and overlap of 0. The event-based adaptation is not detecting looming or recessing events for most of the video, while the frame-based implementation is showing some output. The recall is so low because most of the time the frame-based implementa-

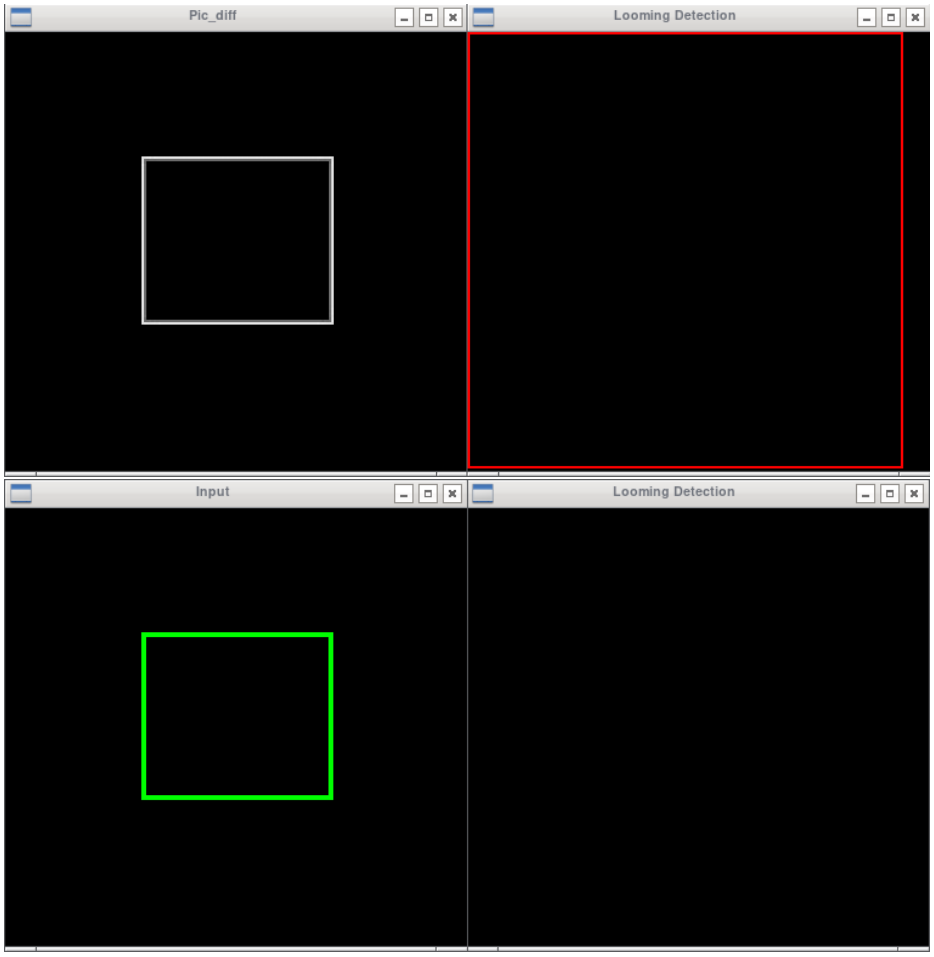


Figure 3.8: Comparison between frame-based and event-based implementations.

tion is deciding for looming or recessing while the event-based is producing a NOTHING output.

The event-based adaptation is not producing an output because the the calculated *res* values are perfectly constant. For instance, if the rectangle in the video is approaching at time t, the perimeter of excitatory events is 89×89 pixels, and the trailing silhouette of inhibitory events is 88×88 pixels. The number of excitatory events is:

$$ex_t = 89 \times 2 + (89 - 2) \times 2 = 352$$

.

The number of inhibitory events is:

$$inh_t = 88 \times 2 + (88 - 2) \times 2 = 348$$

Hence the *res* value is, as explained in Section 3.4:

$$res_t = r(352 - 348) = 4$$

At time $t + 1$, the video advances exactly one pixel to every side. The perimeter of excitatory events is 90×90, and the perimeter of inhibitory events is now 89×89. The calculations are as follows:

$$ex_{t+1} = 90 \times 2 + (90 - 2) \times 2 = 356$$

.

$$inh_{t+1} = 89 \times 2 + (89 - 2) \times 2 = 352$$

.

$$res_{t+1} = r(356 - 352) = 4$$

Both res values equal to 4, and equation (2.10) shows that if both values are exactly the same, the decision is *NOTHING*. This is not a problem in the frame-based implementation, because it does not just count events. Instead it sums the intensity values of the pixel. These intensity values have been modified by compression noise, the weights and the offsets of the different functions. The events of the artificial video are perfect, the actual video is not.

This effect is not seen in the smaller receptive fields because the boundary of the figure does not remain in the receptive field for a long time. The effect is further masked by the overlap of receptive fields.

The complexity of processing a single event was explained in Section 3.8. Because of

the algorithm's amortized time, processing an event takes constant time on average. There are two important measures in Tables 3.5, 3.6 and 3.7, Average Event Time and Worst Event Time. Average event time is simply the time it took to process the entire video divided by the number of events. Although the worst time of MergingCards0 in Table 3.5 is 54.4 milliseconds, on average each event took 2.21 microseconds (over a thousand times faster than the frame-based implementation).

Naturally, the number of events received per unit of time is also important. Notice also that Walking0 has the longest times across the different datasets. This is the only dataset in which the camera is moving. From the point of view of the camera almost everything is changing. Hence the number of input events is considerably larger than the other datasets. Table 3.2 confirms this—the number of events per second in Walking0 is the highest.

Notice that the longest times are in Table 3.5. It is to be expected since there are so many receptive fields with maximum overlap. This is clear in Table 3.7 as the times are significantly reduced.

The frame-based implementation takes a considerable amount of time to run compared to the event-based implementation. For instance, in Table 3.5, the frame-based implementation takes over 5 times longer than the event-based implementation on the MergingCards0 dataset. The only time this is reversed is in the Walking0 video, which, as explained before, contains a larger number of events every second.

3.10 Summary

In this chapter, we presented an event-based adaptation of the frame-based bio-inspired looming detection algorithm of Fülöp and Zarándy. Experiments were performed to determine how well the event-based algorithm emulates the frame-based algorithm. When the receptive fields are small, the precision and recall rates of the event-based implementation were low due to a number of factors including noise. However, the looming and recessing decisions appear to be correct qualitatively when effect noise is ignored. We also showed

that the processing time required by the event-based algorithm is significantly smaller than that required by the frame-based algorithm in most cases.

In the next chapter we will describe a new looming object detection algorithm using the event-based algorithm here as a starting point. It will be shown that the low precision and recall rates as well as the effect of noise do not affect the result of the looming detection algorithm developed in the next chapter, because the results are correct qualitatively.

Chapter 4

Looming Object Detection

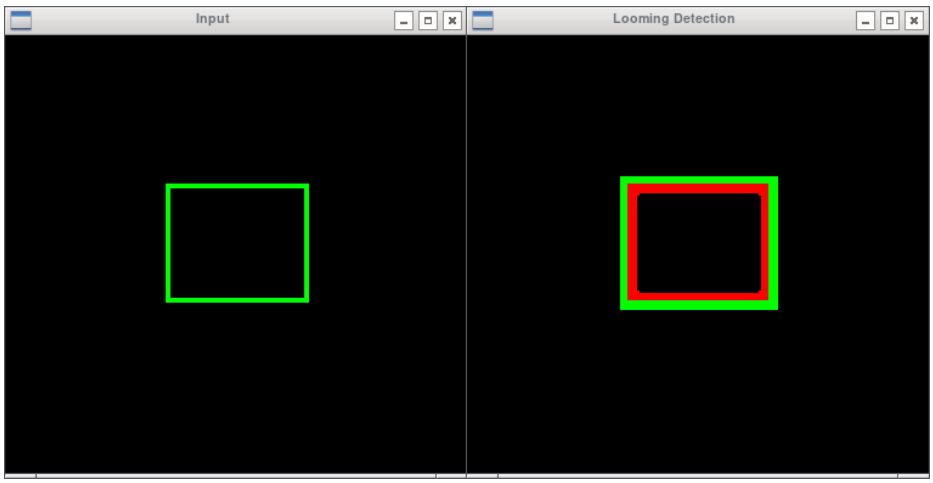
In Chapter 3, the bio-inspired looming and recessing detection algorithm for receptive fields was adapted for use with event-based cameras. While the output of the event-based algorithm is not identical to those of the frame-based algorithm, it appears to be correct qualitatively.

The bio-inspired algorithm does not give a proper answer when the size of the object is larger than the receptive field. This chapter improves the previous model to solve this issue. It is also shown that the algorithm in Chapter 3 provides output that can be used to compute the optical flow with a new approach. Moreover, the new approach will partition the image into boxes of different sizes in an attempt to find the ones that are most fitting for the objects being observed. Each separate box will use the output of the new optical flow algorithm to determine if there is an object looming in that box.

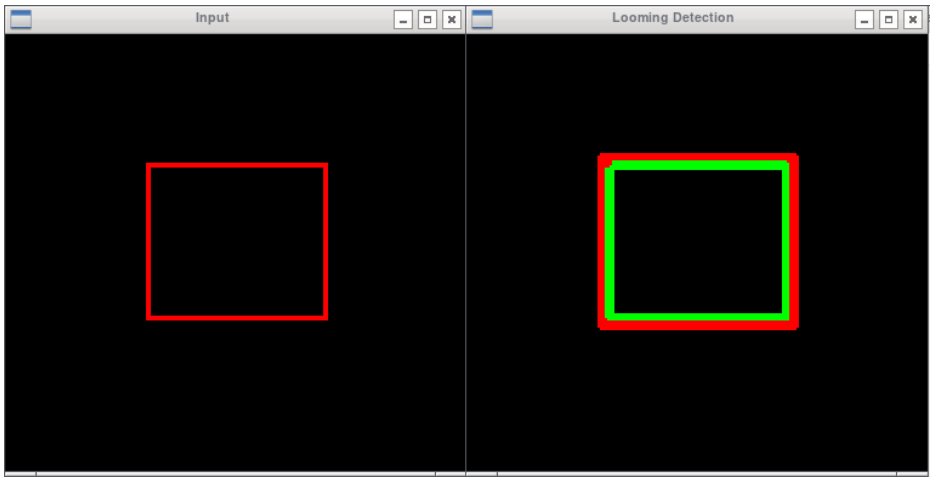
4.1 Optical Flow

Consider Figure 4.1(a) which is the same image as Figure 3.7(b). The window on the left shows the input events, and the one on the right shows the output of the looming detection algorithm of Chapter 3. The algorithm was configured to use 3×3 receptive fields with an overlap of 2 pixels. This image was taken when the rectangle in the image was looming. The receptive fields in the outer edge of the rectangle produces looming decisions while receptive fields in the inner edge produces recessing decisions. The opposite happens in the output of Figure 4.1(b), where the rectangle is recessing. As the edges of the rectangle

move, there is a leading perimeter of looming receptive fields, and a trailing perimeter of recessing fields. In short, the edges of an object move from a recessing receptive field towards a looming one. Thus, our new optical flow algorithm searches for looming receptive fields next to recessive ones.



(a) Looming rectangle.



(b) Recessing rectangle.

Figure 4.1: Artificial video of a looming and recessing rectangle. Looming receptive fields are coloured green while recessing receptive fields are coloured red.

Algorithm 4 shows how to calculate the optical flow at a pixel when a decision from Algorithm 3 is made. The *for* statement in line 6 searches the 8-neighbourhood [9] around the input (X,Y) for a recent decision which is opposite to the one received. Notice the search order gives priority to the coordinates above, below, left, and right of (X,Y), for

these are the closest. The *if* condition at line 7 ensures (x,y) are within the limits of the image, and that the last decision at that location happened within the optical flow threshold. Also notice that the directions in lines 9 and 11 ensure that the optical flow vector points from a recessing event to a looming event. Finally, if no suitable adjacent decision is found, the program returns $(0,0)$ to indicate no motion was found.

Algorithm 4 Optical Flow Algorithm

1: **Global Variables:** Optical flow threshold oth, Array with the last decision received at location x,y $lastDes$, Array with the time stamp of the last decision received at location x,y $lastDesTs$.
2: **Input:** X coordinate, Y coordinate, Time stamp ts, Decision des.
3: **Output:** Direction of motion, in rectangular representation (x,y).
4: $rs \leftarrow$ number of rows in lastDes.
5: $cs \leftarrow$ number of columns in lastDes.
6: **for all** $(x,y) \in \{(X-1,Y),(X+1,Y),(X,Y-1),(X,Y+1),(X-1,Y-1),(X+1,Y-1),(X+1,Y-1),(X+1,Y+1)\}$ **do**
7: **if** $0 \leq x < cs$ **and** $0 \leq y < rs$ **and** $ts - lastDesTs[y][x] \leq oth$ **then**
8: **if** $des = LOOMING$ **and** $lastDes[y][x] = RECESSING$ **then**
9: **return** (X-x,Y-y)
10: **else if** $des = RECESSING$ **and** $lastDes[y][x] = LOOMING$ **then**
11: **return** (x-X,y-Y)
12: **end if**
13: **end if**
14: **end for**
15: **return** (0,0)

4.2 Analyzing Optical Flow Events

As explained in Section 2.5, Ridwan's algorithm computes the optical flow of the input DVS events. It then computes the centroid of the optical events as the centre of the object. The dot product of the optical flow vector and the vector from the centre to the optical flow event is computed to determine see if the motion is pointing away from the centre. If enough vectors point away from the centre, the object is considered to be looming.

Our new approach does not compute the centre of the observed object. Instead, the centre of the image is assumed to be an interior point of the object. Using the dot product,

we compute the number of optical flow events pointing towards and away of the centre. If there are more vectors pointing away from the centre than vectors pointing towards it, the object is looming. To verify that the centre is enclosed by the object, the algorithm ensures that at least one of the optical flow events is in each of the four quadrants of the image, representing the areas above-left, above-right, below-left, and below-right of the centre.

The thresholds used in the algorithms are listed as follows:

oth: this is the threshold for expiring old *LOOMING/RECESSING* events as well as for expiring the computed optical flow events.

dth: it is a real number between 0 and 1. It represents the minimum ratio of optical flow events that have to point away from the centre of the optical flow field for it to be considered looming.

Algorithm 5 shows what happens when a new optical flow event is received from a receptive field. This function is in charge of keeping track of the latest event from each receptive field. This can be observed in lines 10 and 11. Lines 12 to 26 use Algorithm 4 to compute the optical flow of the received event, and if appropriate adds it to the optical flow event queue (*ofeq*). Two things happen after a new optical flow event is placed in the queue—the quadrant where the event is located is identified, and the dot product between two vectors is calculated. If the dot product is positive, the away count is increased. If it is negative the toward count is increased. Line 27 shows how the algorithm expires old optical flow events from *ofeq*. When an optical flow event is expired the appropriate count for its quadrant is decreased as well as the appropriate count for its direction (away or toward). As in Algorithm 2, observers are only notified when the time stamp changes, as it is shown in line 4. This avoids reacting multiple times to events with the same time stamp. This algorithm has a constant time complexity for each event on average, using an amortized analysis similar to what was done in Section 3.8. This approach would not output the correct decision if the centre of the picture is not inside the object, or if there are multiple objects.

Algorithm 5 Analysis of Optical Flow Events

Global Variables: Counters *away* and $toward$, Optical flow threshold oth, Time stamp of the last decision received $lastTs$, Queue of optical flow events $ofeq$, Array with the last decision received at location x,y $lastDes$, Array with the time stamp of the last decision received at location x,y $lastDesTs$.
Input: Event *evt* consisting of location x, y, time stamp ts, and polarity pol.
Output: Notifications to observers (Display or others).
if $evt.ts > lastTs$ **then**
    $lastTs \leftarrow evt.ts$
    notifyObservers()
end if
$cx \leftarrow$ number of rows in lastDes divided by two.
$cy \leftarrow$ number of columns in lastDes divided by two.
$lastDes[evt.y][evt.x] \leftarrow evt.pol$
$lastDesTs[evt.y][evt.x] \leftarrow evt.ts$
(dx,dy) $\leftarrow$ computeDirection(evt.x,evt.y, evt.ts, evt.pol)
if not (dx = 0 **and** dy = 0) **then**
    ofevent $\leftarrow$ (evt.x,evt.y,dx,dy,evt.ts)
    ofeq.push(ofevent)
    $aboveLeft \leftarrow aboveLeft + 1$ **if**($ofevent.x < cx$ **and** $ofevent.y < cy$)
    $aboveRight \leftarrow aboveRight + 1$ **if**($ofevent.x > cx$ **and** $ofevent.y < cy$)
    $belowLeft \leftarrow belowLeft + 1$ **if**($ofevent.x < cx$ **and** $ofevent.y > cy$)
    $belowRight \leftarrow belowRight + 1$ **if**($ofevent.x > cx$ **and** $ofevent.y > cy$)
    $dot \leftarrow (ofevent.x - cx) \times ofevent.dx + (ofevent.y - cy) \times ofevent.dy$
    if $dot > 0$ **then**
        $away \leftarrow away + 1$
    else if $dot < 0$ **then**
        $toward \leftarrow toward + 1$
    end if
end if
while not ofeq.empty() **and** $evt.ts - ofeq.front().ts > oth$ **do**
    $ofevent \leftarrow ofeq.front()$
    $aboveLeft \leftarrow aboveLeft - -$ **if**($ofevent.x < cx$ **and** $ofevent.y < cy$)
    $aboveRight \leftarrow aboveRight - -$ **if**($ofevent.x > cx$ **and** $ofevent.y < cy$)
    $belowLeft \leftarrow belowLeft - -$ **if**($ofevent.x < cx$ **and** $ofevent.y > cy$)
    $belowRight \leftarrow belowRight - -$ **if**($ofevent.x > cx$ **and** $ofevent.y > cy$)
    $dot \leftarrow (ofevent.x - cx) \times ofevent.dx + (ofevent.y - cy) \times ofevent.dy$
    if $dot > 0$ **then**
        $away \leftarrow away - 1$
    else if $dot < 0$ **then**
        $toward \leftarrow toward - 1$
    end if
    $ofeq.pop()$
end while
$lastTs \leftarrow evt.ts$

Algorithm 6 Notify Observers

1: **Global Variables:** Queue of optical flow events $ofeq$, Array with the last decision received at location x,y $lastDes$, Array with the time stamp of the last decision received at location x,y $lastDesTs$.
2: **Input:** None.
3: **Output:** Notifications to observers (Displays or others).
4: $surrounded \leftarrow aboveLeft > 0$ **and** $aboveRight > 0$ **and** $belowLeft > 0$ **and** $belowRight > 0$
5: $diff \leftarrow away - toward$
6: **if** $|diff| < ofeq.size \times dth$ **then**
7: $des \leftarrow NOTHING$
8: **else if** $away > toward$ **and** $surrounded$ **then**
9: $des \leftarrow LOOMING$
10: **else**
11: $des \leftarrow NOTHING$
12: **end if**
13: Notify observers about the des

Algorithm 6 takes the queue of optical flow events from Algorithm 5 and implements the new looming detection approach. Line 4 sets that the centre is *surrounded* if an optical flow event was found in each of the four quadrants. Since there could be a case of lateral movement where the away count and the toward count would be very close to each other, the conditional in line 6 makes sure that the absolute difference between them is higher than a configurable ratio. Line 8 tests for the looming case. Since this algorithm is meant to detect looming objects, there is no test for recessing. Once a decision is reached, the algorithm terminates by notifying the observers.

As in Section 3.5, it is necessary to keep track of the *decision events* that enter the observed area of the picture. Each field has an array that keeps track of the most recent decision and the time stamp at *(x,y)*. This structure works best since there is no need to count the number of received events at any particular time, but it is necessary to have random access to any of the latest events.

The opposite is true for the optical flow vectors. There is no need to access a particular vector randomly, but it is necessary to count how many recent vectors point toward the centre. Since not every decision in the array is recent enough or relevant, a queue is used

to store only the recent vectors. Again, the complexity of the overall algorithm is constant time on average for each event.

4.3 Partitioning Into Optical Flow Fields

Since the centre of the observed image is used as the centroid of the object, this method of looming detection would only be capable of giving a correct answer if the object's perimeter encloses the centre of the image. The approach will fail if this assumption is false, or if there are multiple objects in the scene.

To handle these issues, we developed a new approach which divides the image into sections of different sizes and locations, in the hope that one or more of these sections will contain an object containing the centre of the section. We call these sections optical flow fields. Just as an array of overlapping receptive fields is used in Chapter 3, the new approach uses an array of overlapping optical flow fields.

The new algorithm uses different grids of overlapping optical flow fields. There can be grids of optical flow fields of various sizes so that objects of different sizes can be detected. The appropriate optical flow field can then respond using Algorithms 5 and 6. The average complexity of the resulting algorithm per event is constant for each optical flow field, regardless of the size of the field. Thus, the average complexity per event is linear in the number of optical flow fields.

Since the new approach uses a looming detector for each optical flow field in the grid, it is important to filter out erroneous decisions from fields that do not really contain the entire object. For instance, Figure 4.2 shows four optical flow fields (in yellow) of 89×89 pixels each without any overlap. The looming and recessing decisions of the receptive fields from Algorithm 3 result in most of the optical flow vectors pointing towards the centre of the four optical flow fields. Hence the four fields will not give a looming response. This would be wrong, since none of them encompasses the complete object.

To mitigate this problem, the algorithm verifies that there is at least one vector in each

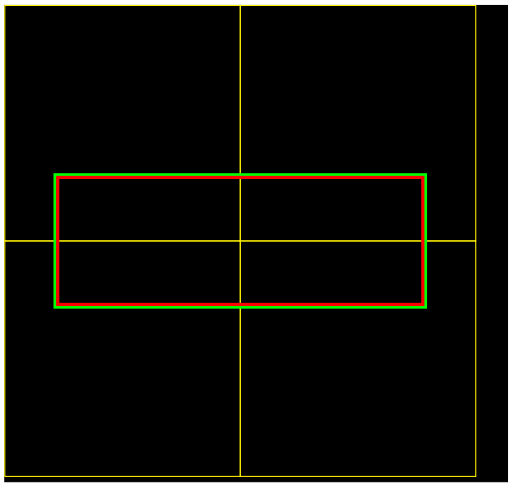


Figure 4.2: Smaller fields.

of the four quadrants of the optical flow field as a way to confirm that the centre of the optical flow field is within the object. If one of the quadrants is missing, the field reaches a decision of *NOTHING*. Notice that this is a fast but inaccurate test, as there may be cases in which the centre is enclosed inside an object and yet not all four quadrants will contain an optical flow event (e.g. a triangular object). As a result, this verification sacrifices recall to favour precision. Since it only needs to have a single optical flow event in each quadrant, it is susceptible to artifacts in the quadrant.

4.4 Experiments

4.4.1 Datasets

Because the focus of this algorithm is to detect looming motion, the tests were run against the datasets described in Appendix A that contained looming objects.

Table 4.1 contains the properties of the artificial and DVS datasets that were not described in Chapter 3.

4.4.2 Setup

The display class used by the adapted algorithm creates pseudo-frames to show each of the events—events are collected at regular intervals for display purposes. The algorithm is

Table 4.1: Number of events in dataset.

Dataset	Number of Events
A01	23744
A02	26410
A03	16818
A13	19300
B09	176565
B12	79850

fully event-based and does not collect events into frames to obtain a decision.

The DVS outputs a stream of events as well as a regular video. Taking advantage of this, the capture period for each of the frames was taken from the frame-rate of the output videos of the event-based camera, 30 fps, and in the frame-rate of the artificial videos, 20 fps. All the thresholds were set to $\frac{1}{frame-rate}$ (the period between two frames) of the output video.

Since the new algorithm is not based on Fülöp and Zarándy's algorithm, it was not possible to use the frame-based implementation to automatically generate a ground truth for calculating the precision and recall. The ground truth in this case was created by visually analyzing each pseudo-frame and labelling them Looming, or Not Looming (Lateral/Recessing/Nothing). The videos contain certain motions that are hard to catalogue by human, such as small lateral motions that could also be either looming or recessing. These cases were removed in the precision and recall calculations.

Table 4.2 shows the parameter settings for each of the tests. In addition to the thresholds oth and dth, the other constants are:

size: each optical field contains size $\times$ size receptive fields (instead of actual pixels).

overlap: it is the amount of overlap between optical fields measured in number of receptive fields.

Notice B09 appears twice because two different values of dth were tested.

Table 4.2: Configuration for looming tests.

	Receptive Fields					Optical Flow			
Dataset	**eth**	**ith**	**rth**	**size**	**overlap**	**oth**	**dth**	**size**	**overlap**
A01	50000	50000	50000	3	2	50000	30	100	22
A02	50000	50000	50000	3	2	50000	30	100	22
A03	50000	50000	50000	3	2	50000	30	100	22
A13	50000	50000	50000	3	2	50000	30	100	61
ArtificialOne	50000	50000	50000	3	2	50000	30	178	0
ArtificialTwo	50000	50000	50000	3	2	50000	0	89	80
B09	33333	33333	33333	3	2	33333	30	147	120
B09_2	33333	33333	33333	3	2	33333	20	147	120
B12	33333	33333	33333	3	2	33333	20	178	0
MovingCard2	33333	33333	33333	3	2	33333	30	178	0
MovingCard3	33333	33333	33333	3	2	33333	30	178	0
NonConvex2	33333	33333	33333	3	2	33333	30	178	0

Table 4.3: Configuration for non-looming tests.

	Receptive Fields					Optical Flow			
Dataset	**eth**	**ith**	**rth**	**size**	**overlap**	**oth**	**dth**	**size**	**overlap**
ArtificialOne	50000	50000	50000	3	2	50000	30	89	0
ArtificialTwo	50000	50000	50000	3	2	50000	0	178	0
B09_2	33333	33333	33333	3	2	33333	20	147	120
MovingCard3	33333	33333	33333	3	2	33333	30	89	0

A smaller set of tests was run to compute the recall of non-looming motion. This means that when it is certain that there is no complete looming object in the area enclosed by the optical flow field, there should not be any looming decision. Table 4.3 shows the configurations for the videos.

4.4.3 Evaluation Metric

Equations (4.1) and (4.2) show how precision and recall are calculated. Precision is the ratio of all the detected looming pseudo-frames over all detected looming pseudo-frames. Recall is the ratio of all detected looming pseudo-frames over all the frames that should have been detected as looming.

$$\text{precision} = \frac{\text{\# of True Looming}}{\text{\# True Looming} + \text{\# False Looming}} \tag{4.1}$$

$$\text{recall} = \frac{\text{\# of True Looming}}{\text{\# True Looming} + \text{\# False Not Looming}} \tag{4.2}$$

Equation (4.3) shows how to obtain the recall of non-looming motion. It used in the fields that should not react, for they do not contain or fit the objects in the scene. It is the ratio of times the field should not have reacted, against all of the decisions:

$$\text{recall} = \frac{\text{\# of True Non-Looming}}{\text{\# True Non-Looming} + \text{\# False Looming}} \tag{4.3}$$

4.4.4 Results

Table 4.4 shows the execution times of the tests. Notice that most of the average event processing times are below 4 microseconds. In most cases the worst event processing time is below 3 milliseconds. Naturally, the number of optical flow fields affects the processing time. In the case of ArtificialTwo, there are more events and there are more optical flow fields to process. The average event processing time is comparable to Ridwan's algorithm, although our approach allows for multiple looming objects to be detected as well.

To test how the algorithm's processing times change with different optical flow field size and overlap, different configurations were tested against B09_2. Table 4.5 shows the results of the different tests. Note that though there are more optical flow fields covering the image in the 3×3 configuration, the 45×45 is has the longest processing time. The number of optical flow fields is not slowing the processing time but the number of fields covering each event is.

Table 4.6 shows the precision and recall for each of the experiments. In the A01 dataset the top right object is moving sideways. There is only one pseudo-frame where the circle appears to approach and the algorithm detected it, generating a recall of 100%. It also detected a looming motion when there was none to detect, so the precision rate is only

Table 4.4: Execution times

Dataset	**Total time**	**Average Event Processing Time**	**Worst Event Processing Time**
A01	0.06s	2.66μs	1.6ms
A02	0.07s	2.59μs	2.0ms
A03	0.04s	2.17μs	1.6ms
A13	0.10s	4.93μs	2.2ms
ArtificialOne	0.49s	3.15μs	1.9ms
ArtificialTwo	1.23s	29.85μs	11.3ms
B09_2	0.93s	5.25μs	2.3ms
B09	0.93s	5.27μs	1.8ms
B12	0.60s	7.48μs	8.7ms
MovingCard2	2.02s	2.27μs	2.3ms
MovingCard3	1.73s	2.80μs	1.6ms
NonConvex2	1.28s	3.18μs	1.9ms

50%. For the bottom left object, the algorithm was able to detect all the looming motions. It reported looming for one pseudo-frame where the object was not looming, so the precision was not 100%.

In the A02 dataset, the algorithm detected both objects with equal precision and recall. There was only one instance in which it detected looming motion from both objects when there was no looming motion.

In the A03 dataset, the algorithm was again able to detect all of the looming motions of the top right object, and detected one false positive. For the bottom object it failed to detect one of the looming motions, but it did not give any false positives.

In the A13 dataset, the two squares presented no looming motion, and the algorithm did not detect any false positives. Precision and Recall are undefined because the denominators in (4.1) and (4.2) are zero. This means that the recall is undefined because there was nothing to detect, but the precision is undefined because there were no false positives and no true positives. The algorithm detected every looming motion of the centre object and had one false positive.

Table 4.5: Time tests B09_2 with different size and overlap.

Size	**Overlap**	**Number of Optical Flow Fields**	**Total time**	**Average Event Processing Time**	**Worst Event Processing Time**
178	0	1	0.58s	3.27μs	1.4ms
89	0	4	0.50s	2.85μs	1.4ms
89	80	120	8.34s	47.23μs	7.7ms
45	0	12	0.46s	2.63μs	1.3ms
45	40	783	17.00s	96.26μs	16.9ms
25	0	49	0.52s	2.97μs	1.3ms
25	20	1023	6.22s	35.25μs	8.5ms
13	0	182	0.59s	3.33μs	1.7ms
13	10	3304	5.31s	30.07μs	7.9ms
3	2	32736	3.96s	22.41μs	8.1ms

Table 4.6: Test results.

Dataset	**Precision**	**Recall**
A01(Top Right)	50.0%	100.0%
A01(Bottom Left)	93.8%	100.0%
A02 (Top Right)	91.7%	100.0%
A02 (Bottom Left)	91.7%	100.0%
A03 (Top Right)	90.9%	100.0%
A03 (Bottom Left)	100.0%	75.0%
A13 (Top Right)	undefined	undefined
A13 (Centre)	93.8%	100.0%
A13 (Bottom Left)	undefined	undefined
ArtificialOne	100.0%	100.0%
ArtificialTwo(Left)	100.0%	100.0%
ArtificialTwo(Right)	90.0%	97.3%
B09	89.3%	80.6%
B09_2	88.6%	96.9%
B12	95.0%	95.0%
MovingCard2	0.0%	undefined
MovingCard3	94.4%	86.4%
NonConvex2	83.6%	49.0%

ArtificialOne showed a precision and recall of 100%. The left object in ArtificialTwo has the same result. The right object in ArtificialTwo had one false negative and 4 false

positives.

Most of the artificial videos had high precision and recall. In most cases the false positives and false negatives happened when the object reached its minimum and maximum sizes during the looming or recessing motion. These were the times when the object just starts or just stops the looming motion.

Both B09 and B12 showed high precision and recall, even when the input datasets have a large mount of noise. Two configurations were used in B09. Reducing the dth ratio from 30% to 20% reduced precision by less than 1%, but it increased the recall by more than 16%.

MovingCard2 had no looming motion, it is a similar case to A01 and A13. There was nothing to detect for recall, but there were 29 false positives. Given that the stream is 7.8 seconds long, and the pseudo-frames are being generated at 30 fps, the false positives represent 12.4% of all the decisions presented.

MovingCard3 had looming motion, and continued to show that the algorithm works well even in noisy conditions.

NonConvex2 had an acceptable precision though a low recall. Concave objects are still challenging to detect by analyzing the optical flow directions against the centre of the optical flow field.

Experiments were also run to evaluate the accuracy of our algorithm in not producing looming decisions when there is no looming object in an optical flow field. Table 4.7 shows the recall for non-looming motion. Results in ArtificialOne show that even though there is a part of the rectangle in each of the optical flow fields, the fields rarely detect looming motions. The experiments show that the only moment they detect it is when the border crosses the centre of the field. At that moment alone are there optical flow events in every quadrant of the field.

Results in ArtificialTwo show that a large field is able to ignore the smaller objects. Though these are encapsulated by the field, their perimeters do not surround the centre of

Table 4.7: Non-looming motion recall.

Dataset	**Recall**
ArtificialOne(top-left)	96.0%
ArtificialOne(top-right)	96.0%
ArtificialOne(bottom-left)	100.0%
ArtificialOne(bottom-right)	98.0%
ArtificialTwo	100.0%
B09_2(top-left)	100.0%
B09_2(top-right)	100.0%
B09_2(bottom-left)	100.0%
B09_2(bottom-right)	100.0%
MovingCard3(top-left)	95.7%
MovingCard3(top-right)	91.1%
MovingCard3(bottom-left)	83.2%
MovingCard3(bottom-right)	81.3%

the field.

In B09_2 the configuration was left as it was in the previous tests. On running this test again it was noticed that what originally looked like false positives was just when the video was initializing or when the object was changing from looming to recessing. Hence they were ignored and the non-looming recall resulted in 100%. The test shows how even with a large amount of noise in the image, the algorithm still ignores non-looming motion. Figure 4.3 shows three moments of the dataset. The first shows the rectangle looming towards the camera, the other two show the recessing motion. Each row of screens contains the output of the camera, the output of the algorithm in Chapter 3, and the output of the looming algorithm (the green squares indicate looming detection in that area). Notice the amount of noise present in the output of the camera, and the output of the first algorithm. When the figure is recessing the algorithm does not produce a looming decision.

MovingCard3 is similar to ArtificialOne. The recall is lower in the bottom fields because the object is not perfectly centred, and there are moments when it fits in the height of the bottom fields. When this happens the four quadrants of each field actually have an optical flow event.

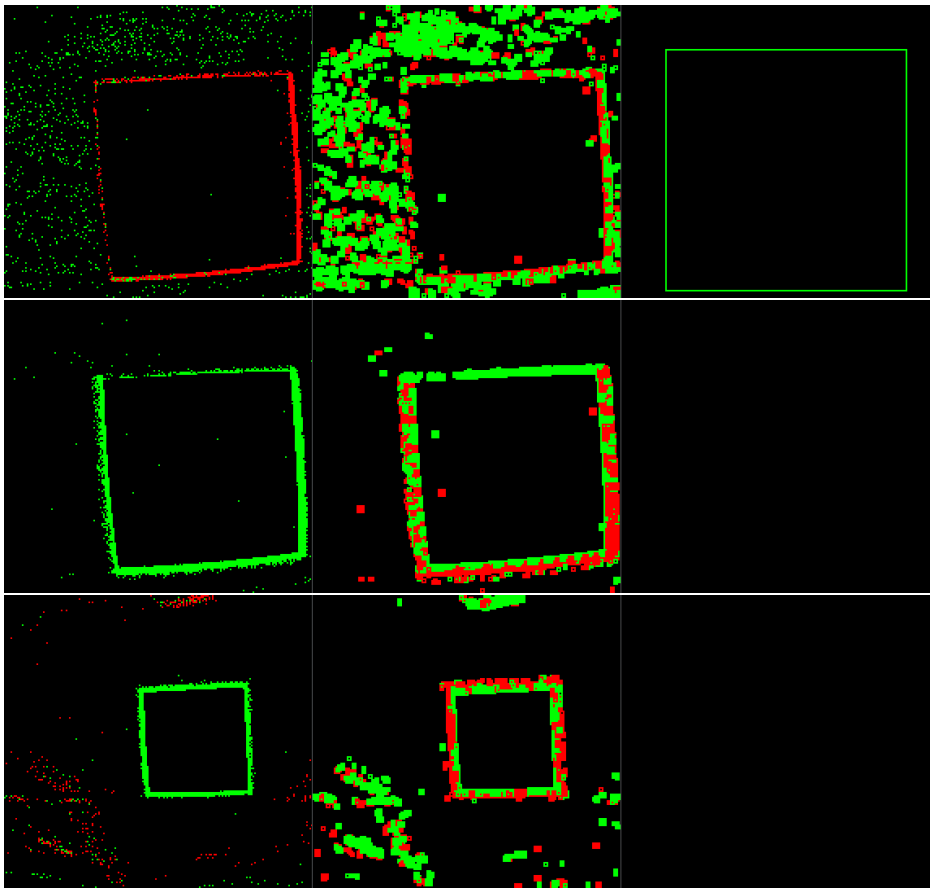


Figure 4.3: Three moments in the B09 dataset. In the top row, the object is looming. The object is recessing in the second and third row. In each row, the image on the left shows the DVS events and the middle image shows the output of the algorithm in Chapter 3. The image on the right contains a green square if the corresponding optical flow field detected looming.

4.5 Summary

This chapter presents a novel approach to looming object detection. It uses a similar algorithm to compute the optical flow of the image as Ridwan's algorithm, but differs by using our adapted algorithm from the works of Fülöp and Zarándy to compute optical flow. Our approach is tolerant to noise. Our approach also does not attempt to compute the centre of the objects. Instead it attempts to find the most appropriate optical flow field enclosing the object. Our new algorithm is capable of detecting multiple looming objects of different sizes in different parts of the scene, and has run time similar to Ridwan's single looming object detection algorithm. However, the algorithm have the same limitation as Ridwan's algorithm for detecting looming concave objects.

Chapter 5

Conclusion

In this thesis we have presented a novel approach to detect looming objects in an event-based camera. This algorithm is adapted from Fülöp and Zarándy's work with frame-based cameras. We have run both quantitative and qualitative experiments to validate our approach. The result of our event-based adaptation is visually similar to the result from the frame-based algorithm. Also, the run time is significantly shorter.

We have also designed an optical flow algorithm using the output from the event-based adaptation of Fülöp and Zarándy's algorithm. Our approach differs from the previous algorithm of Ridwan and Cheng in that our optical flow algorithm filters out noise using Fülöp and Zarándy's algorithm. Finally, we designed a looming object detector by partitioning the scene into optical flow fields of different sizes and applying a single looming object detector in each field. Our looming detector does not attempt to compute the centroid of the perceived objects. Instead it attempts to encapsulate them in the appropriate field. We have tested out approach on both artificial and captured datasets, including some that are noisy, and received positive results. We have been able to successfully detect multiple looming objects and ignore other motions. The run time of our algorithm is similar to Ridwan's looming detection algorithm despite the fact that the new algorithm can detect multiple looming objects.

5.1 Future Research Directions

In this section, we outline some possible future research directions to address some of the limitations of our proposed algorithms.

Automatic thresholds: all of our temporal thresholds have to be manually set. A system that changes these thresholds as the video progresses is yet to be devised.

Automatic optical field size and position: the algorithm presented in Chapter 4 relies on having the correct optical flow field at the right position. Several field sizes can be set at the same time, but as the number increases so does the time it takes to process an event. It may be possible to choose the correct size and position of these fields, especially if the algorithm takes into account the sizes and positions of detected objects in the past.

Better out-of-bounds detection: to decide if an object is properly enclosed by an optical flow field in our algorithm, we verify if there is at least one optical flow event in each of the four quadrants of the field. We could improve on this by requiring that an even amount of optical flow events is distributed in the four quadrants. This can improve precision rates by filtering out the results from more optical flow fields. However it would potentially reduce the rate of recall as well.

Concave objects: an improved version of the approach in Chapter 3 could potentially detect looming concave objects better than the current implementation. The approach in Chapter 4 still has the same limitation as Ridwan's algorithm.

Bibliography

[1] Full specifications and features dsc-rx10m2. https://www.sony.com/electronics/cyber-shot-compact-cameras/dsc-rx10m2/specifications. Accessed: 2019-11-06.

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

[3] T. H. Cormen, C. Leiserson, R. Rivest, and C. Stein. *Introduction to Algorithms*, chapter 10, pages 201–203. McGraw-Hill, 2nd edition, 2001.

[4] 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.

[5] T. Fülöp and Á. Zarándy. Bio-inspired looming object detector algorithm on the Eye-RIS focal plane-processor system. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2010 12th International Workshop on*, pages 1–5. IEEE, 2010.

[6] T. Fülöp and Á. Zarándy. Bio-inspired looming direction detection method. In *Cellular Nanoscale Networks and Their Applications (CNNA), 2012 13th International Workshop on*, pages 1–6. IEEE, 2012.

[7] E. Gamma, R. Helm, R. Johnson, and J. Vlissides. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 75 Arlington Street Suite 300 Boston MA 02116 USA, 1 edition, 1994.

[8] P. Gil-Jiménez, H. Gómez-Moreno, R. 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.

[9] R. C. Gonzalez and R. E. Woods. *Digital Image Processing*, chapter 4. Pearson Prentice Hall, Upper Saddle River, New Jersey 07458, 3 edition, 2008.

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

[11] iniVation. Davis 240 specifications. https://inivation.com/wp-content/uploads/2019/08/DVS240.pdf. Accessed: 2019-11-04.

[12] iniVation. Dynamic vision sensor. https://inivation.com/dvs/. Accessed: 2019-11-04.

[13] User guide: jaer. https://inivation.com/support4/software/jaer/. Accessed: 2019-11-04.

[14] libcaer. https://gitlab.com/inivation/libcaer. Accessed: 2019-11-04.

[15] User guide: libcaer. https://inivation.com/support4/software/libcaer/. Accessed: 2019-11-04.

[16] 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.

[17] L. Panahi and V. Ghods. Human fall detection using machine vision techniques on RGB–D images. *Biomedical Signal Processing and Control*, 44:146–153, 2018.

[18] 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*, pages 439–444, 2010.

[19] 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.

[20] D. M. Powers. Evaluation: from precision, recall and F-measure to ROC, informedness, markedness and correlation. *Journal of Machine Learning Technologies*, 2(1):37–63, 2011.

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

[22] I. Ridwan. Looming object detection with event-based cameras. Master's thesis, University of Lethbridge (Canada), 2017.

[23] I. Ridwan and H. Cheng. An event-based optical flow algorithm for dynamic vision sensors. In *International Conference Image Analysis and Recognition*, pages 182–189. Springer, 2017.

[24] F. C. Rind and P. J. Simmons. Seeing what is coming: building collision-sensitive neurones. *Trends in neurosciences*, 22(5):215–220, 1999.

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

[26] 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.

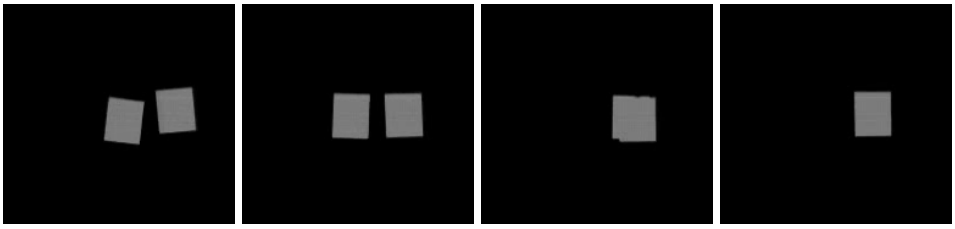
[27] B. Webb. Robots in invertebrate neuroscience. *Nature*, 417(6886):359–363, 2002.

[28] Á. Zarándy and T. Fülöp. Approaching object detector mouse retina circuit model analysis and implementation on cellular sensor-processor array. *International Journal of Circuit Theory and Applications*, 40(12):1249–1264, 2012.

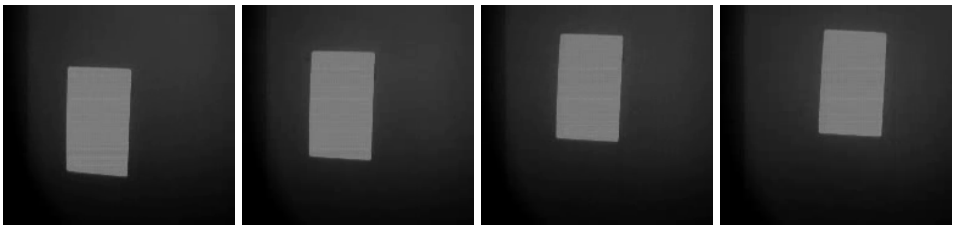
Appendix A

Dataset Description

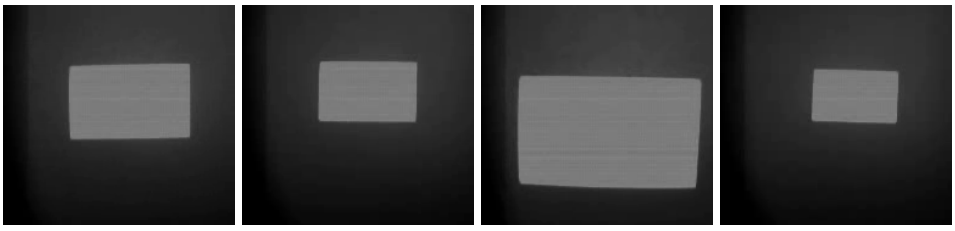
MergingCards0 : Two playing cards being moved sideways, across each other. At some point one eclipses the other, and they look as if they were a single figure.



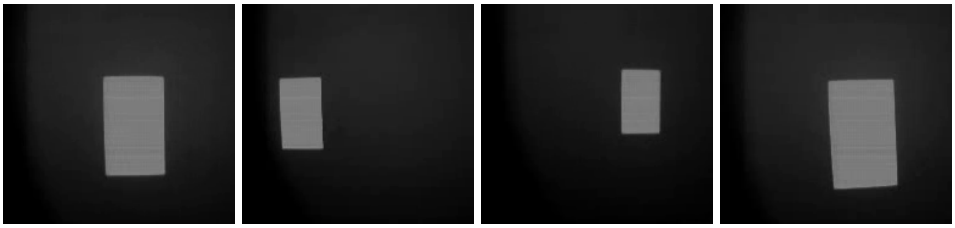
MovingCard2 : A playing card being moved in a circle, in counterclockwise direction. Since the card is not looming or recessing, the video is a good way to test the algorithm's response to sideways motion.



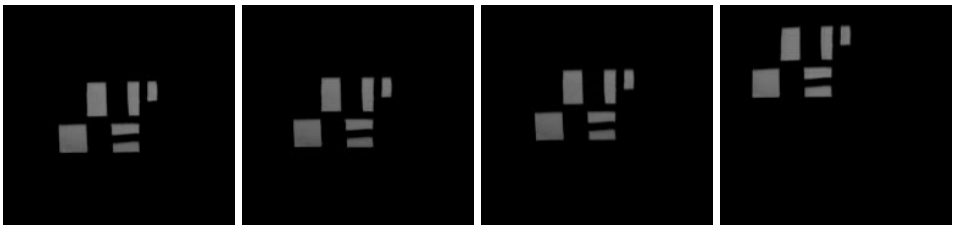
MovingCard3 : A playing card being moved back and forth towards the camera. It is a good way to test the algorithm's ability to detect looming and recessing motion.



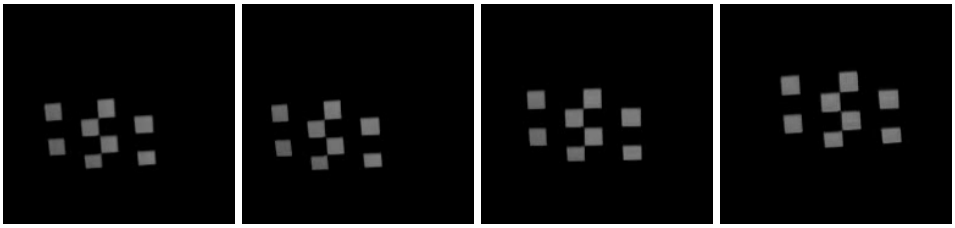
MovingCard4 : A playing card being moved in a horizontal circle, in clockwise motion. The card looms on the right of the picture, then moves to the left, recesses, moves right, and repeats. This is a more complex motion than the previous two. It is a more challenging test to the algorithm.



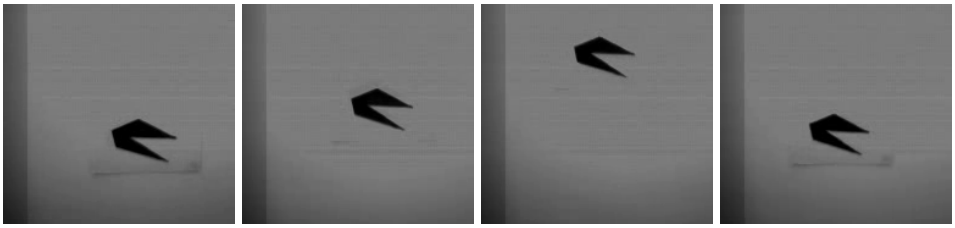
MultipleRect0 : Six rectangles of different sizes and orientations being moved sideways, upwards and downwards. There is no looming motion. It is a fitting test to see how the algorithm reacts to multiple objects, moving as a whole.



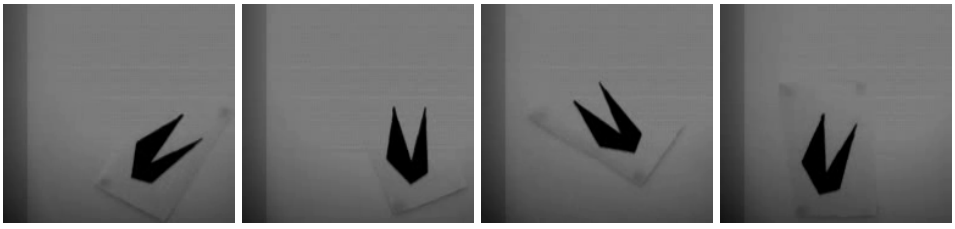
MultipleRect1 : Eight squares are being moved towards the camera, away from it, up and down. It is also rotated around a midpoint in the squares. There is some looming and recessing motion. Again, this is a fitting test to see how the algorithm reacts to multiple objects, moving as a whole.



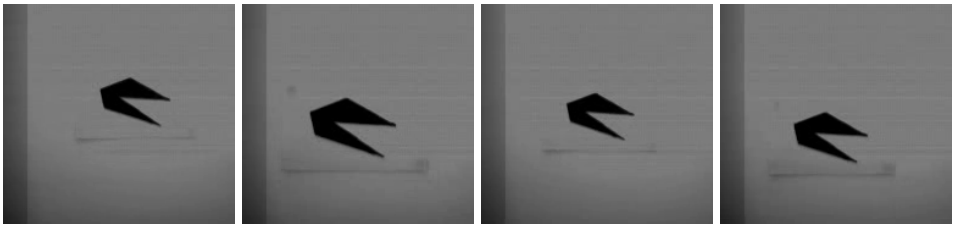
NonConvex0 : A concave figure being moved upwards, downwards and sideways. This one is not looming. This video and the following two will show if the adapted algorithm is able to properly classify the motion of a concave object.



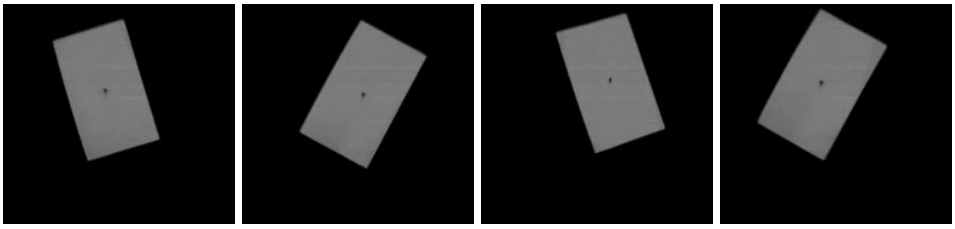
NonConvex1 : A concave figure being moved and rotated. There is no looming motion.



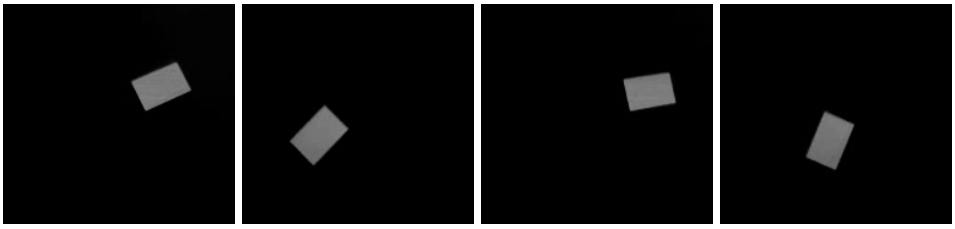
NonConvex2 : A concave figure being moved back and forth towards the camera.



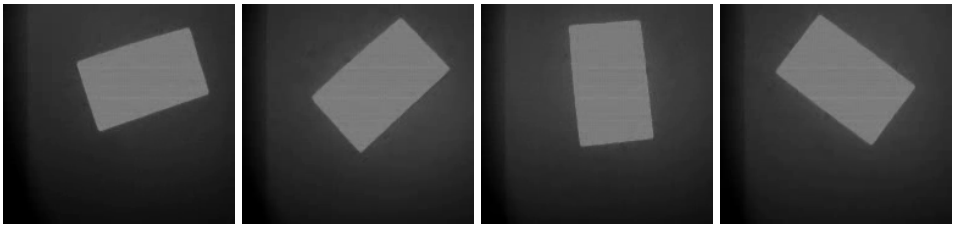
RotatingCard0 : A playing card is being rotated clockwise and counterclockwise. This is a fitting test to see how the adapted algorithm reacts to an object that is not moving laterally, towards, or away from the camera, but rotates instead.



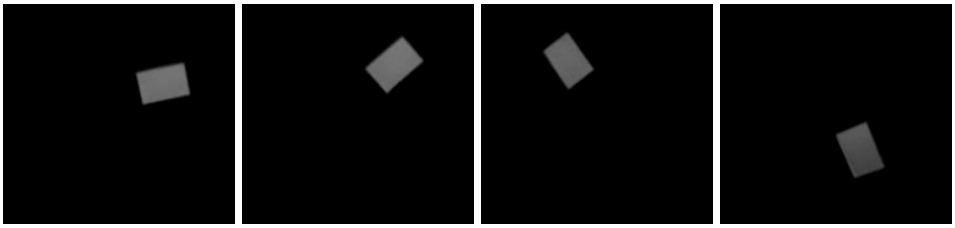
RotatingCard1 : A card following the path of a circle, and rotating in a manner that makes one of its sides to face the centre of the trajectory at all times. It switches between clockwise and counterclockwise directions. In this case the object is presenting lateral motion.



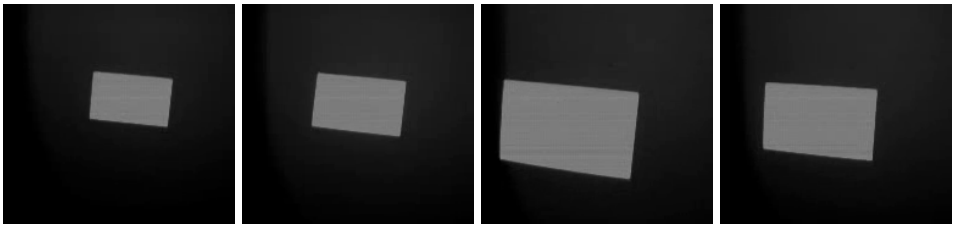
SpinningCard0 : A playing card being rotated in a counterclockwise direction. Just as with RotatingCard0, this is a fitting test to see how the adapted algorithm reacts to an object that moves in its own axis.



SpinningCard1 : A card following the path of a circle, and rotating clockwise in a manner that makes one of its sides to face the centre of the trajectory at all times. This is similar to RotatingCard1, useful to see if the algorithm properly classifies lateral motion.



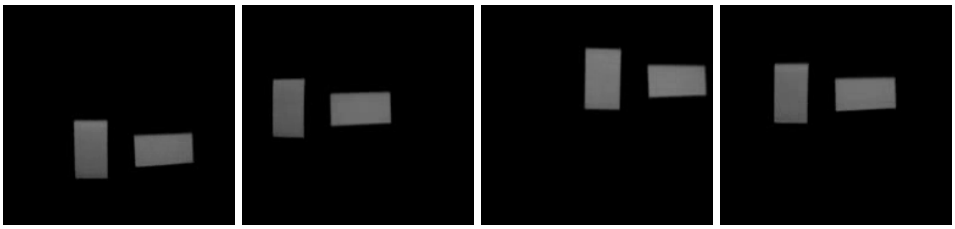
Testvideo : A playing card being moved back and forth towards the camera. This is a good way to start testing the adapted algorithm, it does not have other kind of movement other than looming and recessing.



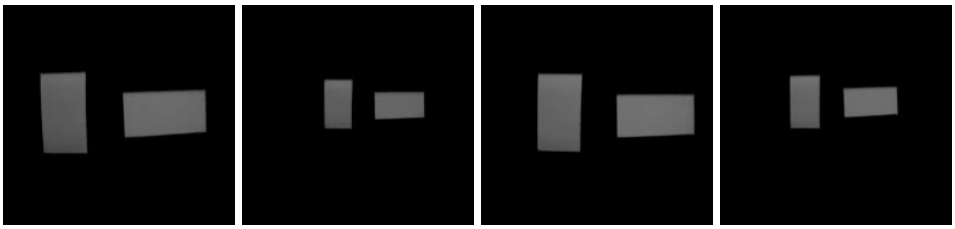
Text0 : A piece of paper containing letters being moved quickly in a small circle. Text0 and Text1 are another fitting test for concave objects as well as multiple objects moving as a whole.

Text1 : A piece of paper containing letters being moved back and forth towards the camera.

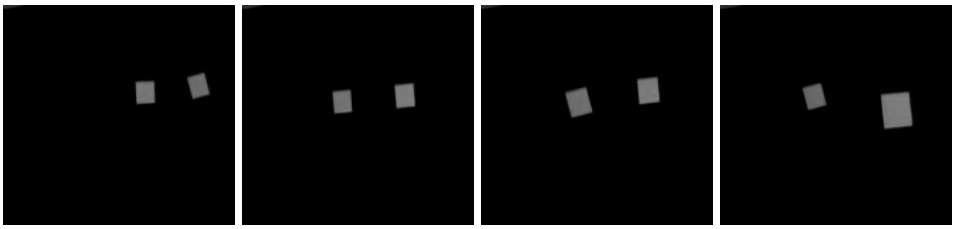
TwoCardsMoving0 : Two playing cards (one rotated 90° from the other) being moved upwards, downwards and sideways. There is no looming motion. TwoCardsMoving0 and TwoCardsMoving1 are another test for multiple objects moving as a whole.



TwoCardsMoving1 : Two playing cards (one rotated 90° from the other) being moved back and forth towards the camera.

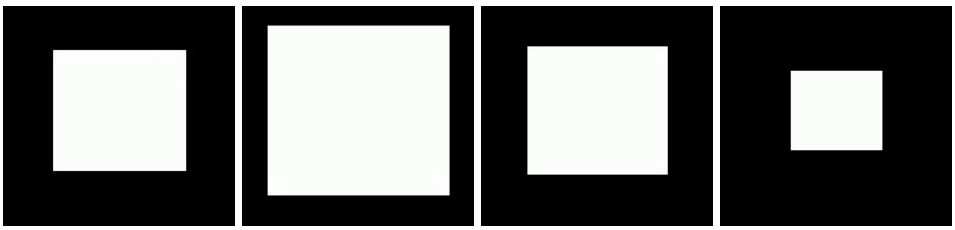


TwoCardsMoving2 : Two cards are being moved back and forth towards the camera. One card is always moving in the opposite direction of the other. This video is a fitting test to see how well the algorithm classifies the motion of two objects moving in different directions.

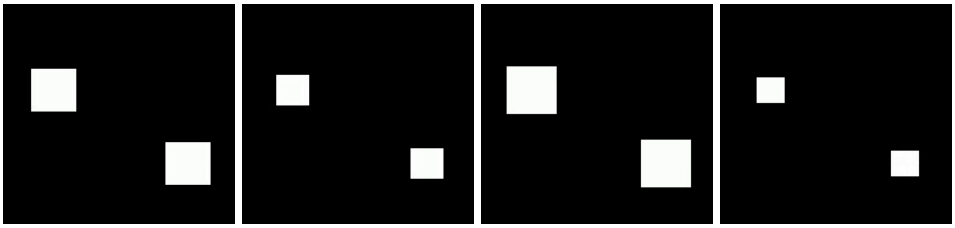


Walking0 : The camera is being carried through a hallway. Elements such as doors and shelves are visible to the sides. The lamps are visible on the ceiling, and their reflection on the floor. Since the camera is the one moving in this video, it the adapted algorithm should detect everything as a looming object.

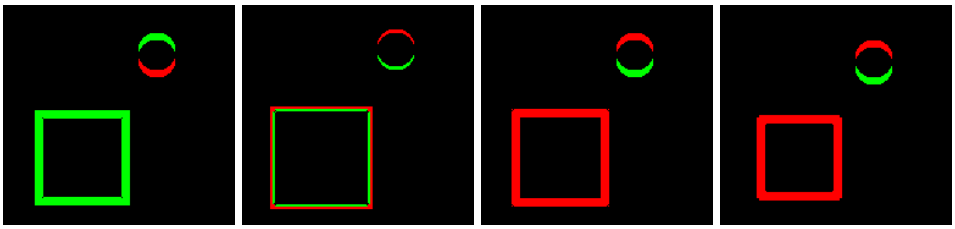
ArtificialOne : A single square that approaches and recesses from the observer. It grows/shrinks exactly one pixel every 50ms.



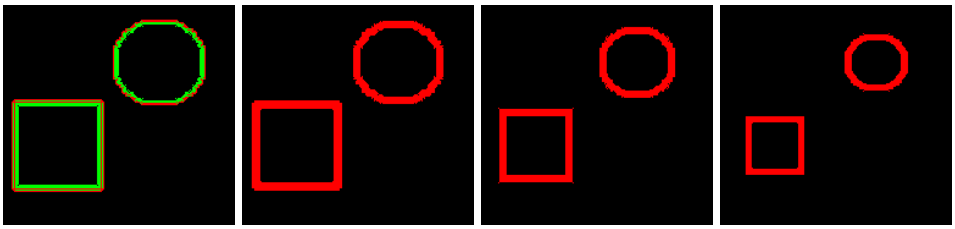
ArtificialTwo : Two squares that approach and recess from the observer at the same time. Each square grows/shrinks exactly one pixel every 50ms.



A01: This is an artificial dataset containing two objects: a sideways moving circle on the top left and a looming and recessing square in the bottom left.

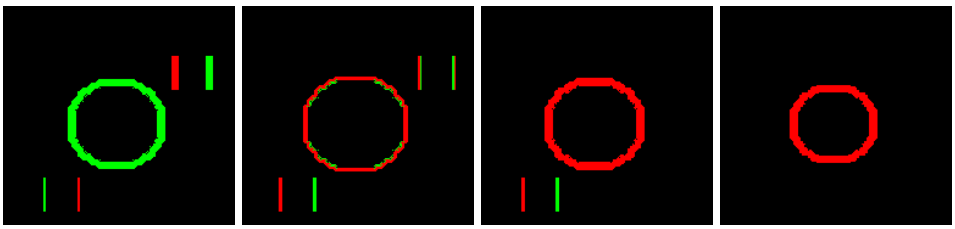


A02: This is an artificial dataset with the same objects as A01. In this dataset both objects are looming and recessing at the same rate.

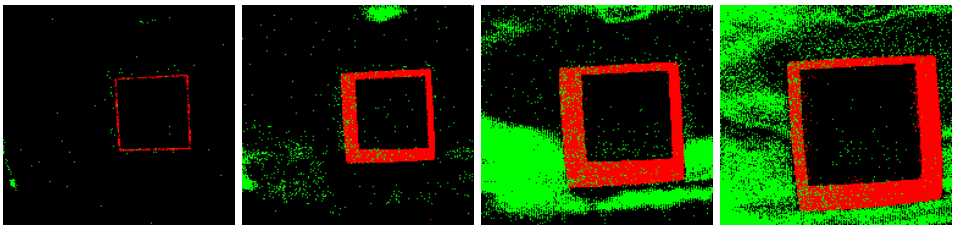


A03: This is an artificial dataset with the same objects as A01. In this case the circle looms and recesses as it did in A02. The square starts looming, then stops for a few milliseconds, and starts looming and recessing again. Since this is a stream of DVS events, when the square stops moving it disappears from the screen.

A13: This is an artificial dataset with a circle at its centre and two squares, one at the top right and another at the bottom left. The circle approaches and recesses while the squares move sideways. Because of their horizontal motion, only the left and right edges are visible.



B09: This is a captured dataset. The main object is a square that is at the centre and bottom right side of the screen. It approaches and recesses in the picture. Some of the pseudo-frames on the right show a considerable amount of noise.



B12: This is a captured dataset. The main object is a circle that approaches and recedes at the centre of the screen.