Insect Detection using Event Cameras

Rahul Thiruthinmel Premnavas
Rahul.ThiruthinmelPremnavas@student.hs-rm.de
RheinMain University of Applied Sciences
Wiesbaden, Hesse, Germany

1 INTRODUCTION

Insects play a very vital role in ecosystems, they are the major agents in pollination, managing the life cycle of all living organisms, etc. Insect populations are decreasing and it is of utmost importance for ecologists to monitor them, with this scenario in the picture they have been tracking insects using various systems. As per [6] Malaise traps, light trapping, and bait trappings are pertinent, but one major disadvantage of all those above-mentioned systems is that these insects die after they get trapped. In this scenario, it is very important to make optimal research in the field. My humble research in this field is noted down below.

1.1 Related works

We will primary look for the answers for the questions

(1) What are the existing methods?
(2) Are there any DVS related works?
(3) What are the existing processing techniques?

In the article written by [5] apart from other monitoring and capturing techniques described above, some radar techniques are mentioned. They have addressed the flaws in the existing techniques for example reflectivity and in equipments. They have proposed a two-radar system shown below to be have addressed the previous problems. They in [5] also proposed novel radar setups with an

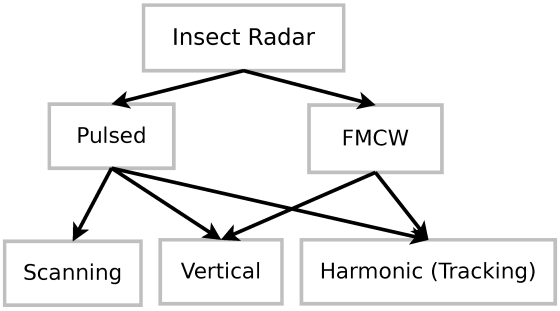

Figure 1: radar groups (figure from [5])

FMCW (frequency modulated continuous wave) radar system for the future considering the limitations in the identification of species and insects-swarms.

In the [4] article several data sets were classified and detected by insect pest detection algorithm. The figure below gives information about it.

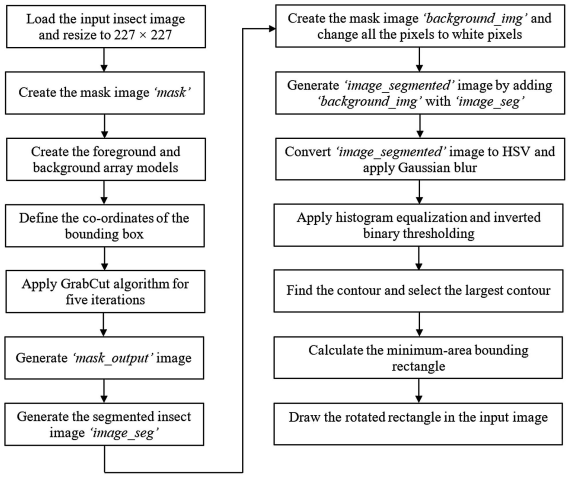

Figure 2: pest detection algorithm

As per [6] the event based monitoring is a very new area of research. We are thus forced to move to find DVS-based based data sets as not much of the information is available in public. One of such is the article published by [2] in that they addressed the fact that no more labelled data sets for algorithm development and evaluation. The labelled data sets that contain the effects of environmental influences outdoor are recorded. They cannot be relied on as the analysis of insect monitoring or labelled data sets of insects are not available.

To get an idea about the DVS-based data sets [2], in this paper, they have mentioned about the difficulties, especially the lack of availability of the labeled datasets, and in their work, they have described a recording setting of a DVS-based long time monitoring of an urban public area and provide labeled DVS data that also contain effects of environmental outdoor influences recorded in the process. They also described the processing chain used for label generation, as well as results from a performed de-noising benchmark utilizing various spatiotemporal event stream filters

Dynamic Vision Sensor-based insect monitoring is a new of its kind and has a sensor-based camera that records events rather than the frames by usual cameras. In this project, we are using DAVIS346 (a Dynamic Vision Sensor Camera) to record events. The specifications of the DAVIS346 are given by the table 1

Conference'17, July 2017, Washington, DC, USA

ACM ISBN 978-x-xxxx-xxxx-x/YY/MM…$15.00
https://doi.org/10.1145/nnnnnnn.nnnnnnn

Spatial resolution	346 x 260
Temporal resolution[1]	1 µs
Max throughput	12 MEPS
Typical latency	<1 ms
Dynamic range	Approx. 120 dB
Contrast Sensitivity	14.3% (on), 22.5% (off)
Spatial resolution	346 x 260
Frame rate	40 FPS
Dynamic range	55 dB
FPN	4.2 %
Dark signal	18000 e−/s
Readout noise	55 e−

Table 1: DAVIS346 Specifications

A Davis346 Event Camera image is also given below for familiarising.

Figure 3: DAVIS346

1.2 Literature Review

Although there is much information available for insect monitoring, we are mainly interested in Literature pertaining to Event Cameras. The Article from [6] gives us a thorough idea about the various steps one should pursue in order to get a real working model of insect detection using event cameras. Various Steps mentioned in the paper are:

(1) Research and Familiarisation
(2) Data Acquisition
(3) Preprocessing
(4) Clustering
(5) Training
(6) Evaluation

Research and Familiarisation include the reality check, getting familiarised with the existing information, and doing my own research on the given topic, which includes reading articles, surfing the internet, and asking for advice from project guides.

Data Acquisition includes the usage of the object at our disposal, in this case, the DAVIS346 event camera. As there are not enough data sets available [6] we had to capture data sets of approximately 30-60 seconds length. The recorded data in.AEDAT4 format is then read using Python for further processing of the data

Pre-processing includes refining of the events captured using the DAVIS346 Camera, and making some filtrations with the given standards which will be explained in the later part of this paper.

Clustering is the pre-processed event data, that helps to identify the specific flight paths, or distinct objects int he event data. We use here the DBSCAN(Density-Based Spatial Clustering of Applications with Noise) Algorithm for the clustering of data.

Training uses the data here very small data set to identify similar objects, here insects.

Evaluation of the results, the accuracy and reliability are determined.

2 METHODOLOGY

In this section, we will discuss step by step, how this project is being carried out. As per [6] DVS technology is the major breakthrough in neuromorphic engineering, They operate on sensors that differ fundamentally from other conventional cameras, the changes in the events are conceivable through the changes in the pixels and the recorded events can later be read using programming languages. DAVIS manufactured offers the necessary library package to read the data. The below code snippet shows how the data can be read using Python.

```
from dv import AedatFile

with AedatFile(<Path to aedat file>) as f:
    # list all the names of streams in the file
    print(f.names)

    # Access dimensions of the event stream
    height, width = f['events'].size

    # loop through the "events" stream
    for e in f['events']:
        print(e.timestamp)

    # loop through the "frames" stream
    for frame in f['frames']:
        print(frame.timestamp)
        cv2.imshow('out', frame.image)
        cv2.waitKey(1)
```

Listing 1: getting data from saved file...

The below code snippet can be used to record live data using the dv library.

```
    from dv import NetworkEventInput


#events connects events to eventserver
with NetworkEventInput(address=
```

```
'127.0.0.1',port=53093) as i:
    for event in i:
        print(event.timestamp)
```

Listing 2: getting data live...

2.1 Data Collection

This is the first and foremost step that i had to undertake while doing the project. The event camera has to be connected to the system to get the event data in .aedat4 file format. As per [6] the Dynamic vision sensor is a comparatively new domain which means the available sources for reference and the available data sets are limited than the frames-based domains. And addition to it the available data sets are recorded either indoors or in the laboratory, thus making an insect monitoring system practically difficult. The author of [6] also mentions about the data sets [1] by Binas et al. which does not have object annotations at all.

For a wider view of the various methods to process the output of a DVS camera as per the author of [6] the readers can have a look at the article published by Gallego [3].

For our study, we captured six videos of 30-60 seconds of duration, which is then further used for preprocessing. all the videos were captured using a DAVIS346 Event Camera with a spatial resolution of 346 x 260 and a temporal resolution of 1 micro seconds. The below-given image shows the difference between a camera image and the output of the DAVIS346 is shown below

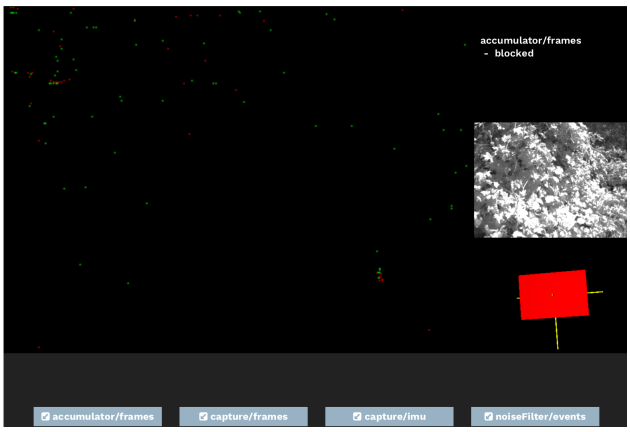

Figure 4: DAVIS346 output

2.2 Pre-processing

For the labeling, it is very important to prefilter the data, which is therefore used to reduce the manual efforts to differentiate the insect movements from other external noises such as movements caused by wind, etc [6]. A statistical outline remover is initially applied by calculating the average distances from each point to the nearest 15 neighbors, then these events more than the average distance, and a standard deviation ion of 0.5 is applied to get the prefiltered coordinates. The below code snippet shows how the data is interpreted.

```
from dv import AedatFile
import cv2
import numpy as np
from scipy.spatial import cKDTree
from scipy.spatial.distance import cdist


with AedatFile("fly1.aedat4") as f:

    events = np.hstack([packet for
    packet in f['events'].numpy()])
    #Access information of all events by type
    timestamps,x,y,polarities =
    events['timestamp'],
    events['x'], events['y'], events['polarity']
     # numberevents = 100
    # for i in range(numberevents):
    #print(timestamps[i],x[i],y[i],polarities[i])

    # points = [[p[1],p[2],p[3]] for p in events]
    # # print(points)
    # # import sys
    # # sys.exit(0)


num_neighbors = 15 # Number of neighbors

x_array = np.array(x)
y_array = np.array(y)

# Combine x and y coordinates
coordinates = np.column_stack
((x_array[:100], y_array[:100]))
 # Build the KD-tree
kdtree = cKDTree(coordinates)
# Query the KD-tree for the 15 nearest neighbors
(including the
distances, indices = kdtree.query(coordinates, k=15)
point itself)
# Calculate the average distance excluding
the point itself
average_distances = np.mean(distances[:, :])

print(average_distances)

std_distance = np.std(average_distances)
threshold = 0.5  #  the standard deviation
# Find the outlier indices
outlier_indices = np.where(average_distances >
(np.mean(average_distances) + threshold *
std_distance))[0]
# Filter out the outlier coordinates
```

```
filtered_coordinates =
coordinates[~np.isin(np.arange(len(coordinates)),
outlier_indices)]

# Print the filtered coordinates
print(filtered_coordinates)

filtered_x = filtered_coordinates[:, 0]
#print(filtered_x)
filtered_y = filtered_coordinates[:, 1]
print(filtered_y)
```

Listing 3: interpreting saved data...

After determining the filtered y and x coordinates from the above data, we now will proceed to find the linearity, which is shown in the below code snippet

```
def calculate_linearity(filtered_x,
filtered_y, radius):
    num_points = len(filtered_x)
    linearity_values = np.zeros(num_points)

    for i in range(num_points):
        x_i, y_i = filtered_x[i], filtered_y[i]
        distances = cdist([(x_i, y_i)],
        np.column_stack((filtered_x,
        filtered_y)))


# Find indices of neighbors within the radius

'r'
        neighbor_indices =
        np.where(distances[0] <= radius)[0]

        # Skip the current point itself
        neighbor_indices =
        neighbor_indices[neighbor_indices != i]

        if len(neighbor_indices) < 3:
# Insufficient neighbors to calculate Eigenvalues,
            set linearity to 0
            linearity_values[i] = 0.0
        else:
# Compute the covariance matrix of the
distances
            covariance_matrix =
            np.cov(distances[:,
            neighbor_indices])


# Calculate Eigenvalues of the covariance matrix
            eigenvalues =
            np.linalg.eigvals(covariance_matrix)

            # Calculate linearity feature
            linearity = (max(eigenvalues) -
            min(eigenvalues)) / max(eigenvalues)
            linearity_values[i] = linearity

    return linearity_values
```

Listing 4: linearity calculations...

2.3 Clustering

Clustering is used to group together events that are likely to be caused by the same object. There are many different clustering algorithms, One we are using is the DBSCAN Algorithm, and this clustered data can be used to identify patterns associated with the insects.

DBSCAN is a density-based clustering algorithm that groups together points that are close in space, here the algorithm takes two parameters eps and minsamples

This code snippet first creates a NumPy array of the filtered coordinates. Then, it creates a DBSCAN object with the specified eps and minsamples parameters. Finally, it calls the fitpredict() method on the DBSCAN object to cluster the data. The fitpredict() method returns an array of cluster labels.

```
    # Combine filtered x and y coordinates into a
    single array
filtered_coordinates =
np.column_stack((filtered_x, filtered_y))

# DBSCAN clustering parameters
eps = 30  # Neighborhood search radius
# Minimum number of neighbors required to identify
a core point
min_samples = 10

# Perform DBSCAN clustering
dbscan = DBSCAN(eps=eps, min_samples=min_samples)
cluster_labels =
dbscan.fit_predict(filtered_coordinates)

# Get the number of clusters found
num_clusters =
len(set(cluster_labels)) - (1 if -
                1 in cluster_labels else 0)

# Print the number of clusters found
print("Number of clusters:", num_clusters)

# Print the cluster labels for each event point
print("Cluster labels:")
print(cluster_labels)
```

Listing 5: Clustering...

2.4 Results

The cluster labels for each event point are printed to the console. The number of clusters found is also printed to the console. This is the number of unique cluster labels minus one[-1], to account for the noise cluster.

```
Number of clusters: 3
Cluster labels:
[-1 -1  0  1  0 -1 -1 -1  0  2  0  0 -1 -1 -1
0 -1 -1  0  1 -1  0 -1  0
 -1 -1  0 -1  2  0  1  2  0  1  0 -1  2  0  0 -1
0  1 -1  0  0  1  1  2
 -1  0  1 -1  2  0 -1 -1 -1 -1  0  1 -1 -1  2
0 -1  0 -1  0 -1  1  2  0
  1 -1  0 -1 -1  2 -1 -1  0  1  1  0  0 -1 -1 -1
0  1  2 -1  1  0 -1 -1
 -1  1 -1  0]
```

2.5 Further possibile Results

Besides we could also write a short snippet to display a 3D event space-time point cloud visualization. A sample code can be provided below with references from the Internet.

```
import numpy as np
import matplotlib.pyplot as plt
from mpl_toolkits.mplot3d import Axes3D

# Assuming you have 'filtered_x',
'filtered_y', 'timestamps', and
'cluster_labels' as described before

# Combine filtered x, y, and
timestamps into a single 'coordinates' array
coordinates = np.column_stack(
(filtered_x, filtered_y, timestamps))

# Create a 3D plot using Matplotlib
fig = plt.figure()
ax = fig.add_subplot(111, projection='3d')

# Dictionary to map cluster labels to
unique colors
color_map = {-1: 'gray'}
# Assign noise points a gray color

# Generate a set of unique cluster

labels (excluding the noise label -1)
unique_clusters = set(cluster_labels)
unique_clusters.discard(-1)

# Assign a unique color to each cluster
for cluster_label in unique_clusters:
    color_map[cluster_label] =
    plt.cm.jet(cluster_label /
    len(unique_clusters))
    # Using jet colormap

# Plot the event data points in the
3D space-time point
cloud with cluster-based colors
for i in range(len(coordinates)):
    cluster_label = cluster_labels[i]
    color = color_map[cluster_label]
    ax.scatter(coordinates[i, 0],
    coordinates[i, 1],
    coordinates[i, 2], c=color, marker='o')

# Customize the plot properties
ax.set_xlabel('X')
ax.set_ylabel('Y')
ax.set_zlabel('Timestamp')
ax.set_title('3D Event
Space-Time Point Cloud with Clustering')

# Show the plot
plt.show()
```

Listing 6: 3D event space-time point cloud.....

The possible outcome is illustrated in [6] and the figure with the original caption is provided below.

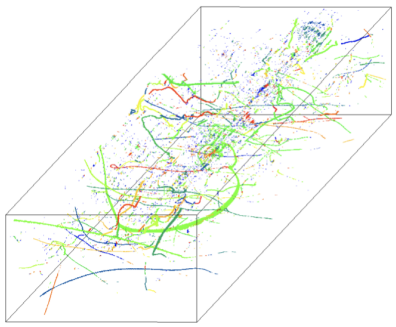

Figure 4: 3D event space-time point cloud after clustering. Clusters are highlighted by random colors.

Figure 5: from the citation [6]

2.6 Training

In the results, we can observe labels like 1, 0, and 2 that represent clusters. It's crucial to mention the to-be-realized section, in our project as it greatly impacts the success of any machine learning-based project. The quality and relevance of the training data utilized

play a role in this regard. Properly preparing and training the dataset with care is essential.

To ensure that the model can generalize well comprehensive validation processes were conducted concurrently with the training process. To prevent overfitting and enhance the model's capability to handle variations in real-world scenarios it is advisable to utilize techniques such, as validation and data augmentation (source: www.kaggle.com).

A machine learning model can be trained to recognize various insect species using the filtered coordinates. A dataset of filtered coordinates that have already been classified as belonging to various insect types would be used to train the model. The model would then develop the ability to recognize the data patterns that distinguish between various insect types. Once trained, the model may be used to recognize different insect species based on their filtered coordinates. a possible model from the web sources has been represented below for future reference.

```
# Import necessary libraries
import numpy as np
import tensorflow as tf
from tensorflow import keras
from sklearn.model_selection
import train_test_split
from sklearn.preprocessing import LabelEncoder
from sklearn.metrics
import classification_report

# Outline for Training an Insect Detection Model

## Step 1: Data Loading and Preprocessing

# Load the dataset (replace with your
own dataset loading code)
data, labels = load_dataset()
# data is a list of event sequences,
labels indicate presence/absence of insects

# Preprocess the data (e.g., noise removal,
feature extraction)
data_preprocessed = preprocess_data(data)

## Step 2: Data Splitting

# Split the data into training and testing sets
X_train, X_test, y_train, y_test =
train_test_split(data_preprocessed, labels
, test_size=0.2, random_state=42)

## Step 3: Encoding Labels

# Encode class labels to numerical values
label_encoder = LabelEncoder()
y_train_encoded = label_encoder.fit_transform(y_train)
y_test_encoded = label_encoder.transform(y_test)

## Step 4: Model Definition

# Define a deep learning model
(replace with your own model architecture)
model = keras.Sequential([
    keras.layers.Input(shape=(input_shape)),
    # Define input shape based on your data
    keras.layers.Conv2D(32,
    kernel_size=(3, 3), activation='relu'),
    keras.layers.MaxPooling2D(pool_size=(2, 2)),
    keras.layers.Flatten(),
    keras.layers.Dense(128, activation='relu'),
    keras.layers.Dense(1, activation='sigmoid')
    # Binary classification
])

# Compile the model
model.compile(optimizer='adam',
loss='binary_crossentropy', metrics=['accuracy'])

## Step 5: Model Training

# Train the model
model.fit(X_train, y_train_encoded,
epochs=10, batch_size=32, validation_split=0.1)

## Step 6: Model Evaluation

# Evaluate the model on the test data
test_loss, test_accuracy =
model.evaluate(X_test, y_test_encoded)

# Generate predictions
y_pred = model.predict(X_test)

# Decode numerical labels back to original class labels
y_pred_decoded =
label_encoder.inverse_transform
(np.round(y_pred).astype(int))

# Calculate classification report
classification_rep =
classification_report(y_test, y_pred_decoded)

# Print model evaluation results
print("Test Loss:", test_loss)
print("Test Accuracy:", test_accuracy)
print("Classification Report:\n",
classification_rep)
```

Listing 7: training...

2.7 Conclusion and Future Work

We described a technique in this research for grouping insect movement based on their filtered coordinates. The technique was tested on a data set of filtered coordinates, and it was found to be successful in clustering insects. The technique did not, however, identify the insects in each cluster.

Future work on this paper will include:

- Figuring out a way to identify the insects in each cluster. This might be accomplished by classifying the insects based on their attributes using a machine-learning model.
- Adding more bug species to the dataset of filtered coordinates. This would enhance the accuracy of the identification process and enable the clustering algorithm to find more clusters.
- Enhancing the clustering algorithm's performance by utilizing a different machine-learning model. A model that is better at spotting patterns in the data could be used to achieve this.

According to [6] analysis of flying patterns is a first step in discriminating between different insect species. The examination of trajectories provides a rough differentiation. Some butterflies, for instance, have a tendency to flutter erratically and in zigzags, whereas bumblebees, on the other hand, have a tendency to fly purposefully in the direction of a flower. There are basically three methods mentioned in [6] those are a) classification based on derived features directly from DVS Events b) classification based on polarity c) classifications based on simulated gray scale. However [6] preferred the classification based on simulated grayscale on their implementation.

To detect species in swarms [7] has come up with good insights, they have also provided a data set of three-dimensional, time resolved trajectories as data sets.

REFERENCES

[1] Jonathan Binas, Daniel Neil, Shih-Chii Liu, and Tobi Delbruck. Ddd17: End-to-end davis driving dataset, 2017.

[2] Tobias Bolten, Regina Pohle-Frohlich, and Klaus D Tonnies. Dvs-outlab: A neuromorphic event-based long time monitoring dataset for real-world outdoor scenarios. In *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, pages 1348–1357, 2021.

[3] Guillermo Gallego, Tobi Delbrück, Garrick Orchard, Chiara Bartolozzi, Brian Taba, Andrea Censi, Stefan Leutenegger, Andrew J. Davison, Jörg Conradt, Kostas Daniilidis, and Davide Scaramuzza. Event-based vision: A survey. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(1):154–180, 2022. doi: 10.1109/TPAMI.2020.3008413.

[4] Thenmozhi Kasinathan, Dakshayani Singaraju, and Srinivasulu Reddy Uyyala. Insect classification and detection in field crops using modern machine learning techniques. *Information Processing in Agriculture*, 8(3):446–457, 2021.

[5] Alexey Noskov, Joerg Bendix, and Nicolas Friess. A review of insect monitoring approaches with special reference to radar techniques. *Sensors*, 21(4):1474, 2021.

[6] Regina Pohle-Fröhlich and Tobias Bolten. Concept study for dynamic vision sensor based insect monitoring. pages 411–418, 01 2023. doi: 10.5220/0011775500003417.

[7] Michael Sinhuber, Kasper Van Der Vaart, Rui Ni, James G Puckett, Douglas H Kelley, and Nicholas T Ouellette. Three-dimensional time-resolved trajectories from laboratory insect swarms. *Scientific data*, 6(1):1–8, 2019.