Detecting Invasive Insects Using Uncewed Aerial Vehicles and Variational Autoencoders

Scott Daniel Stewart
Marquette University

DETECTING INVASIVE INSECTS USING UNCREWED AERIAL VEHICLES AND VARIATIONAL AUTOENCODERS

by

Scott Daniel Stewart

A Thesis submitted to the Faculty of the Graduate School,
Marquette University,
in Partial Fulfillment of the Requirements for
the Degree of Master of Science

Milwaukee, Wisconsin

December 2021

ABSTRACT
DETECTING INVASIVE INSECTS USING UNCREWED AERIAL VEHICLES AND VARIATIONAL AUTOENCODERS

Scott Daniel Stewart

Marquette University, 2021

In this thesis, we use machine learning techniques to address limitations in our ability to monitor pest insect migrations. Invasive insect populations, such as the brown marmorated stink bug (BMSB), cause significant economic and environmental damages. In order to mitigate these damages, tracking BMSB migration is vital, but it also poses a challenge. The current state-of-the-art solution to track insect migrations is called mark-release-recapture. In mark-release-recapture, a researcher marks insects with a fluorescent powder, releases them back into the wild, and searches for the insects using ultra-violet flashlights at suspected migration destination locations. However, this involves a significant amount of labor and has a low recapture rate. By automating the insect search step, the recapture rate can be improved, reducing the amount of labor required in the process and improving the quality of the data. We propose a solution to the BMSB migration tracking problem using an unmanned aerial vehicle (UAV) to collect video data of the area of interest. Our system uses an ultra violet (UV) lighting array and digital cameras mounted on the bottom of the UAV, as well as artificial intelligence algorithms such as convolutional neural networks (CNN), and multiple hypotheses tracking (MHT) techniques. Specifically, we propose a novel computer vision method for insect detection using a Convolutional Variational Auto Encoder (CVAE). Our experimental results show that our system can detect BMSB with high precision and recall, outperforming the current state-of-the-art. Additionally, we associate insect observations using MHT, improving detection results and accurately counting real-world insects.

ACKNOWLEDGMENTS

Scott Daniel Stewart

I would like to express my thanks to my advisor: Dr. Henry Medeiros for his guidance with my degree.

I would also like to thank my friends and family for their love and support throughout my life in education.

CONTENTS

LIST OF TABLES

LIST OF FIGURES

LIST OF ALGORITHMS

CHAPTER 1
INTRODUCTION

Invasive insects such as the Brown Marmorated Stinkbug (BMSB) cause over $120 billion in damages every year in the United States [1]. Invasive species disturb and threaten ecosystems. These damages are expected to continue to increase as more invasive species are introduced to new environments, such as in 2020, when "Murder Hornets" were found in the United States for the first time, making national headlines [2]. Any solution to the invasive insect problem must first be able to determine how they migrate, otherwise removing them from one area may become a temporary solution. The typical method for tracking the migration of invasive species is Mark-Release-Recapture [3]. An example of this method is when ecologists capture a bear, put a collar on the bear, release it into the wild, and recapture the bear at a later date [4]. We can monitor where the bear travels based on the unique identifier on the collar itself. Unfortunately, while it is possible to affix localization devices to BMSBs, it is a labor-intensive, time-consuming, and consequently expensive process [5]. Currently, one of the most widely used methods to capture mark and recapture invasive insects relies on coating the insects with fluorescent powder and going to fields at night to try and locate the insects using an ultraviolet light source, but this procedure is inconsistent and time-consuming [3].

Uncrewed aerial vehicles (UAVs), also known as drones, have the ability to efficiently solve this problem. Instead of manually searching for the insects in the field, using a drone to automate the process could reduce the human effort required to detect the insects while potentially increasing insect retrieval rates. Furthermore, we can use computer vision technology to automate the detection of the insects on the videos collected by the drones. Existing methods can successfully detect insects from

videos acquired by drones [1]. However, these methods do not take full advantage of recent developments in machine learning techniques applied to computer vision and hence leave room for improvement. Besides, existing systems do not have the capability of associating multiple insect detections acquired at different times or to map them to a global coordinate system.

In this thesis, we present a novel approach for detecting invasive insects using UAVs and computer vision techniques. Specifically, we use a computer vision algorithm called a convolutional variational auto encoder (CVAE) [6] that learns to represent the expected appearance of an image. Then, by taking the difference between the real image and the expected image only "anomalies" are left. In our case, these anomalies are the image pixels corresponding to insects. We then associate multiple detections among consecutive frames using multi-object tracking algorithms and project these association to a global coordinate frame based on the trajectory information reported by the drone during data collection flights.

1.1 Contributions

This work presents a significantly more accurate method for detecting the migration of invasive insects using computer vision than existing methods. Specifically, it provides the following contributions:

1. We propose and evaluate a novel use of anomaly detection methods for determining the location of insects in aerial videos.
2. We propose and evaluate a novel use of multiple object tracking methods to keep track of insects across multiple frames in a dataset.
3. We introduce an additional annotated dataset of illuminated insects in a nighttime setting.

1.2 Organization

This work is organized into the following chapters. Chapter 1 provides a brief introduction to the problem and the proposed approach to address it. Chapter 2 introduces related works and other research relevant to this project. Chapter 3 discusses the background knowledge related to this thesis. Chapter 4 describes the methods used to collect and annotate the insect data. Chapter 5 contains the descriptions of the overall system design and its components as well as the experimental results. Finally, Chapter 6 provides a conclusion and final thoughts related to the thesis.

CHAPTER 2
RELATED WORK

This chapter discusses recent research contributions closely related to this thesis. The interdisciplinary nature of this thesis requires some background on topics such as artificial intelligence in agriculture (2.1), anomaly detection in computer vision(2.3), multiple object tracking methods (2.4), among others. This section aims to to provide readers with the needed context. The current state of the art in automated insect detection using drone videos is based on simple color thresholding strategies, which are effective in limited scenarios. When there are leaves or other elements in the background, such methods may fail to accurately find the insects. To compound on this issue, these methods do not keep track of which insect detection in a given video frame should be associated with detections in other frames, so they can only determine in which frames an insect is found, but not how many insects are found in the entire video sequence.

2.1 Computer Vision in Agriculture

Computer vision has proven a useful and increasingly popular tool in agricultural applications [7]. These applications include detecting weeds [8], grading the quality of fruits and vegetables [9], analyzing the flowers on fruit bearing trees to determine bloom intensity [10], measuring the growth of pecan nuts [11], vision-based orchard monitoring and management [12, 13], among several others. There have been a few different previous methods for detecting invasive insects with uncrewed aerial vehicles and computer vision [1, 14]. For example, in [14], the authors used drones to monitor changes in reflectance of crops and use that information to detect outbreaks of invasive insects and other pests. Other previous methods [1] detect insects coated with reflective powders from images collected by drones using specialized illumina-

tion systems and performing operations directly on the features of the images, such as thresholding the color channels (Section 5.6.1) or clustering image pixels based on their colors using the k-means algorithm (Section 5.6.2).

2.2 Insect Detection

Modern techniques to track invasive insects involve marking the insects with a bright fluorescent powder and searching for them over a region of interest using high-intensity ultraviolet lights. This method has a number of drawbacks as it requires substantial investments of time and effort while still being inefficient. Rice et al. found it took volunteers 14 minutes, on average, to find 80% of the BMSBs placed within a large tree. [15]. Recent efforts to track invasive insects have made great strides toward improving the efficiency and the accuracy of mark-release-recapture studies. Rice et al. have improved the process to apply fluorescent powders to the insects [15]. Rojas-Araya et al. [16] have shown that the fluorescent powder has a slight effect on insect behavior, but does not affect their survival rates. On the other hand, Kirkpatrick et al. [3] have noted little influence of fluorescent coatings on the mobility or survival rates of the BMSB. Other studies aim to remove the need for fluorescent powder coatings altogether. Kirkpatrick et al. [17] have attached passive communication devices to the insects, making it possible to recapture the insects at unprecedented rates. However, it is difficult to attach these devices to the insects, making this approach time-consuming and expensive. Computer vision techniques might provide a non-contact alternative to these strategies.

Computer vision has been applied to the insect detection problem before. Ebrahimi et al. use Support Vector Machines (SVM) to classify if strawberry plants in greenhouses are affected by Thrips (a small insect that is harmful to crops) [18]. Alves et al. use Deep Residual Networks, a type of Neural Network, to classify what type of pest, if any, is affecting cotton crops [19]. Kaya et al. implement an

effective classifier for determining species of butterfly [20]. Huddar et al. created an effective algorithm for the segmentation of whiteflies (another type of harmful insect) on leaves [21]. These algorithms, while effective at the task they are designed for, need the camera to be significantly closer to the insect than is possible with a UAV.

2.3 Anomaly Detection

Detecting invasive insects using computer vision techniques can be cast as an instance of the unsupervised anomalous pixel detection problem. Anomalous pixel detection is the problem of trying to identify which, if any, pixels in an image are different from the corresponding pixels in a typical image for a given application. For example, anomalous pixel detection can be used to examine x-ray images of circuit boards to quickly determine if any solder joints do not complete a circuit [22]. State-of-the-art methods for unsupervised anomaly detection in images typically employ one of three strategies: i) search for salient features directly in the original image [23], ii) learn to represent the image in a deep latent feature space and look for anomalous latent features [24, 25], iii) or learn to recreate the image and evaluate the corresponding reconstruction error [26]. In this thesis, we choose to use the third method as it can be solved using unsupervised machine learning strategies. Unsupervised methods are preferable for the anomalous insect detection problem since there is a lack of comprehensive labeled data and creating such datasets would be difficult for a number of reasons. First, annotating videos containing insects is difficult and time consuming, and must be done multiple times to ensure no human error affects the results. In addition, it is necessary to obtain permission from the Federal Aviation Administration (FAA) to fly drones to collect data, which further hinders the data gathering procedure.

Recent efforts have greatly improved anomaly detection for one-class and multi-class classification problems [27, 28, 29, 30]. Many of these algorithms focus on

detecting anomalous images within a set of images instead of looking for anomalous pixels within an image. Existing methods for anomalous pixel detection typically analyze deep latent features of the images generated by a Convolutional Neural Network (CNN) to locate anomalous areas [31] or use generative models, such as Generative Adversarial Networks (GANS) [32] or Variational AutoEncoders (VAEs) [33, 34], to compute the reconstruction error of the recreated image. Bergmann et al. [31] investigates the performance of using a student-teacher approach to solving the anomalous pixel problem with promising results. However, their approach is semi-supervised and greatly benefits from transfer learning strategies [35]. This makes this approach less-than-ideal for insect detection. Similarly, Napoletano et al. [36] take advantage of a pretrained Resnet-18 model [37] and principal component analysis (PCA) with good results, but that method also requires training data, which precludes its application to the insect detection problem. Because of these limitations, we chose to approach the problem using methods based on generative models.

Methods based on generative networks such as such as GANS [32] or VAEs [6, 33, 34] attempt to recreate the original image based on latent features generated by a deep neural network and compute the reconstruction error of the image to determine where the expected image differs from the true image. Perera et al. [32] propose a novel approach that adds an extra discriminator to the latent dimension of a GAN to reduce reconstruction error. The discriminator forces the latent dimension to follow a uniform distribution. While this works well to sort anomalous images by class, it has not been applied to the anomalous pixel problem. Fan et al. [38] achieve remarkably good performance on anomalous pixel detection by combining the results from a Convolutional Variational Auto-Encoder (CVAE) and an optical flow model. Their method is able to isolate abnormal events in video footage such as a vehicle driving on a pedestrian path. Unfortunately, for the insect detection problem, as long as the insects remain stationary, their displacement among consecutive video

frames would be indistinguishable from that of background clutter objects, since the drone moves at a constant velocity with respect to the background. This, however does not mean that CVAEs are not suitable for this problem, as they can still detect anomalies without resorting to optical flow information. Hence, we choose to build upon previous works on anomaly detection using CVAEs to solve the insect detection problem.

2.4 Multiple Object Tracking

Multiple Object Tracking (MOT) corresponds to a class of algorithms that can track multiple objects across multiple frames of a video. This has a number of practical applications such as tracking cars which can improve autonomous vehicles, and tracking pedestrians [39, 40]. MOT methods are often composed of a number of models, such as an appearance model, a motion model, and an interaction model, working in conjunction [41].

2.4.1 Multiple Hypotheses Tracking

Multiple Hypotheses Tracking (MHT) is a method for tracking multiple objects by making a number of hypotheses based on previous observations of the detection [42]. By taking the appearance, location, and velocity of detected objects into consideration, MHT can associate detections, so each detection has a single temporal identifier across the video sequence. In our case this means each 'real' insect is assigned a unique identifier in all the frames in which it appears.

2.4.2 The MOT Metrics

Bernardin et al. created a set of metrics to evaluate and compare Multiple Object Tracking techniques: the MOT Metrics [43, 44]. The Mostly Tracked metric corresponds to the number of objects tracked for at least 80% of the lifespan of the ground truth objects. Similarly, Mostly Lost is the number of objects tracked less than

20% of the lifespan of the ground truth objects. Partially Tracked is the number of objects tracked between 80% and 20% of their lifespan. MOTA is the multiple object tracking accuracy, and is defined as

$$MOTA = 1 - \frac{\sum_t (m_t + fp_t + mme_t)}{\sum_t g_t}, \tag{2.1}$$

where m_t is the number of misses, fp_t is the number of false positives, mme_t is the number of mismatches at time t, and g_t is the total number of ground truth ids. ID Switches is the count of Track Switches. For example, an ID Switch happens when the ground truth insect is labeled in the prediction as 'A' for 10 frames and switches to 'B' after that. Number of Fragmentations is the count of tracks that start tracked and become untracked or vice-versa. ID Precision is defined as

$$ID_{Precision} = \frac{ID_{TruePositives}}{ID_{TruePositives} \times ID_{FalsePositives}}, \tag{2.2}$$

where $ID_{TruePositives}$ and $ID_{FalsePositives}$ are the number of true positives and false positives found by using a global assignment algorithm respectively.ID Recall can be defined as

$$ID_{Recall} = \frac{ID_{TruePositives}}{ID_{TruePositives} \times ID_{FalseNegatives}}, \tag{2.3}$$

where $ID_{TruePositives}$ and $ID_{FalseNegatives}$ are the number of true positives and false negatives found by using a global assignment algorithm respectively [44]. ID F1 can be defined as

$$ID_{F1} = 2 \times \frac{ID_{Recall} \times ID_{Precision}}{ID_{Recall} + ID_{Precision}}. \tag{2.4}$$

CHAPTER 3
BACKGROUND

In this chapter we cover a number of well established methods and techniques used throughout this thesis. We discuss three main topics: Uncrewed Aerial Vehicles, Multiple Object Tracking, and Image Segmentation.

3.1 UAVs in Agriculture

Uncrewed Aerial Vehicles are becoming an increasingly popular tool for collecting data remotely and monitoring large areas [45]. This is because they allow a single operator to perform tasks that normally would require multiple people, and in some cases can be operated automatically [46]. By recording video from a UAV, we can inspect a large area for invasive insects without the need for manually walking around the area. A significant advantage to using UAVs for pest insect detection is that they are already in use for a variety of agricultural applications. Therefore, the cost of implementing new UAV solutions is significantly reduced. Popular uses for UAVs in agriculture include mapping crops [47], spraying pesticides [48], monitoring crops for weed growth [49], irrigation [50], and diagnosing the symptoms of pests [45]. Because there already exists a number of uses for UAVs in agriculture, the cost associated with the proposed solution could be dramatically amortized as the investment on the UAV might cover several tasks.

3.1.1 Uncrewed Aerial Vehicle Operation and Regulatory Issues

Because drones introduce both safety and privacy concerns, we underwent the process of acquiring a part 107 drone pilot's license [51]. This process includes taking a course, understanding airspace regulations, understanding Aviation Routine Weather Reports (called METARs), and passing an examination to prove sufficient

knowledge. This license is provided by the FAA and is designed for commercial drone operators. Due to the nature of this project's data collection needs, which require night time drone operation, we also needed to acquire an FAA night-time operation waiver (§107.29 - Operation at night) [52]. To obtain night flight permission one must apply to the FAA and provide a plan including detailed descriptions of the steps the pilot will take to reduce the chances of a collision. In our case this involved attaching a strobe to the top of the UAV and having multiple observers on the ground to ensure safe flight. After applying for and receiving the waiver we were able to collect a number of night-time data sets.

3.2 Image Segmentation

Image segmentation consists of assigning the pixels in an image into a class [53]. This is useful for the insect detection problem as each pixel in the dataset is either an insect or a background object.

3.2.1 Pixel Intensity Thresholding

By applying a threshold to the intensity levels of the individual pixels of a gray-scale image, it is possible to segment its contents into foreground and background elements. This can be represented as a simple equation:

$$p_{\text{cls}} = p_{\text{val}} \geq \tau, \tag{3.1}$$

where p_{val} is the value of the pixel intensity, and τ are is the thresholding value, and p_{cls} is the corresponding pixel class, i.e., pixels with $p_{\text{cls}} = 1$ (i.e., those for which $p_{val} \geq \tau$) correspond to the foreground whereas pixels with $p_{\text{cls}} = 0$ are part of the background. That is, any pixel intensity greater than or equal to τ is classified as a foreground pixel, and any value less than that is a background pixel. For color images, multi-thresholding techniques can be applied to the individual image channels. The main challenge with thresholding is finding an appropriate value for τ, and techniques

such as Otsu's method (described in Section 3.2.2) attempt to address that problem. However, Otsu's always finds a single, fixed threshold, which assumes that all the background pixels have lower intensity than foreground pixels (or vice-versa). This is not always the case for the invasive insect detection problem as there are often frames of the dataset without any insects whatsoever. However, that problem can be solved by dynamically adjusting the threshold on individual video frames and verifying whether the segmentation results are consistent with the presence of insects [1].

3.2.2 Otsu's Method

Otsu's method [54] is one of the simplest ways to define a threshold to segment a gray-scale image, and can serve as a baseline when creating and testing new segmentation methods. Otsu's method classifies each pixel of the image as belonging either to the "foreground" or the "background". Thus, it can only be applied to binary segmentation problems. This strategy is applicable to the insect classification problem as we only need two classes: "insects" and "background".

Otsu's method tries to find an optimal threshold for determining the "foreground" pixels based on the distribution of pixel intensities within an image. This is done by maximizing the inter-class variances:

$$\sigma_{bkg}^2 = \frac{\sum_{i=1}^{T}(i-\mu_{bkg})^2 p_i}{\omega_0}, \tag{3.2}$$

and

$$\sigma_{frg}^2 = \frac{\sum_{i=T+1}^{i_{max}}(i-\mu_{frg})^2 p_i}{\omega_1}, \tag{3.3}$$

where

$$\omega_0 = \sum_{i=1}^{T}(p_i), \tag{3.4}$$

and

$$\omega_1 = \sum_{i=T+1}^{i_{max}} p_i. \tag{3.5}$$

In these equations σ^2_{bkg} and σ^2_{fgd} are the inter-class variances for the background and the foreground, respectively, T represents a tentative threshold value, i_{max} is the highest pixel intensity present in the image, μ_{fgd} and μ_{bkg}, are the average pixel intensities of the foreground and background at the current threshold level. ω_0 and ω_1 are both functions of T. p_i defines what percentage of pixels have a an intensity of i for each value of $i = T + 1, \ldots, i_{max}$. Because both functions are entirely based on the value of the threshold, we can rewrite the equation as

$$\sigma_{combined}(T)^2 = \frac{[\mu_{image}\omega_0 - \mu_T]^2}{\omega_0\omega_1}, \tag{3.6}$$

where

$$\mu_T = \sum_{i=0}^{i_{max}-1} i \cdot p(i), \tag{3.7}$$

and μ_{image} is the mean pixel value of the image. The variance is computed for every value of $1 \leq T \leq i_{max}$ and the value that maximizes $\sigma_{combined}$ is selected as the threshold. This threshold is then applied to every pixel in the image to classify pixels greater than or equal to the threshold as "foreground" and the others as "background."

This approach works relatively well for the application under consideration as the insects are generally brighter than the background so they should be classified as foreground pixels. However, there are shortcoming to Otsu's method. For example, leaves often look similar to the insects in the nigh-time videos. Another downside is that Otsu's method always finds a threshold between the minimum and maximum value in the dataset, so if there is no insect in the data it will incorrectly pick a threshold value as the insect cutoff. This problem is illustrated in Figure 3.1.

3.2.3 K-Means Clustering for Segmentation

K-means is an algorithm to cluster data points together. It accomplishes this goal by randomly determining k different seed points, and associating each data point with one of the k seeds. Then, the means of all points associated with the

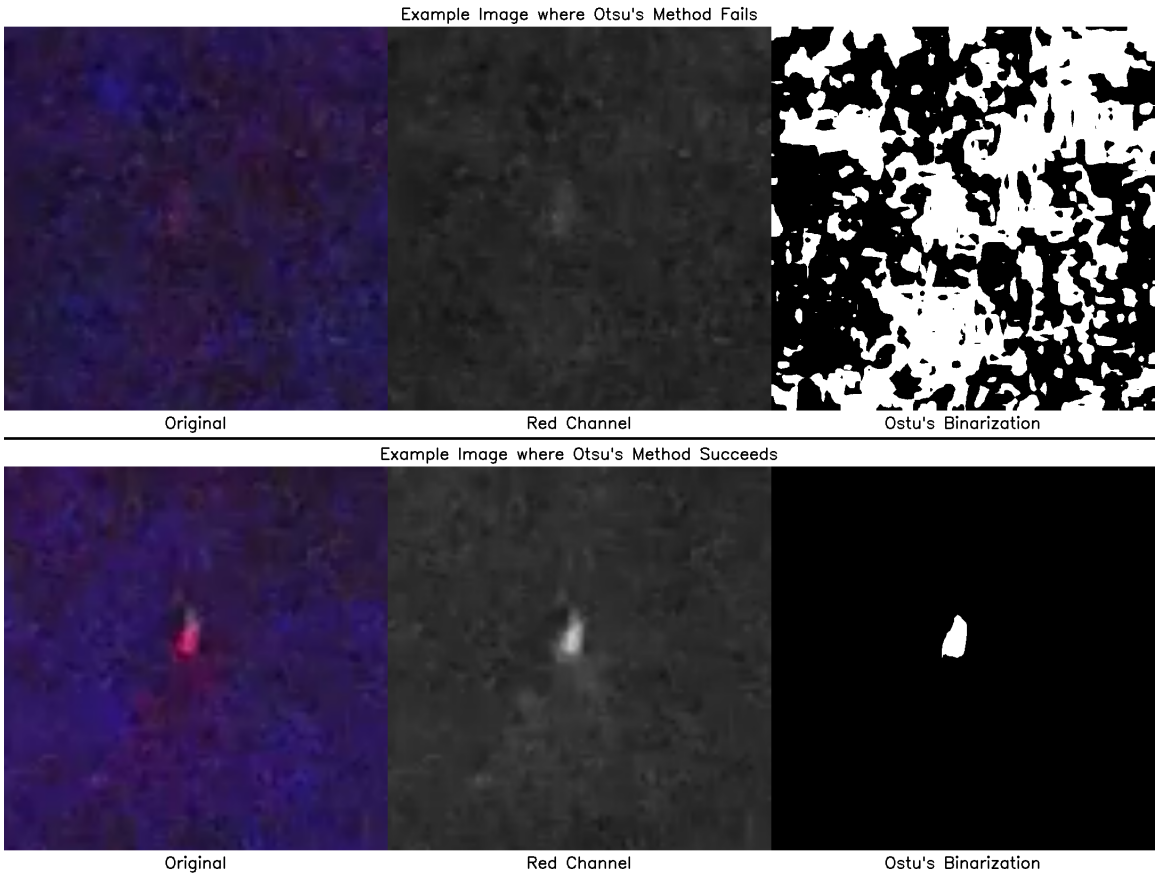

Figure 3.1: A sample image where Otsu's method fails (top row), and a sample image where it succeeds (bottom row). In both cases there is an insect and Otsu's method is used to find a threshold on the red channel of the image.

corresponding seed are computed. The means then become the new seeds for the next iteration. This is repeated iteratively until a stopping criterion is met. The stopping criterion is often a number of iterations, or a threshold on the total change in the values of the cluster means.

K-means can be used as a segmentation method, with a number of advantages. For example, it is possible to define different types of distance metric, so position in an image as well as similarities between pixel colors can be taken into account. Furthermore, the number of classes k segmented by k-mean clustering is flexible. K-means clusters data points into k clusters centered on a mean based on iteratively

minimizing the average distance metric to the k points. This allows it to find patterns in unlabeled data, or to find outliers. Both of these applications of k-means are used and discussed. The number of clusters k can either be determined beforehand or be iteratively tested to minimize the silhouette score. Then, a distance metric must be determined. This is often the Euclidean distance among the elements in a dataset. The k-means algorithm can be described as follows: 1) Assign k random center points (centroids), 2) Calculate the distance from each point in the dataset and assign it to the nearest centroid, 3) Move the centroid to the mean location of all of the points assigned to it, 4) Repeat everything after the initial random points until there is no longer any change in the location of the centroids, or a defined maximum has been found.

In order to apply this to a segmentation problem, instead of using random initial locations, a starting location is selected using prior knowledge. However, for many segmentation problems an adequate prior is not known. Fortunately, prior information is known for the anomalous insect detection problem: pixels whose hue is close to orange are more likely to correspond to insects. Besides, it is known that there are a large number of background pixels compared to insect pixels. Stumph et al. [1] leverages that information by executing the k-means algorithm multiple times on each image, so that the detected foreground cluster of the image is iteratively refined. The cluster with the higher hue value is kept and k-means is re-run until convergence. This process can be seen in Algorithm 1, which also pre-filters the image pixels using a set of experimentally determined color thresholds to increase the robustness of the algorithm to background clutter.

3.2.3.1 Silhouette Scores

When evaluating different implementations of K-means it is important to have a criterion to assess how well a set of parameters performs. Silhouette scores provide

Algorithm 1: Insect detection with K-means.

Input : Cropped ROI video frame, $I_n^{(N)}$
Output: Binary image, $I_n^{(b)}$, where foreground pixels are insects

```
for Each pixel, p_i^(r) do
    if p_h > τ_h and p_v > τ_v then
        p_i^(r) = p_i^(r);
    else
        p_i^(r) = 0;
L = 0 ;
while L < γ and μ_h^(f) < μ_h^(r) + σ_h and μ_v^(f) < μ_v^(r) + σ_v do
    (c_f, c_b) = k − means(I_n^(r) ≠ 0) ;
    L = L + 1;
```

one such criterion. For example, by iterating through various values of k to determine how many clusters exist in a dataset, the Silhouette score can give provide a measure of the performance improvement obtained by increasing the number of clusters. While performance always increases with the value of k, if it does not increase by a significant amount, the additional cluster may have over-partitioned the data rather than found a new real cluster. K-Means performs at its best when all elements in any cluster are normally distributed around the mean. The Silhouette score can be used to determine if this is whether that assumption is being satisfied. The Silhouette Score is defined as:

$$S(i) = \frac{\alpha - \beta}{min[\alpha, \beta]}, \tag{3.8}$$

$$\alpha = \left(\min_{k \neq 1} \frac{1}{C_k} \sum_{j \in C_k} d(i,j) \right), \tag{3.9}$$

$$\beta = \left(\min_{k \neq 1} \frac{1}{|C_k| - 1} \sum_{j \in C_k, i \neq j} d(i,j) \right), \tag{3.10}$$

for each centroid. Lower values of S indicate better clustering performance.

3.2.4 AutoEncoders

In contrast with traditional neural networks for classification and regression, AutoEncoders are generative and unsupervised learners. As a result, AutoEncoders have been used for a large variety of tasks such as generating synthetic images [55]. This section explores the basic concepts related to AutoEncoders, Variational AutoEncoders, and Convolutional Variational AutoEncoders.

AutoEncoders have several benefits over many other generative models. Like other generative models, AutoEncoders allow the production of synthetic data. Unlike many other generative models, they allow you to do that by either randomly generating new data, or modifying existing data. This allows researchers to have control over where the data goes, and what happens to the data.

An AutoEncoder comprises two separate neural networks in one model. These networks are often referred to as the Encoder and Decoder, but in this thesis, we will referred to them as the Inference Network and the Generative Network. The Inference Network takes the input and learns to represent it in a lower dimensionality space. This space is often referred to as either the latent space or the z-dimension. This is similar to how a classification neural network operates. In a classifier, the network learns to represent the data in a latent space that is consistent with each class of data.

The Generative Network is trained to take the latent dimension, z, and reconstruct the original image. This is usually done by mirroring the structure of the Inference Network. For example, a 2D convolution becomes a 2D convolutional transpose of the same size. The Inference Network is trained hand-in-hand with the Generative Network, and optimized to encode for the Generative Network. They are both trained via back-propagation. This means that AutoEncoders are specifically optimized to recreate their own input. The networks are trained with a mean-squared

error or cross-entropy loss between the original data and the reconstructed data. This penalizes the network for incorrect reconstruction of the data.

Because the latent dimension contains less information that the original image the Generative Network must lose some information. However, this is compensated by the Generative Network as it learns what the typical appearance of an image, so less information is needed to recreate the original image. Together, the Inference Network and Generative Network can effectively encode an image into significantly less information than what they originally contained.

3.2.4.1 Limitations of Traditional AutoEncoders

Traditional AutoEncoders have certain shortcomings. Because of the nature of the interaction between the Inference Network and the Generative Network, the latent space is often not used to its fullest potential. While they are still useful for a number of tasks, they fall short at creating synthetic data and their reconstructions are often of low quality. The problem lies with the latent space: it is often only partially used. When building synthetic data it would be better to be able to randomly create data from the latent space. We can do this by examining how data populates the latent space and exploring the populated areas. When there are unpopulated areas in the latent space the decoder is still be able to create an output, but it is not similar to any trained inputs. This results in unrealistic or misleading outputs. Variational AutoEncoders aim to solve this problem.

3.2.5 Variational AutoEncoders

Variational Autoencoders (VAEs) are able to generate latent spaces that are more uniform than traditional Autoencoders [56]. Traditional Autoencoders tend to use the same areas of the latent space to represent different information, losing information. Variational Autoencoders try to use more of the latent space by adding small amounts of noise to the latent space. In VAEs, instead of encoding the input

onto a latent dimension, the Inference Network encodes it to a latent space comprised of two sections. These sections represent the mean μ and variance σ of a normally distributed latent vector z:

$$z = \mathcal{N}(\mu, \sigma). \tag{3.11}$$

This means that for the same input, the latent dimension is allowed to vary to some extent. Despite this variation of the latent vector, we expect the VAE to generate the same output. As a consequence, each input inherently uses more of the latent space, so the total dataset encompasses a much larger region of the latent space. As long as the latent vector is centered around the 'true' encoding (i.e., the mean), the standard deviation determines a manifold around it where everything inside the manifold corresponds to the same encoding. This means that the generative net explores a significantly larger number of latent dimension values, even when trained multiple times on the same data. This strategy creates smoother latent spaces with a small amount of overlap. The Inference network must then learn to generate different means for each class, essentially clustering them apart. That is, the inference network attempts to maximize the distances among the means of the distribution while minimizing their standard deviations so that encodings themselves do not vary too much for the same sample. To accomplish that goal, we must optimize a loss function that penalizes the network when it tries to center the encoding around a mean of 0 and a standard deviation of 1. If the standard deviation becomes 0, the network would simply function as a traditional AutoEncoder, so penalizing this prevents it from happening. By doing this we encourage the AutoEncoder to utilize more of the latent space. This is accomplished by minimizing the the Kullback-Leibler (KL) divergence:

$$\mathcal{L}_{KL} = \sum_{i=1}^{n} \left(\sigma_i^2 + \mu_i^2 - \log(\sigma_i) - 1\right), \tag{3.12}$$

where σ_i are the variances generated for the latent space, μ_i are the corresponding means, and n is the dimensionality of the latent space. This loss is minimized when

each $\mu_i = 0$ and $\sigma_i = 1$. However, this alone would not cause the network to reconstruct the original data. We must also incorporate the mean square error (MSE) or the cross entropy loss (CEL) of the reconstruction error into the loss function in combination with the KL loss to correctly reconstruct the image while maintaining a well explored latent space. This can be defined as

$$\mathcal{L}_{oss} = \mathcal{L}_{KL} + \mathcal{L}_r. \tag{3.13}$$

$\mathcal{L}_{oss}$ is the loss function used to train the network and $\mathcal{L}_r$ is the MSE or CEL loss of the reconstruction error. If MSE is chosen, it can be defined as

$$\mathcal{L}_r = \frac{\sum_{i=1}^{n_x} \sum_{j=1}^{n_y} (\mathcal{F}_{i,j} - R_{i,j})^2}{n_x \times n_y}. \tag{3.14}$$

In this case, n_x and n_y are the number of pixels in the x and y dimensions of the input image. $\mathcal{F}_{i,j}$ is the value of the input image at pixels i, j and $R_{i,j}$ is the value of the reconstructed image at pixels i,j. If CEL is chosen, the loss can be defined as

$$\mathcal{L}_r = \frac{\sum_{i=1}^{n_x} \sum_{j=1}^{n_y} \mathcal{F}_{i,j} \times \log(R_{i,j}) - (1 - \mathcal{F}_{i,j}) \times \log(1 - R_{i,j})}{n_x \times n_y}. \tag{3.15}$$

3.3 Multiple Object Tracking

Multiple Object tracking is a common problem in computer vision [57]. It typically consists of monitoring the location of several objects in a video sequence by detecting them in multiple frames and temporally associating the detections, a framework known as tracking-by-detection. For example, autonomous vehicles need to both detect the cars around them and keep track of their trajectories. There are a number of successful multiple object tracking solutions available [58, 59, 60], but the vast majority of these approaches take advantage or rely entirely on appearance models [58, 59]. Unfortunately most of these method cannot be used to associate insects detected over a sequence of video frames, as the insects look virtually identical, and are small with relatively no visual information. Therefore, our solutions must

rely on movement and location information only. The following sections describe to strategies explored in this thesis.

3.3.1 Constrained Clustering for Multiple Object Tracking

Algorithm 2: Insect association using constrained K-means.

Input : Set of Bounding Boxes and frame numbers of Insect Detections I_o

Output: A set of means representing the insect locations

```
for K = 1, ..., K_max do
    M_p = ∅;
    M_n = {m_k}_{i=1}^K, where m_k ∈ ℝ^3 is a uniformly distributed vector;
    C = {C_k}_{k=1}^K, where C_k is the set of points closest to m_k;
    while M_p ≠ M_n do
        M_p = M_n;
        I_d = I_o ;
        for i ∈ I_d do
            I_d = I_d \ {i};
            T = C;
            while T ≠ ∅ do
                k_min = argmin_{k∈M_n} {||i − m_k||} ;
                C_{k_min} = C_{k_min} ∪ {i};
                for j ∈ C_{k_min} do
                    if j_frame = i_frame then
                        if ||i − m_{k_min}|| < ||j − m_{k_min}|| then
                            I_d = I_d ∪ {j};
                            C_{k_min} = C_{k_min} \ {j};
                        else
                            I_d = I_d ∪ {i};
                            C_{k_min} = C_{k_min} \ {i};
                T = T \ C_{k_min};
                M_n = M_n \ k_min;
                for m, c ∈ M_n, C do
                    m = (Σ_{i=1}^{|c|} c_i) / |c|;
```

One strategy for multiple object tracking is to associate detections over multiple frames using constrained clustering techniques [61]. For example, the constrained k-means algorithm [62] allows previous knowledge about the points to be clustered to be incorporated in the form of "must-link" and "cannot-link" constraint graphs. Points that are connected through an edge in the "must-link" graph are forced to be placed in the same cluster, whereas points connected by an edge in the "cannot-link" graph are not allowed to be clustered together. In our problem, we create cannot-link constraints such that if two insect detections appear in the same frame they cannot correspond to the same 'real' insect.

Algorithm 2 illustrates our implementation of this simplified constrained k-means approach. For our specific problem, two practical considerations must be addressed. First, the number of clusters K should be equal to the the number of unique insects observed in a video sequence. To address that problem, the algorithm iterates over the values of $k = 1, \ldots, 10$ to optimize over the silhouette score and determine the correct number of insects over the video frames. Second, when assigning data points to the nearest cluster centroid, we must check whether there is already an element from the same frame in that centroid. If there is, we assign the element to the cluster whose centroid is closest to it, and assign the new element to the second closest cluster. This is repeated until there are no more changes to the locations of the centroids.

CHAPTER 4
DATA COLLECTION SYSTEM

Gathering data is essential to tackling the anomalous insect detection problem. By gathering accurate and varied data we can test the robustness of various methods of detecting insects. This chapter presents the hardware and methods for gathering the data.

4.1 Field Data Acquisition System

The Unmanned Aerial Vehicle used to capture videos of the insects is a DJI Matrice 100 (M100) quadcopter shown in Figure 4.1. We customize the vehicle with an array of ultraviolet LEDs mounted on the bottom, and with an extra battery bay for extended flight duration. The drone also carries a Zenmuse X3 camera, which records video with a 4096×2160 pixel resolution at 29 frames per second. The M100 has an extended flight time compared to typical UAVs even with an additional payload. This, however, comes with the downside of being heavier than other commercial UAVs. This extended flight time is required due to the large size of the fields being analyzed.

4.2 Hardware Testing

Ensuring that hardware works as expected is an important step to collecting accurate data. In this section we talk about how we tested the various hardware components such as the UAV (4.2.1), the Camera (4.2.2), and the Ultra Violet Light (4.2.5).

4.2.1 UAV testing

The unmanned aerial vehicle must be able to carry out several tasks in order to complete the assigned mission. First, it must be able to carry the weight of the

Figure 4.1: The DJI Matrice 100 drone.

added lighting array. Second, it must be able to tell the real-world location of the drone during any given frame of the video. Third, it must have a long enough flight duration to take a large enough dataset for our purposes. Finally, it must be both safe and easy to operate and use.

4.2.1.1 M100 flight log information

In order to obtain the drone's real-world position and time data, the M100 flight logs must be accessed. The flight logs can be extracted via USB using the manufacturer's specialized data acquisition software, which runs on the Microsoft Windows operating system. The flight logs are stored in a proprietary binary format and must be converted into a readable format. We convert it to comma-separated values format (CSV) using a publicly-available open-source conversion library[1]. Recently, DJI has also made a publicly available tool to convert the flight logs into CSV

[1]https://datfile.net/DatCon/downloads.html

Table 4.1: Matrice 100 flight log values. Captured and recorded 30 times per second.

Field	
Tick#	Battery(0):minCurrent | maxCurrent | avgCurrent
offsetTime	Battery(0):minVolts | maxVolts | avgVolts
IMU_ATTI(0):Longitude	Battery(0):minWatts | maxAmps | avgAmps
IMU_ATTI(0):Latitude	Motor:Speed:Rfront | LFront | LBack | RBack
IMU_ATTI(0):numSats	Motor:EscTemp:Rfront | LFront | LBack | Rback
IMU_ATTI(0):barometer:Raw	Motor:PPMrecv:Rfront | LFront | LBack | Rback
IMU_ATTI(0):barometer:Smooth	Motor:V_out:Rfront | LFront | LBack | Rback
IMU_ATTI(0):accel:X | Y | Z | Composite	Motor:Volts:Rfront | LFront | LBack | Rback
IMU_ATTI(0):gyro:X | Y | Z | Composite	Motor:Current:Rfront | LFront | LBack | RBack
IMU_ATTI(0):mag:X | Y | Z | Mod	Motor:Status:Rfront | LFront | LBack | RBack
IMU_ATTI(0):mag:Y	Motor:thrustAngle
IMU_ATTI(0):mag:Z	AirComp:AirSpeedBody:X | Y | Alti | VelNorm
IMU_ATTI(0):mag:Mod	AirComp:AirSpeedGround:X | Y | VelLevel
IMU_ATTI(0):vel:N | E | D | Composite | H | GPS-H	eventLog
IMU_ATTI(0):roll | pitch | yaw | yaw360	IMUEX(0):err
IMU_ATTI(0):totalGyro:X | Y | Z	flyCState
IMU_ATTI(0):magYaw	flycCommand
IMU_ATTI(0):distanceHP	flightAction
IMU_ATTI(0):distanceTravelled	nonGPSCause
IMU_ATTI(0):directionOfTravel[mag]	compassError
IMU_ATTI(0):directionOfTravel[true]	connectedToRC
IMU_ATTI(0):temperature	Battery:lowVoltage
flightTime	RC:ModeSwitch
navHealth	gpsUsed
General:vpsHeight	visionUsed
General:relativeHeight	IMUEX(0):err
General:absoluteHeight	MotorCtrl:Status
GPS(0):Long | Lat | Date | Time | HeightMSL	MotorCtrl:PWM:Rfront | LFront | LBack | RBack
GPS:dateTimeStamp	AirCraftCondition:fsmState | nearGround | landState
GPS(0):hDOP | pDOP | sAcc	AirCraftCondition:launch_acc_dur | launch_delta_v
GPS(0):numGPS | numGLnaS | numSV	AirCraftCondition:thrust | gyro | gyro_acc | land_dur
GPS(0):velN | velE | velD	AirCraftCondition:thrust_proj_gnd
RC:Aileron | Elevator | Rudder | Throttle	AirCraftCondition:thrust_proj_gnd_compen
Controller:gpsLevel | ctrl_level	AirCraftCondition:thrust_compensator
Battery(0):cellVolts1 | 2 | 3 | 4 | 5 | 6	AirCraftCondition:hover_thrust | safe_tilt
Battery(0):current | totalVolts | temp | batter% | watts	Attribute|Value
Battery(0):FullChargeCap | RemainingCap | voltSpread	ConvertDatV3

files. Table 4.1 lists the information available in the flight logs. Unfortunately, none of these options directly translates into the drone's heading. Hence, we estimate the drone's heading based on its position, as discussed in greater detail in Section 5.4.

4.2.1.2 Maximum Flight Duration Tests

Because of the custom payload, the weight of the M100 nearly doubled. This results in a dramatic change in the expected flight duration of the aircraft. In addition, the camera draws power from the Matrice 100 batteries, so power depleted by the

camera must be taken into consideration when collecting data. It is thus necessary to test the maximum flight duration of the modified aircraft in motion while recording a video. For that purpose, Stumph et al. fly the UAV in a square pattern measuring 20 meters per side over the field south of Marquette University's Engineering Hall[1]. The M100 has a return-to-home function that causes the M100 to return to the takeoff point when its battery is depleted. Therefore, the test is terminated when the return-to-home function is triggered. The UAV was tested both with and without the added weight of the UV light array. Without a payload the UAV weights 998g, and with the payload it weighs $2,187$g.The flights lasted 27 minutes 34 seconds and 17 minutes 49 seconds, respectively. While this is a significant decrease, a 17-minute flight duration is acceptable for the data collection as our largest dataset can be collected in less than 13 minutes, as shown in Section 4.3. It is worth noting that decreasing the weight of the payload would significantly increase the flight duration of the drone. Furthermore, the flight duration of the drone may be affected by the age of the battery being used.

4.2.1.3 Pitch Angle Testing

UAVs generate forward velocity by slightly pitching their noses downwards. The faster the UAV moves the further the UAV needs to pitch forward. While the camera is attached to a gimbal to minimize the effect of the pitch angle on the videos, the lighting array is attached directly to the bottom of the M100, so the further the drone pitches forward the more out-of-frame the UV light beam becomes. Because we can define a constant velocity for the drone we can calculate a corresponding constant pitch angle. To do this, we must determine the UAV's pitch angle as a function of its velocity. For this test, we varied the drone velocity between 1 m/s and 5 m/s in increments of 0.5 m/s, and monitored the M100 flight logs to identify the corresponding pitch angles. Because of the initial change in pitch angle the drone

makes in order to begin moving, the first few seconds of each experiment have a large amount of noise and are ignored. Each test was conducted by Stumph et al, who fly the drone for 45 meters in a straight line starting from a standstill[1]. The pitch angle was obtained from the M100 flight logs. The maximum defined pitch angle where the UV light still sufficiently falls in the center of frame of the drone videos is approximately 16° [1], and all tests up to and including 5 m/s fall within this limit. However, there are still other metrics to consider when determining the maximum drone speed as discussed in Section 4.2.4.

4.2.1.4 Stability Testing

To verify that the added weight of the light array does not dramatically affect the stability of the drone a stability test was undertaken. The test was conducted both with and without the added payload of the illumination array as discussed in Section 4.2.5 by Stumph et al[1]. The total weight of the drone is 2.187g and 998g with and without the payload, respectively. The stability test measured whether the drone could properly hover in place with and without the payload. Both experiments were conducted with a wind speed of approximately 13 k/h. Using the information from the M100 flight logs we calculated its stability metrics [1]. If the drone had similar pitch, roll, and yaw angle variations with and without a payload in similar conditions, we can conclude that the payload has minimal effect on stability. To calculate if the stability in each test was similar we calculate the Mean Root Squared Error(E_{RMS}) of the yaw pitch and roll of each test, which is given by:

$$E_{RMS} = \sqrt{\frac{\sum_{i=1}^{n}(\hat{y}_i - y)^2}{n}}, \tag{4.1}$$

where y is the initial value and $\hat{y}_i$ is the value of each sample of the yaw, pitch and roll. We found that the average E_{RMS} of the default configuration is 0.59°, and the UV light configuration is 0.40°. We found that the worst E_{RMS} of the default configuration is 0.34° and the UV light configuration is 1.90°. While this increase is

high it is not large enough to cause concern especially since in the data acquisition the UAV only moves forward, so variations in roll or yaw are minimized. This means the UAV is stable enough with and without the UV lighting array.

4.2.1.5 Operating the UAV

Because we must perform data collection at night when visibility is limited, drone safety and ease of operation are especially important. Fortunately, DJI includes a number of safety features in their drones. In addition to the return-to-home feature described above, which also lands the drone where it took off if the signal between the controller and the drone is lost. Furthermore, the Matrice 100 drone can be operated via the DJI Ground Station Pro App for the iPad[2].

4.2.2 Camera Testing

The Zenmuse X3 camera used in our experiments is both practical and meets the needs of our project. The camera captures 4k videos, which provides sufficient resolution to allow the insects to be visible in the video frames. The camera also comprises a 3-axes gimbal to correct for any unexpected pitch, yaw, or roll variations the drone may experience. In addition, the Zenmuse X3 camera is equipped with a microSD card reader that can be changed quickly in the field, allowing multiple tests to be performed back-to-back.

4.2.2.1 Camera ISO and Shutter Speed testing

Due to the nighttime data collection, it is important to optimize camera parameters for low-light levels. To do this, we focus on two relevant camera parameters: ISO and shutter speed. The ISO affects how much light the camera's sensor is able to capture. However, higher ISO values lead to increased image capture noise. Usually, this can be fixed with a longer exposure by adjusting the camera's shutter speed.

[2]https://www.dji.com/ground-station-pro/info

However, because the drone and camera are moving during data collection, if the shutter speed is too long motion blur can occur. For our work, we need to find the optimal point adjusting these two parameters that allow maximum image brightness without adding significant noise or motion blur. To accomplish this goal, we conducted a number of experiments to optimize these values.

Stumph et al. first test several configurations of ISO values. The Zenmuse X3 camera supports ISO values from 100 to 3200 on an exponential scale (100, 200, 400, 800, 1600, 3200). We visually inspected the videos collected at each ISO value at a height of 10 meters, as determined in section 4.2.3. For this test, we determined that the insects were only visible for ISO values of 800, 1600, and 3200. Then, a second test was performed by flying the UAV at 1 m/s over two groups of three insects with a shutter speed of 1/25 s using the three ISO values identified in the first test. This was evaluated by measuring the performance of frame-by-frame insect segmentation using Otsu's method [1, 54], which is discussed in detail in Section 3.2.2. If the camera's parameters are incorrect the segmentation will contain noise or no visible insects. In either case, Otsu's method will fail to separate it from the background. Therefore, by calculating the precision of the segmentation results we can find the optimal settings. Based on this evaluation strategy, we determine that only the 1600 and 3200 ISO options allowed for the insects to be properly segmented.

To determine which of these ISO options better meets the needs of the problem under consideration, we tested the shutter speed and ISO combinations in tandem. Specifically, for each ISO value, we evaluated at 1/40 s, and 1/60 s [1].At an ISO of 3200 we tested at 1/80 s. 1/25 s was already tested in the previous ISO test so it was left out. These results can be seen in Table 4.2. The settings with the highest precision were chosen for the camera settings, so an ISO of 1600 ISO and 1/25 s shutter speed was chosen.

Table 4.2: Precision results when applying Otsu's method to various ISO and shutter speeds.

ISO Value	**Shutter Speed** (s)	**Precision**
1600	1/25	0.30
1600	1/40	0.23
1600	1/60	0.00
3200	1/40	0.24
3200	1/60	0.28
3200	1/80	0.00

4.2.3 Maximum Altitude Testing

If the UAV can fly at a higher altitude, it can cover more ground over a shorter period of time. However, at higher elevations the insects become smaller in the video frames and the UV light beam diffuses, reducing the effectiveness of the illumination system. To determine the maximum data collection height, we tested the drone at multiple altitudes. These tests were conducted at night by hovering the drone with the lighting array attached at an altitude of 8 meters and slowly flying upwards until the insects can no longer be seen. While it was found insects were still visible at an altitude of 11 meters, visibility was significantly reduced when compared to an altitude of 10 meters, so 10 meters was selected as the data collection altitude.

4.2.4 Maximum Velocity Testing

Table 4.3: Motion blur maximum velocity testing.

Velocity (m/s)	**Insect Hue**	**Bkg Hue**	**Insect Value**	**Bkg Value**
1.0	227.59	183.13	78.62	31.75
1.5	207.69	182.37	59.34	25.20
2.0	203.41	183.31	47.91	25.20
2.5	205.29	181.62	48.91	22.63

The faster the drone moves the more ground it can cover in one data collection mission. However, higher speeds introduce motion blur in the collected images. To assess the impact of motion blur, we recorded footage the drone flying over clusters of insects at varying velocities. The videos were recorded using the camera settings discussed in section 4.2.2.1. To determine the effect of motion blur, a metric must be created. When the pixels in the image belong to an insect, they typically have a higher hue and value than the typical background pixels. This is especially true under the conditions in which the tests were performed: a recently mowed, healthy lawn in the summer. Under these conditions, there is relatively little noise in the data collection. We observed that when the drone moves too quickly no insects are visible in the video frames, so the test was limited to speeds of up to 2.5 m/s. We performed our test at the speeds of 1.0, 1.5, 2.0, and 2.5 m/s. For each test, we compared the background hue and value channels to the hue and value channels of the pixels corresponding to insects. The results can be seen in Table 4.3. The difference in hue and value between the insects and the background slowly decrease as the drone's velocity increases. However, the change in difference between 1.0 and 1.5 m/s is more significant than at other speeds. The data acquired at 1.0 m/s has very high quality when compared to all of the other data at the cost of longer data collection times. Hence, we chose to collect data at 1.0 m/s.

4.2.5 UV Light Testing

The Ultra Violet (UV) light array is mounted to the bottom of the M100 drone as seen in Figure 4.2. The array is comprised of 10 high-power LEDs (Engin part number LZ4-40UB00-00U5) that produce light at approximately 395nm wavelength. The light produced by each LED is narrowed through a lens (Engin part number LLNS-2T06-H) to maximize their range. The heat produced by the LEDs is dissipated through an aluminum heatsink. The LEDs are powered by a set of four 3.7V Lithium

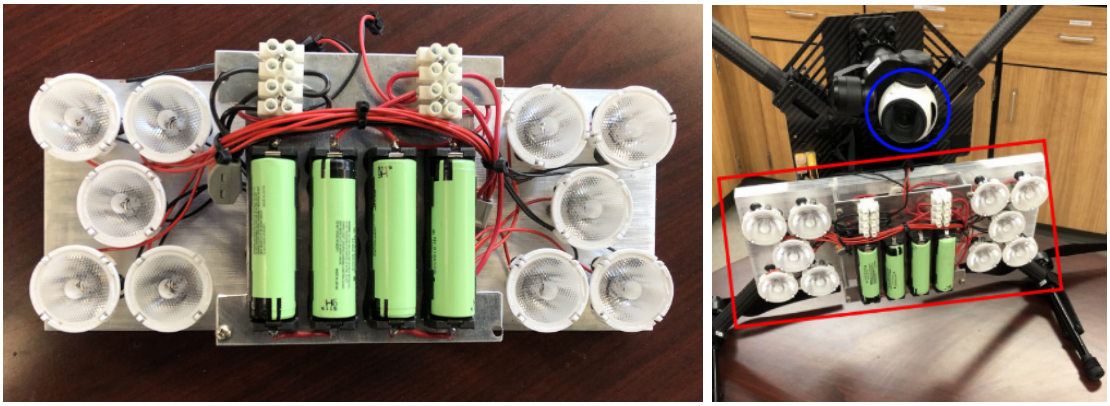

Figure 4.2: Ultraviolet illumination system attached to the bottom of the Matrice 100.

Ion batteries with 3400mAh capacity. The power to the LEDs is controlled via remote control allowing mid-flight use of the LED array. This array was constructed by Scott Wolford at the USDA Appalachian Fruit Research Station in Kearneysville, WV.

4.3 Field Data Acquisition

Table 4.4: Description of the datasets.

Dataset	**Acquisition Date**	**# of Frames**	**# of Insects**	**# of Detections**	**Flight Area** (m^2)	**Wind Speed** (km/h)	**Illumination** (lumens)	**Lat. °**	**Lon.°**
A	Sep 2018	1,869	23	587	3,000	6.0	0.03	43.037	-87.929
B	Nov 2018	8,878	39	1,473	575	1.4	0.10	43.037	-87.929
C	June 2019	19,023	82	1,949	575	2.7	0.30	43.025	-87.945
D	June 2021	3,856	125	3,005	N/A	N/A	N/A	N/A	N/A

Data was collected in three test locations – Mitchell Park in Milwaukee, the Marquette University Quad, and an apple orchard in the West Virginia USDA Appalachian Fruit Research Station (AFRS). Datasets A and B were acquired at the Marquette University Quad, Dataset C was acquired at Mitchell Park, and Dataset D was collected at the AFRS research orchard. Dataset A was collected in September

2018, Dataset B was collected in November 2018, Dataset C was collected in June 2019, and Dataset D was collected in June 2021. Environmental conditions, number of insects, and other relevant information for each dataset can be seen in Table 4.4. Dataset D was collected and provided by the USDA and does not include the flight logs, as discussed in Section 4.2.1.1.

Figure 4.3: The flight path of the drone. From left to right the flight path for datasets A, B, then C.

Dataset A is meant to represent an "ideal" data collection scenario, where conditions were favorable to detecting insects; the grass was recently cut and there is no source of noise in the images. Dataset B is significantly more challenging than Dataset A, as the presence of leaves on the ground introduce a large source of visual noise in the data. Figures 4.3 show the flight paths of datasets A, B, and C (note that the background images were not acquired at data acquisition time, so the leaves on Dataset B cannot be seen in the images). Dataset C, while more difficult than dataset A, contains less noise than dataset B. Because dataset C was acquired at a public area, other sources of noise such as trash are present, but because it was

collected during the summer, there is no noise from leaves. However, the grass height varies, and the ground was wet from recent rainfall, causing the ground to generate light reflections in several places. The final difference between datasets A and B vs. C is that for dataset C the latitude and longitude of each insect was determined using an RTK GPS. Dataset D was collected by the United States Department of Agriculture and is significantly different from the other datasets. In Datasets A, B, and C the insects are on the ground, but in D they are placed in the canopies of the trees at the USDA research orchard. Hence, the drone is flown parallel to the rows of trees, with the camera aimed at the trees at an angle of approximately 45°, rather than over the insects as in the other three datasets. Because the insects are not below the UAV in Dataset D, the flight logs could not be used for the constrained K-Means (Section 5.6.2) or Hungarian (Section 5.7.2.1) algorithms used in this thesis for global insect localization. Finally, the fluorescent powder in dataset D is blue instead of orange. Dataset C is notable for having the longest flight time, with around 13 minuets of data recorded. Despite being approximately one fifth the length of Dataset C, Dataset D contains more insect detections.

Factors such as the need to acquire a "§107.29(a)(2) – Operation at night waiver" from the FAA [51], limited periods of weather conducive for drone flying, and the Coronavirus pandemic, limited our ability to collect additional datasets.

4.3.1 Flight Path Configuration

The drone can be operated and configured using the DJI Ground Station Pro app. This app is created and maintained by the Matrice 100 drone's manufacturer, DJI. The ground station prop app provides the ability to plan a mission before taking off and to automatically fly the drone according to the mission. Mission planning includes setting points via latitude and longitude, determining the turning angle of the drone, its altitude, among other parameters. We chose to use the WayPoint Route

mission mode for collecting data. To do this, we simply plotted the desired drone route, including its altitude by setting a number of waypoints for the drone to reach and the speed of the drone. When flying the mission, the drone attempts to travel as close as possible to the set waypoint map. The latitude and longitude waypoints are calculated by determining a start-point and end-point for the mission. Then a python program was created to generate a square-wave pattern for the drone to fly to capture the data between these points. All of the insects to be detected must be placed inside this box. The width of the squarewave-pattern is determined by the desired altitude of the flight, and the known light dispersion of the UV illumination array attached to the drone. If the program has told the drone must fly at a higher altitude it simply scales the squarewave to increase the distance between the parallel lines. Once the squarewave pattern is defined, a set of coordinates that fill out the box are calculated. The algorithm begins by moving directly in a cardinal direction (North, South, East, or West) until either the latitude or longitude of the drone matches the latitude or longitude of the endpoint of the corresponding line. The drone travels along the furthest dimension of the data acquisition region. This helps minimize the number of waypoints the drone is required to make. The drone then rotates 90° to face the next end point, travels the determined width of the squarewave, rotates 90° to face the opposite direction of initial travel, and moves in a straigh line until it reaches either the latitude or longitude of the start-point. The pattern then repeats itself with alternating rotation directions so as to follow each line of the squarewave pattern. A waypoint is recorded for each point where the algorithm determines that the drone must perform a rotation. The algorithm terminates when the drone has approximately reached the endpoint. An example of the drone's path can be seen in Figure 4.3.

4.3.2 Data Collection Procedure

In order to gather data, a flight plan must be created and a data collection flight must be performed. This allows us to gather data including both flight logs and video frames. Sample bugs marked with fluorescent powder are placed throughout the data collection area. Next, the luminosity and other relevant environmental conditions of the field are noted. The drone flies in a pre-defined squarewave pattern 10 meters above ground level as seen in Figure 4.3 while recording a video with the camera oriented perpendicular to the flight plane. Following this procedure, we collected a total of four datasets from four different flights: Datasets A, B C, and D, whose characteristics are described in Table 4.4. Sample frames from each data set can be seen in Figure 4.4. An example of the insect appearance in the dataset can be seen in Figure 4.5.

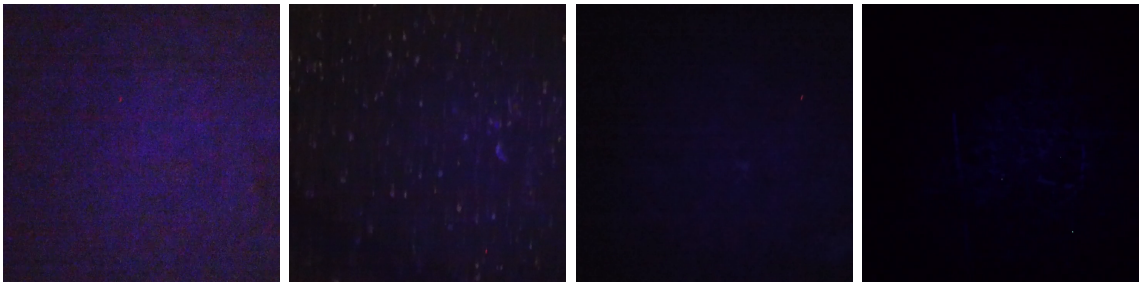

Figure 4.4: **Best viewed in color:** from left to right: the A,B, C, and D datasets.

4.3.3 Data Labeling

The ground truth was manually labeled using a custom-built annotation tool. The order of the images was randomized, and each insect seen was marked with a bounding box. Randomizing was necessary as when displayed in chronological order the annotator may make an inference and put a bounding box even if the insect is not visible in the frame. The insects were then associated by hand so that if an insect

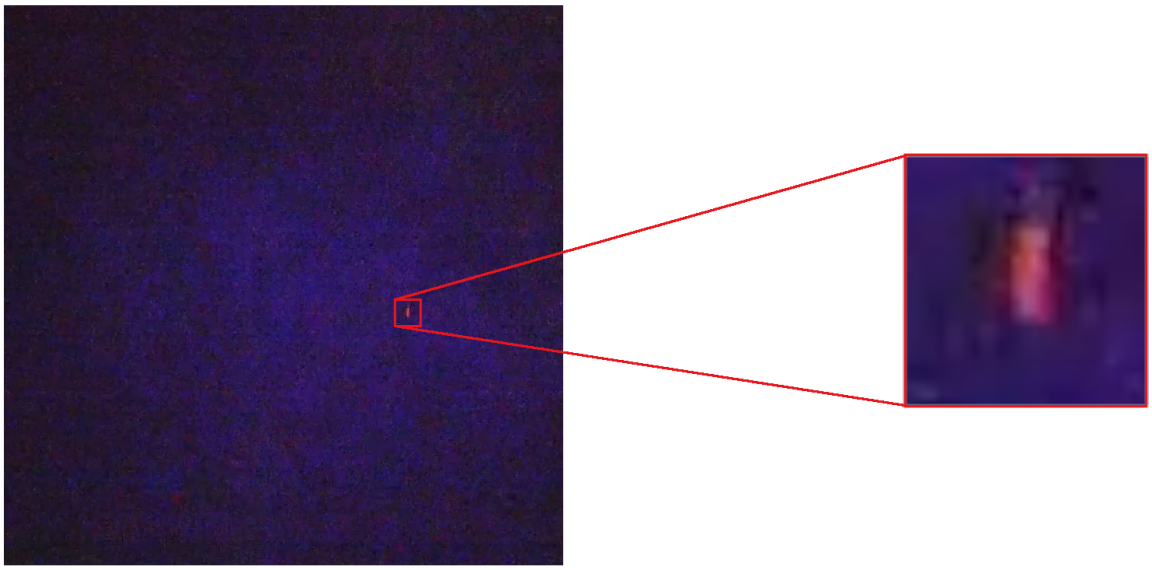

Figure 4.5: An example of what an insect looks like in dataset C.

was seen in multiple frames it had the same identifier in all of those frames.

CHAPTER 5

INSECT DETECTION PIPELINE AND EXPERIMENTAL ANALYSIS

The full algorithm pipeline can be seen in Figure 5.1. Implementation details can be found in Section 5.9. We begin by creating a flight plan as described in Section 4.3.2. We use the frames to train an auto encoder to reconstruct the image and calculate the reconstruction error. Using connected component analysis we group the pixels into frame by frame insects and using MHT we group these into global insects. Our proposed invasive insect detection network takes a video sequence as input and outputs the latitude and longitude coordinates of the detected insects. We use video data recorded from the DJI Matrice 100 drone with ultra violet (UV) lights mounted to the bottom. The illumination system is discussed in detail in Section 4.2.5. We train the CVAE on unlabeled data from each dataset. Because there are significantly more background pixels than insect pixels in each image, the CVAE should learn the expected appearance of the background. Hence, this approach allows us to remove the background (everything that is not an insect) from each frame. Using the drone's internal measurement unit's location information, the orientation of the drone calculated by a Kalman Filter [63], and camera calibration information, we can project the image coordinates of the insect detections into the global coordinate frame to find the corresponding latitude and longitude of each detection. Finally, using Multiple Hypotheses Tracking (MHT [42]) we can track each detection across video frames. Without this each frame in which an insect appears would be treated as a unique insect. However, if an insect appears in nearby locations over a sequence of frames, it is likely the same insect. MHT analyzes the observed trajectory of the insect within the video frames to determine if it should be treated as one or multiple insects.

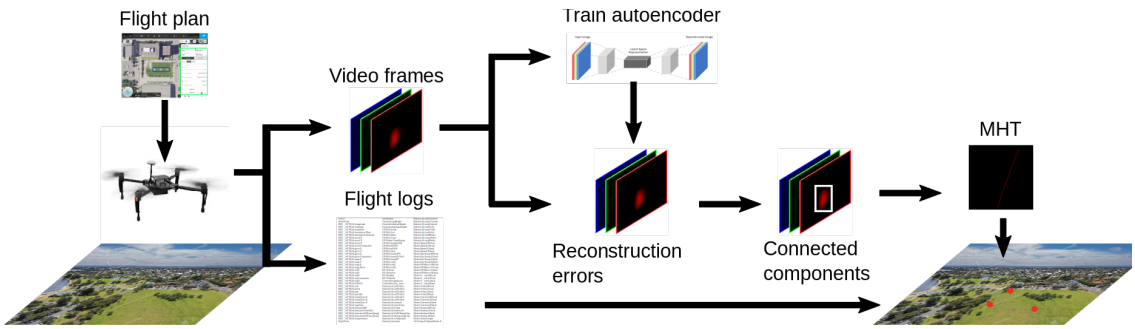

Figure 5.1: Proposed pest insect detection system architecture. By taking the predicted location of the insects relative to the drone as determined by the CVAE and the location of the drone from the flight logs we can calculate an accurate drone location.

5.1 Convolutional Variational Auto Encoders

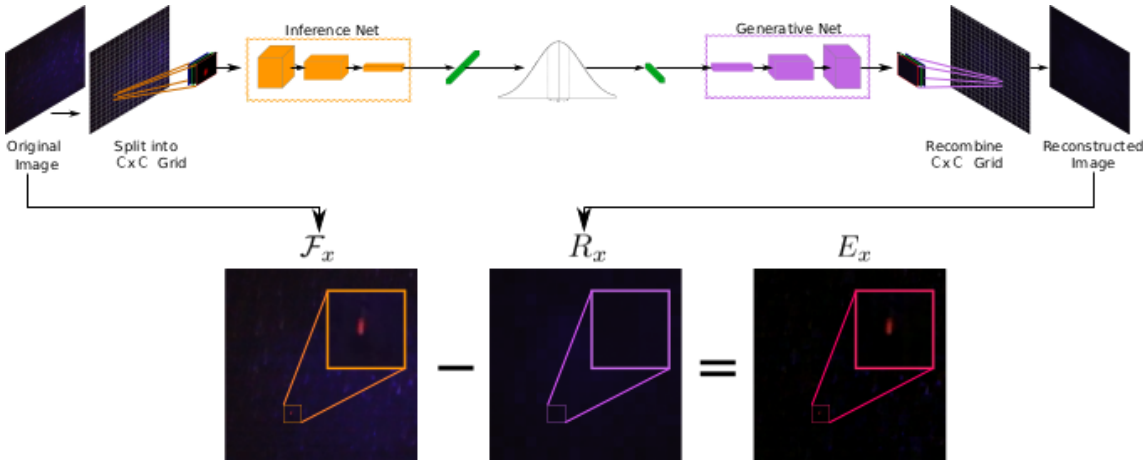

Figure 5.2: **Best Viewed In Color:** Illustration of the CVAE pipeline. The image is tiled, split into its color channels and input into the Inference Net. Then, the output of the inference net is used to create the z-dimension with a Gaussian distribution. Finally, the Generative net uses the z-dimension to re-create the image's red, green and blue component of the image and stitches each sub-image back into the full output image.

Figure 5.2 summarizes our Convolutional Variational AutoEncoder. The AutoEncoder is trained on video frames from our flight data. Its reconstruction (R_x) is then used to calculate the reconstruction error (E_x).

5.1.1 Pre Processing the data for the CVAE

We train a separate Convolutional Variational AutoEncoder for each data set. To do this, from each image, we select a region of interest (ROI) of $p_r \times p_r$ pixels at the center of the original image. This is necessary because most pixels outside the region of interest are not within the range of the illumination array and are hence completely dark, containing no visible information. Removing the area outside the ROI leads to significant reductions in the processing time for the CVAE while improving its reconstruction performance. Then, the ROI is tiled into a grid of $c \times c$ cells $\mathcal{F}_{x,y} \in \mathbb{R}^{n_g \times n_g \times n_c}$, where n_g is the resolution of the grid cells, which is given by $n_g = \lfloor p_r/c \rfloor$, $n_c = 3$ is the number of channels of the ROI image, and $x = 1, \ldots, c$, $y = 1, \ldots, c$. As indicated in Fig. 5.3, to minimize blocking artifacts, we follow the strategy proposed in [10] and allow a small overlap among each tile and its immediate neighbors.

5.1.2 CVAE Processing

Let $\mathcal{F}^i_{x,y}$ correspond to the grid cells comprising the i-th image in the dataset. For each dataset $\mathcal{F} = \left\{\mathcal{F}^i_{x,y}\right\}_{i=1}^N$, where N is the number of images in the dataset, a CVAE is trained to learn the mapping

$$C(\mathcal{F}^i_{x,y}) = G(z), \tag{5.1}$$

where $z \sim \mathcal{N}(\mu, \Sigma)$ and $\Sigma = \text{diag}(\sigma)$ is an $n_l \times n_l$ diagonal matrix whose elements are given by the vector $\sigma \in \mathbb{R}^{n_l}$. The latent features $[\mu, \sigma] \in \mathbb{R}^{2 \times n_l}$ correspond to the mean and variance of a multivariate normal distribution. The latent features are generated by the inference network $I : \mathbb{R}^{n_g \times n_g \times n_c} \Rightarrow \mathbb{R}^{2 \times n_l}$, which maps the input grid cell $\mathcal{F}^i_{x,y}$ onto the latent feature space, i.e., $[\mu, \sigma] = I(\mathcal{F}^i_{x,y})$. Finally, $G(z)$ is the generative network $G : \mathbb{R}^{n_l} \Rightarrow \mathbb{R}^{n_g \times n_g \times n_c}$, which computes the reconstructed image

$$R^i_{x,y} = G(z). \tag{5.2}$$

Hence, the CVAE learns the overall function

$$C : \mathbb{R}^{n_g \times n_g \times n_c} \Rightarrow \mathbb{R}^{2 \times n_l} \Rightarrow \mathbb{R}^{n_l} \Rightarrow \mathbb{R}^{n_g \times n_g \times n_c} \tag{5.3}$$

that maps input grid cells into their corresponding reconstructions. The output grid cells can be placed back into their original grid positions, such that

$$R_x = D(R^i_{x,y}), \tag{5.4}$$

where D is a function that puts each $R^i_{x,y}$ back into the correct (x,y) position in the reconstruction of image i. When there is overlap in the grid pixels, D averages the pixels for that position.

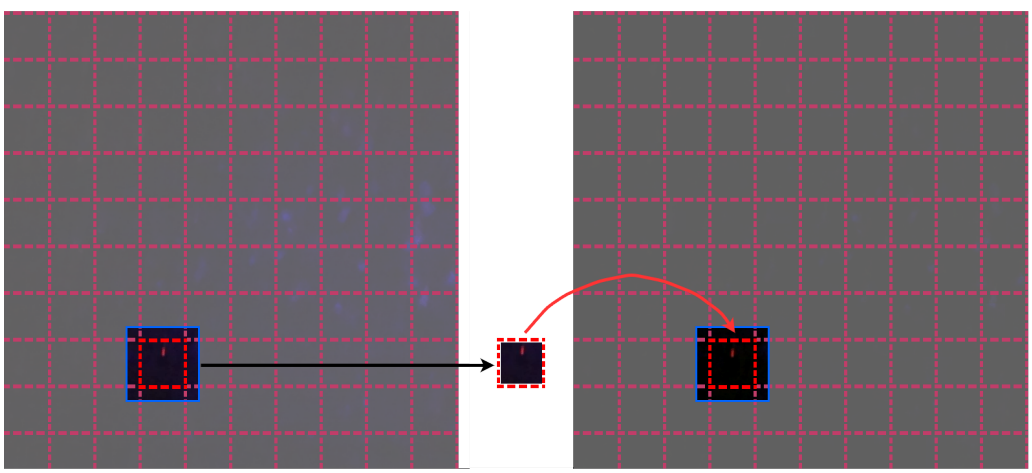

Figure 5.3: Illustration of how the image is tiled into smaller images of 64×64 pixels before being encoded and decoded by the CVAE.

Since the vast majority of the pixels in each grid cell correspond to the background, the CVAE fails to learn to recreate insects. Hence, we expect the reconstruction error

$$E_x = \mathcal{F}_x - R_x \tag{5.5}$$

to be comprised mostly of anomalies such as insect pixels. We have empirically determined that the reflection of the UV light by the coating powder on the insects

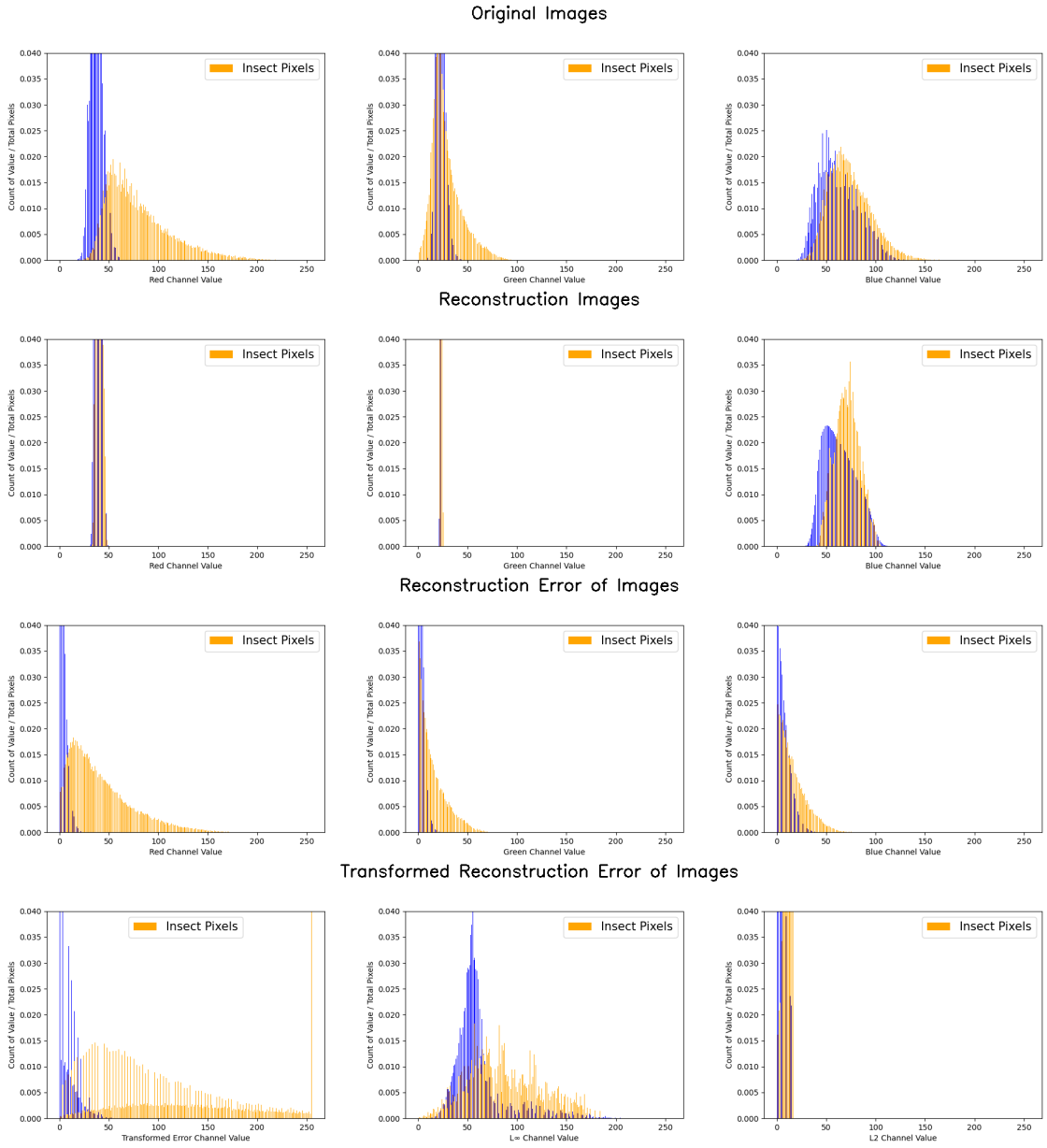

Figure 5.4: The histograms of each channel of the data, reconstruction, reconstruction error, and transformed reconstruction error of dataset A.

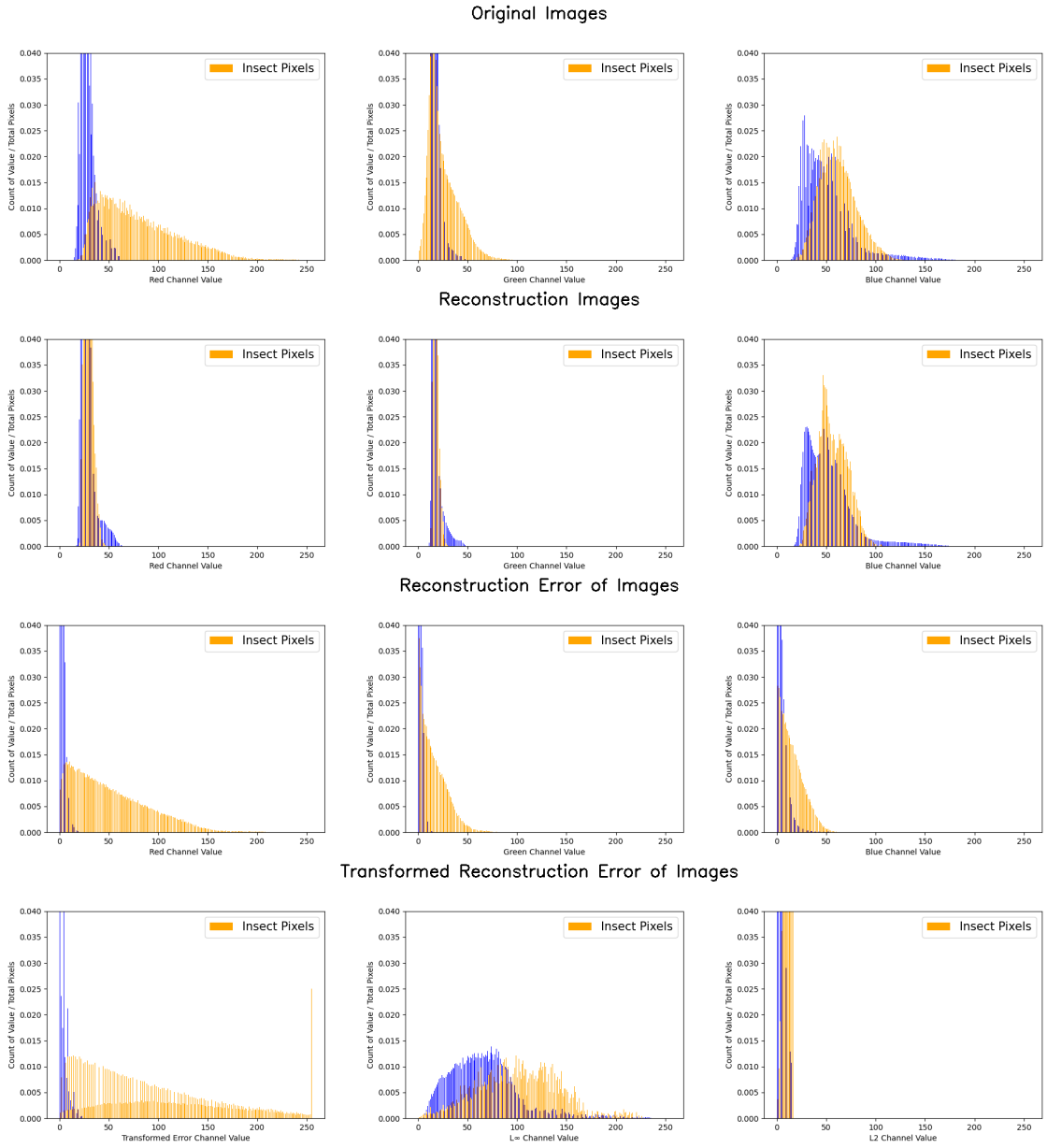

Figure 5.5: The histograms of each channel of the data, reconstruction, reconstruction error, and transformed reconstruction error of dataset B.

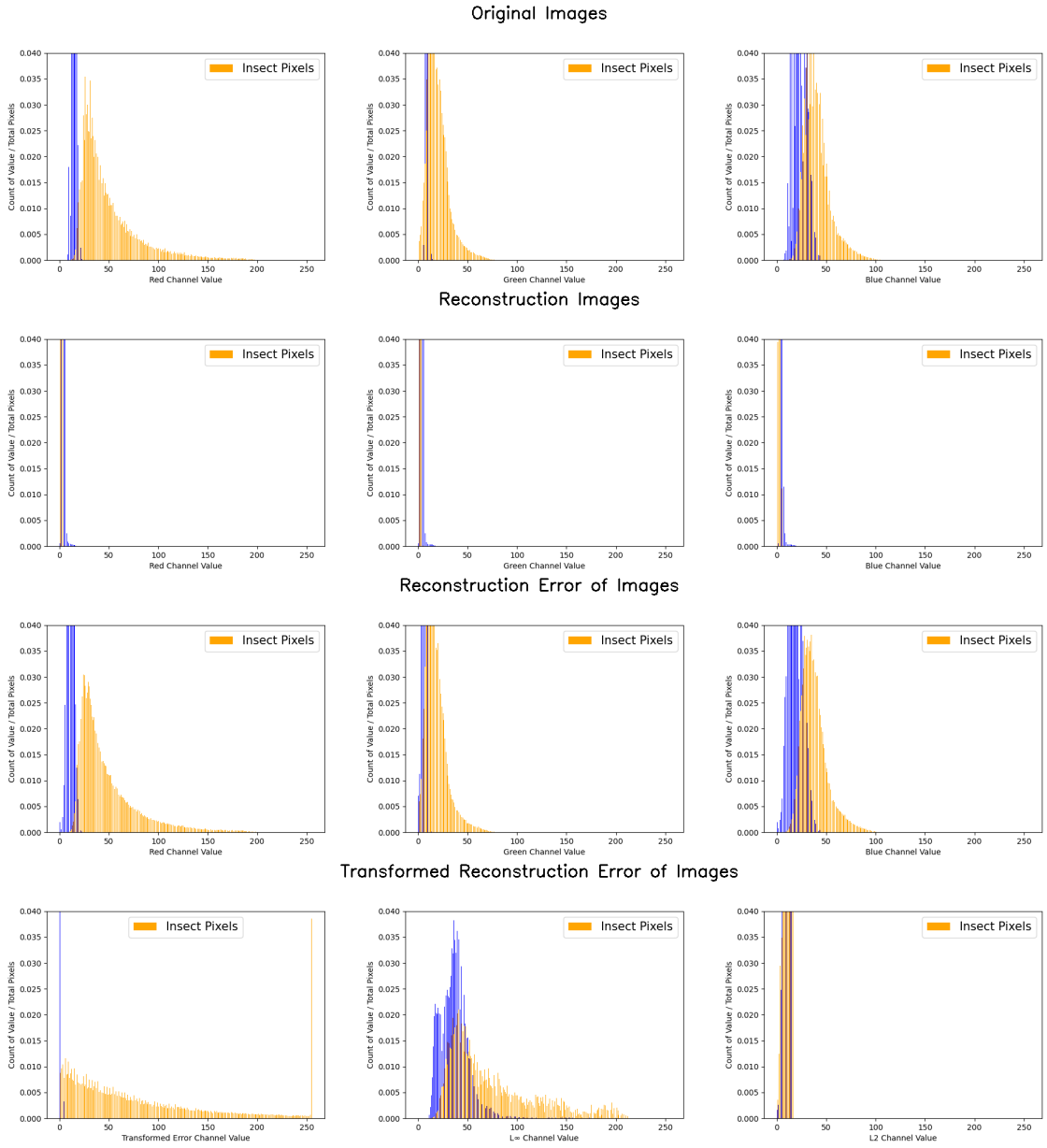

Figure 5.6: The histograms of each channel of the data, reconstruction, reconstruction error, and transformed reconstruction error of dataset C.

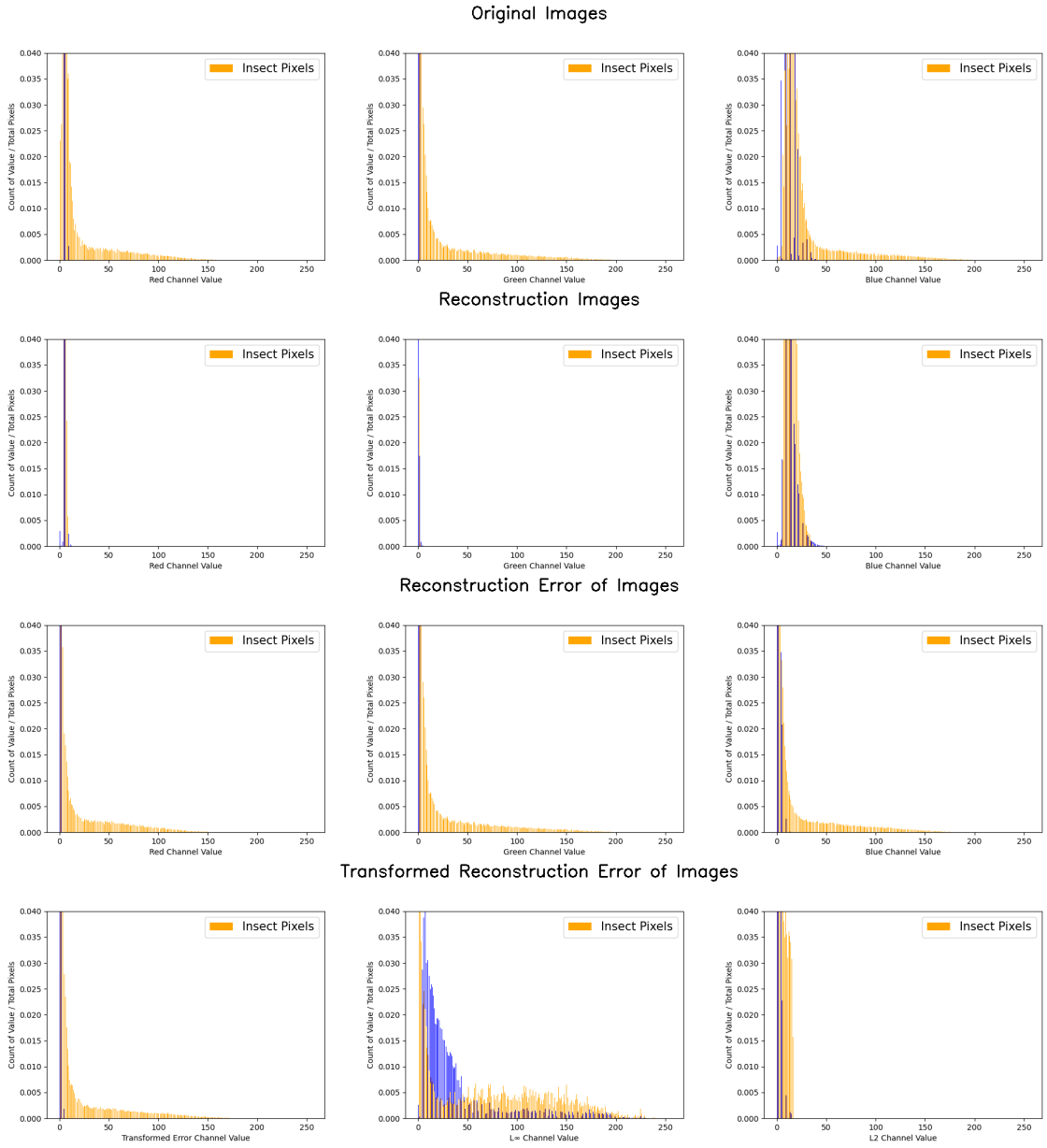

Figure 5.7: The histograms of each channel of the data, reconstruction, reconstruction error, and transformed reconstruction error of dataset D.

is most prominent in the red channel, whereas background noise is present primarily in the blue channel for datasets A,B, and C. This can be seen in Figures 5.4, 5.5, and 5.6. To take advantage of this knowledge, we propose a transformed reconstruction error that prioritizes the red channel:

$$\hat{E}_x = 2 \cdot E_{xr} - E_{xb}, \tag{5.6}$$

where $E_{xr} \in \mathbb{R}^{n_w \times n_h}$ and $E_{xb} \in \mathbb{R}^{n_w \times n_h}$ are the red and blue channels of the reconstruction error E_x.

Dataset D is different from the other datasets in a number of ways. Notably, the insects are not coated with orange fluorescent powder. Instead, these insects are marked with a blue fluorescent coating. To compensate for that we change the transformed reconstruction error to no longer prioritize any channel for this dataset. Instead, we use the average reconstruction error over the three channels, i.e.,

$$\hat{E}_x = \frac{E_{xr} + E_{xb} + E_{xg}}{3}, \tag{5.7}$$

where $E_{xr} \in \mathbb{R}^{n_w \times n_h}$ and $E_{xb} \in \mathbb{R}^{n_w \times n_h}$ and $E_{xb} \in \mathbb{R}^{n_w \times n_h}$ are the red, blue, and green channels of the reconstruction error E_x.

5.1.3 CVAE Network Architecture

As explained in Chapter 3, a CVAE is composed of two stages: The Inference Network and the Generative Network. The Inference Network takes an image as input and learns to represent it using a lower dimensional latent feature composed of an array of means and standard deviations. Next, a new array is created by sampling from a multi-variate Gaussian distribution whose means and standard deviations are given by the latent feature vector. Then, the latent feature vector z is used as the input for the Generative Network, which learns to recreate the original image based on the information in the z vector. The architecture of the Inference Network used

in this thesis can be seen in Table 5.1, and the Generative Network is described in Table 5.2. Definitions for items in the table can be found in Table 5.3

Table 5.1: Architecture of the inference network.

Layer	**Filter**	**Kernel size**	**Strides**	**Activation**	**Output size**
Input	n/a	n/a	n/a	n/a	(64,64,3)
2D Conv.	96	3	(2,2)	relu	(31,31,96)
2D Conv.	192	3	(2,2)	relu	(15,15,192)
Flatten	n/a	n/a	n/a	n/a	(43200
Dense	n/a	n/a	n/a	linear	(3900)

Table 5.2: Architecture of the generative network.

Layer	**Filters**	**Kernel Size**	**Strides**	**Activation**	**Output Size**
Input	n/a	n/a	n/a	n/a	(1950)
Dense	n/a	n/a	n/a	linear	(24576)
Reshape	n/a	n/a	n/a	n/a	(16,16,96)
2D Conv.^T	192	3	(2,2)	relu	(32,32,192)
2D Conv.^T	96	3	(2,2)	relu	(64,64,96)
2D Conv.^T	3	3	(1,1)	linear	(64,64,3)

5.1.4 Training Procedure

The CVAE is trained in an unsupervised manner. That is, the network is trained on all of the frames in the dataset without prior knowledge of which frames contain insects. Because each frame is split into a $c \times c$ grid of $n_g \times n_g$ pixels (see Section 5.1.1), even when a frame does contain an insect, the majority of the grid cells do not. Therefore, the overwhelming majority of the training data does not contain an insect. Due to the unbalanced nature of the insects in the data, the network does

Table 5.3: Definitions for layers in the CVAE.

Term	Definition
Flatten	Changes a 1D Array into a 2D Array
2D Conv.	2D Convolutional Layer
2D Conv.^T	2D Convolutional Transpose Layer
Dense	Takes a 1D input and makes a 1D Output.
Reshape	Changes a 1D Array Input into a 2D Array

not learn how to represent insects. Instead, the network learns only how to represent the background of the images. The CVAE is retrained from scratch for each dataset with a batch size of b_s for e epochs. The network is trained to maximize the evidence lower bound (ELBO) on the marginal log-likelihood [64], i.e.,

$$\mathcal{L}(x, z) = -\left(\log p(x|z) + \log p(z) - \log q(z|x)\right), \tag{5.8}$$

where $\log p(x|z)$ is the log-likelihood of the training sample x, $\log p(z)$ is the log probability of z, which is normally distributed with $\mu = 0$ and $\sigma = 0$, and $\log q(z|x)$ is the log probability of z conditioned on the input x, which is normally distributed with mean μ and variance σ, which are learned based on the data distribution.

5.2 Determining Insects from CVAE output

Because the CVAE outputs an error that is transformed for each pixel as described in Equation 5.6, it does not provide the actual insect locations in each image. Instead, we must find a way to determine which pixels in each image correspond to insects (Section 5.2.1), and a way to group these pixels into different insects if there are multiple insects in an image (Section 5.2.2).

5.2.1 Thresholding Error to Find Insect Pixels

In order to find the insects using the transformed error (E_x) described in Equation 5.6, we chose to determine a threshold (e_{thr}) where any pixel greater than e_{thr}

Algorithm 3: Insect threshold determination.

```
Input  : Set of transformed reconstruction errors for dataset x: Ê_x
         A probability of a pixel not being a insect pixel: P_insectpixel
Output: Integer threshold e_thr with a value in [0, 255]
V_pixels = array of length 255;
T_pixels = 0;
for e ∈ Ê_x do
    for pixel ∈ e do
        V_pixels[pixel] += 1;
        T_pixels +=1;
for p ∈ V_pixels do
    p = p/T_pixels
cdf = array of length 255;
cdf[0]= V_pixels[0];
e_thr = 0;
for i ∈ [1 . . . 255] do
    cdf[i] = cdf[i-1]+V_pixels[i];
    if cdf[i] < P_insectpixel then
        e_thr = i;
    else
        break;
return e_thr
```

is assumed to belong to an insect, and any pixel less than e_{thr} is assumed otherwise. Using Algorithm 3 we calculate the insect threshold. To do this we find a Probability Density function (PDF) for E_x over the entire dataset under consideration. Then we turn this PDF into a Cumulative Density Function (CDF) using Equation 5.9

$$CDF(x) = \sum_{i=1}^{x} PDF(i). \tag{5.9}$$

Finally, we determine e_{thr} by finding the largest $CDF(e_{thr})$ that is still smaller than a probability $P_{\text{insectpixel}}$. Any pixel for which $E_x > e_{thr}$ is added to the set of insect pixels.

Algorithm 4: Creating insect observations using the reconstruction error.

```
Input  : Ê_x
         e_thr
Output: Set of bouding boxes
out = [] for e ∈ ê_x do
    for pixel ∈ e do
        pixel_value = the integer value of the pixel;
        if pixel_value > e_thr then
            pixel_value = 1;
        else
            pixel_value = 0;
        end
    end
    cca = connectedComponentAnalyisis(e);
    for ConnectedComponent ∈ cca do
        x = ConnectedComponent.min_X;
        y = ConnectedComponent.min_Y;
        width =
          ConnectedComponent.max_X-ConnectedComponent.min_X;
        height =
          ConnectedComponent.max_Y-ConnectedComponent.min_Y;
        out.append([x, y, width, height])
    end
end
return out
```

5.2.2 Connected Components Analysis to Group Insect Pixels

Once the set of insect pixels at a given frame is found, we need to determine which pixels belong to the same insect. Because multiple insects can appear in one frame, we cannot assume that all the pixels in the frame belong to the same insect. Instead, we group the pixels into insects using connected components analysis, as described in Algorithm 4. Connected component analysis takes groups of pixels and, if any of its neighbors are also insect pixels, associates itself with those neighbors. Then, it repeats itself for each disjoint group of pixels. It continues this process until all pixels that both touch each other and are insect pixels have a common

identifier. For each group of insect pixels, we extract a bounding box representing the corresponding insect detection.

5.3 Associating Detections

Associating insect detections is a challenging and important aspect of this project. We want to associate detections that belong to the same global insect. If an insect appears in the video, it is highly likely that it will appear in several frames. So, if we can associate the detections to a global insect we can determine its global position. When labeling the ground truth, we give each insect a global insect identifier. This allows us to evaluate the methods we implemented to track the insects and compare them to the hand-labeled insect identifiers using the multiple object tracking (MOT) metrics as discussed in Section 2.4.2 [43]. By accurately associating insect detections together, we can improve the quality of the detections and effectively count the number of insects present in the field. The proposed method does have a shortcoming: if an insect appears, leaves for a significant amount of time, and reappears later, there is a strong chance the algorithm will believe that it is a new insect. However, it is likely that any human walking through a field with a handheld UV light would make the same mistake.

To associate multiple detections of the same insect into a single consistent observation, we evaluated two methods. Our first approach consisted of projecting the insects back to a real world location and using constrained k-means clustering on groups of detections to try and determine if they corresponded to a single insect or to multiple insects that happened to be nearby. Since we did not obtain satisfactory performance with that strategy, we decided to employ the Multiple Hypothesis Tracking (MHT) algorithm [42] to associate the detections. These two strategies are described in detail in the following subsections.

Algorithm 5: Applying constrained K-Means to real data.

Input : Set of detection's bounding box's centers with their x, y, and frame values, C_x, all within D_{min} of at least one other observation in the group

Output: Set of real-world insect locations, loc_x

1 $\mathcal{V} = \emptyset$;
2 **for** $k = 1$ *to* K_m **do**
3 $\quad \mathcal{V} = \mathcal{V} \cup \textsc{constrainedKMeans}(k, C_x)$;
4 $loc_x = \text{argmax}_{v_k \in \mathcal{V}} \textsc{silhouetteScore}(v_k)$;

5.3.1 Constrained K-means Insect association

Constrained K-means clustering is implemented to associate insect detections. Constrained K-means is discussed in detail in Section 3.3.1, but the details of this implementation can be found here. By mapping the insect detections to their real-world coordinates (as discussed in Section 5.5, distances between each observation are found with a small margin of error. Then, any group of insects each within a minimum distance (D_{min}) of each other are inserted into constrained k-means. If the insects' detections share a frame constrained k-means cannot give them the same global insect ID. This is repeated for each k value between 1 and K_m, and the clusters with the best silhouette score is chosen. This process is shown in Algorithm 5.

5.3.2 Multiple Hypotheses Tracking

In MHT each sequence of detections (i.e., a track) is assigned a score $S^l(k)$, which is defined as follows:

$$S^l(k) = w_{mot} S^l_{mot}(k) + w_{app} S^l_{app}(k), \tag{5.10}$$

where $S^l_{mot}(k)$ is the motion score, $S^l_{app}(k)$ is the appearance score, and w_{mot} and w_{app} are their relative weights on the overall computation of the appearance score. However, because there is little variability in the appearance of the insects, we disregard

Figure 5.8: Illustration of the insect tracks by the MHT algorithm on two image sequences, both from datasets C.

the appearance score so that the overall score becomes simply

$$S^l(k) = S^l_{mot}(k). \tag{5.11}$$

The motion score is defined as:

$$S^l_{mot}(k) = \ln\left(\frac{p(y_{i_{1:k}}|i_{1:k} \subseteq T_l)}{p(y_{i_{1:k}}|i_{1:k} \subseteq \emptyset)}\right), \tag{5.12}$$

where $i_{1:k}$ represents a set of observations $i_1, i_2, ..., i_k$, T_l represents the target hypothesis, and $y_{i_{1:k}}$ is the set of location measurements for the set of observations $i_{1:k}$. In other words, $p(y_{i_{1:k}} i_{1:k} \subseteq T_l)$ represents the probability of the set of measurements happening, and $p(y_{i_{1:k}}|i_{1:k} \subseteq \emptyset)$ represents the probability of a set of measurements not being associated. After generating many hypotheses, we can calculate the track score for each hypothesis and take the highest available. Figure 5.8 illustrates the resulting insect trajectories obtained using MHT.

5.4 Drone Heading Estimation Using a Kalman Filter

Because data provided by the DJI M100 Drone does not include heading information, we must find a way to calculate it. We discuss the DJI M100 log file in

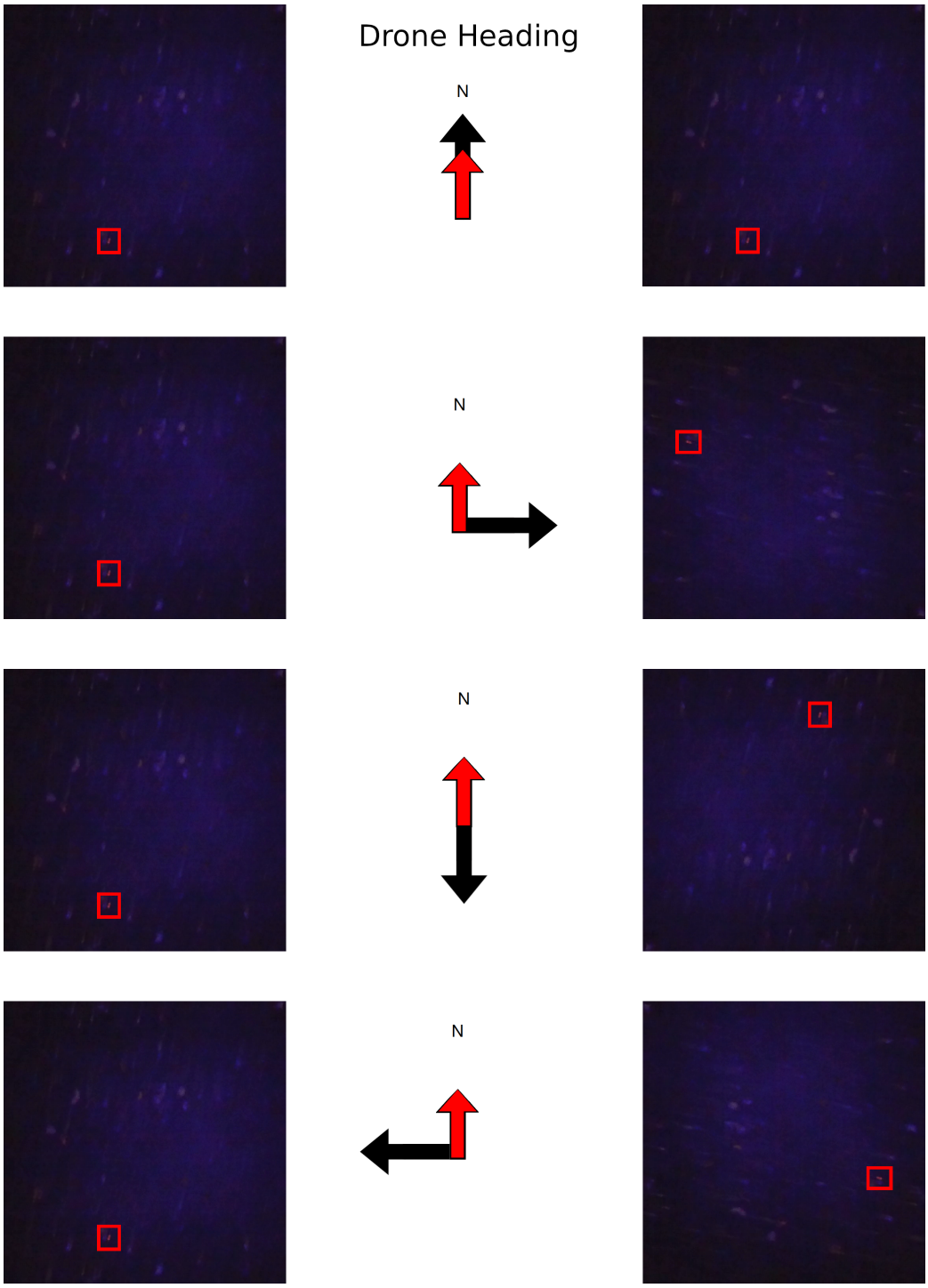

Figure 5.9: **Best viewed in color:** Visualization of applying the Kalman headings to an observation

more detail in Section 4.2.1.1. To map the insect detections to a global coordinate frame, we must know the drone location and orientation. Since the orientation is not available in the flight logs, we use the location information to estimate it. To do so, we implemented a Kalman filter to estimate the drone heading [65]. The Kalman filter uses a state vector given by

$$s_t = \begin{bmatrix} p_t \\ \omega_t \end{bmatrix}, \tag{5.13}$$

where $p_t = \arctan(\Delta_x/\Delta_y)$ is the arc tangent of the direction of the motion of the drone along the x and y coordinates, so that p_t is the current estimated angle of the drone where p_0 corresponds to north. ω_t is the current estimated angular velocity. We define F (the system transition matrix) and H (the observation matrix) as:

$$F = \begin{bmatrix} 1 & 1 \\ 0 & 1 \end{bmatrix}, \tag{5.14}$$

$$H = \begin{bmatrix} 1 \\ 0 \end{bmatrix}. \tag{5.15}$$

The predicted state of the drone is given by

$$s_{t+1} = F \cdot s_t. \tag{5.16}$$

Figure 5.10 illustrates the drone heading estimated using this procedure for the flight corresponding to datasets A, B, and C.

5.5 Mapping Insect Detections to the Global Coordinate System

Using the position, heading and height of the drone as well as the distance and angle of the insects relative to the drone, it is possible to compute the insect's real-world location. We use the angular Kalman filter described in Section 5.4 to estimate the orientation of the drone. The application of the Kalman filter to an

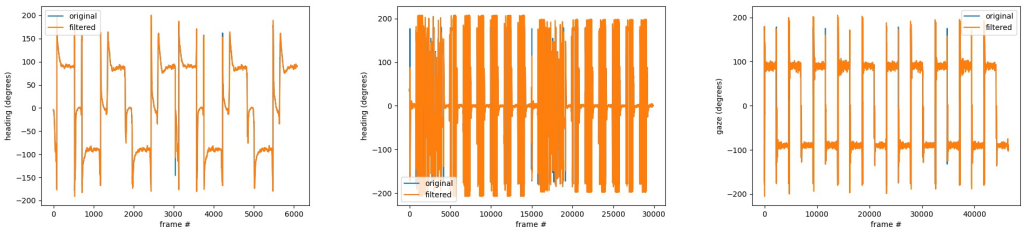

Figure 5.10: Estimated drone heading orientations generated using an angular Kalman filter. Datasets A, B and C from left to right.

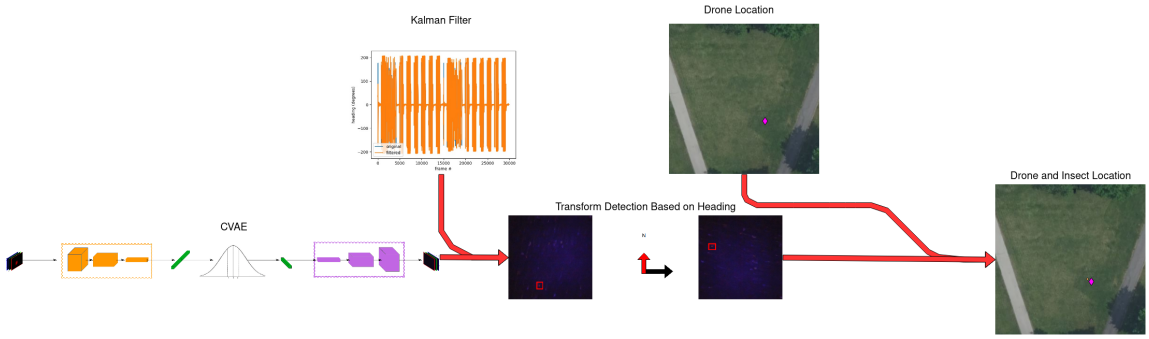

Figure 5.11: **Best viewed in color:** Visualization of applying the Kalman headings and drone location information to an observation

insect detection is illustrated in Figure 5.9. Then, by taking the real world coordinates of the drone as well as the angle and distance of the insect relative to the drone, the insect detection can be given in real-world coordinates using the following affine transformation

$$\begin{bmatrix} x_r \\ y_r \\ 1 \end{bmatrix} = \begin{bmatrix} s\cos\theta_t & s\sin\theta_t & x_d \\ -s\sin\theta_t & s\cos\theta_t & y_d \\ 0 & 0 & 1 \end{bmatrix} \begin{bmatrix} x_i \\ y_i \\ 1 \end{bmatrix}, \tag{5.17}$$

where (x_i, y_i) are the image coordinates of the insect, θ_t is the heading of the drone as provided by the Kalman filter, (x_d, y_d) are the world coordinates of the drone, and s is a scaling factor that represents the region covered by a pixel in meters at an elevation of H meters. An example of this in action can be seen in

Figure 5.11. Because the drone's real-world positional measurements and the Kalman filter are both imperfect there is some error in the real-world position assigned to each detection. Fortunately, there are typically several detections per insect that, when averaged, provide a more accurate location for the insect. To do this we use the MHT algorithm, as outlined in Section 5.3.2.

5.6 Baseline Insect Detection and Association Methods

We compare our proposed insect detection strategy with two methods previously proposed in [1] and [66] by Stumph et al. For completeness, this section provides a brief description of these methods. For additional details, the reader is referred to the corresponding references. We also compare the performance of our insect detection association mechanism described in Subsection 5.3.2 with a naive approach that uses the k-means algorithm to associate detections in multiple frames.

5.6.1 Insect Detection by Color Thresholding

Algorithm 6: Insect color threshold

Input : Cropped ROI video frame, $I_n^{(N)}$
Output: Binary image, $I_n^{(b)}$, where foreground pixels are insects

```
for Each pixel, p_i^(r) do
    if p_g < p_b < p_r and p_h > τ_h and p_s > τ_s and p_v > τ_v then
        p_i^(b) = 1;
    else
        p_i^(b) = 0;
```

Color thresholding looks at the color values of a pixel, and if they fit more into the group of "insect" pixels than the pixels around them, they are classified as belonging to an insect. This is outlined in Algorithm 6. The results of these can be seen in table 5.4.

5.6.2 Insect Detection Using K-Means Clustering

To avoid confusion, it is important to note that this is a different implementation of k-means that previously mentioned in this thesis. While it is the same algorithm, the use-case of this k-means implementation is significantly different. The K-Means method of finding insects relies on clustering pixels in each frame by their color values using the K-means clustering algorithm [67]. First, pixels with low values in either the hue or value color spaces, which represent the majority of the pixels in a frame, are filtered out using color thresholding. Then, K-means (with $k = 2$ to represent the background and the insect pixels) clusters each pixel by its color value. This process is iterated three times discarding the cluster with the smaller number of elements at each iteration. The remaining pixels are classified as belonging to insect clusters. This is discussed in detail in Section 3.2.3. This is outlined in Algorithm 1. The results of these can be seen in table 5.4

5.7 Experimental Results

We evaluate our results with two different methods. First, we evaluate the frame-by-frame performance of our insect detection algorithm. In the frame-by-frame study, each frame is analyzed to find any correct detections (True Positives), extra detections (False Positives), and missed detections (False Negatives). Second, we evaluate the tracking results. In the tracking results section, we analyze how well the insects are tracked using MOT metrics.

5.7.1 Frame-by-Frame Analysis

By performing connected components analysis, we can group neighboring pixels together to create bounding boxes representing each insect S as seen in Algorithm 4. Then, an intersection over union (IoU) calculation is performed on each

Table 5.4: CVAE results compared to baseline methods

Dataset	Method	Precision	Recall	F1
A	K-Means	**0.95**	0.75	0.84
	Color Threshold	0.80	0.52	0.63
	Auto Encoder	0.78	0.81	0.79
	Auto Encoder +MHT	0.81	**0.88**	**0.85**
B	K-Means	0.40	0.50	0.44
	Color Threshold	0.65	0.40	0.50
	Auto Encoder	**0.87**	0.76	**0.81**
	Auto Encoder +MHT	0.72	**0.93**	**0.81**
C	K-Means	0.83	0.40	0.54
	Color Threshold	0.85	0.30	0.44
	Auto Encoder	**0.88**	0.63	0.73
	Auto Encoder +MHT	0.85	**0.67**	**0.74**
D	K-Means	N/A	N/A	N/A
	Color Threshold	N/A	N/A	N/A
	Auto Encoder	0.75	0.92	0.83
	Auto Encoder +MHT	**0.78**	**0.96**	**0.85**

bounding box compared to a hand-labeled ground truth. IoU is calculated by

$$IoU = \frac{GT_l \cap D_l}{CT \cup D}, \tag{5.18}$$

where GT_l is a ground truth detection, and D_l is a observed detection. Traditionally, an IoU of greater than 50% is considered a true positive. However, because of the small size of the insects using a IoU of 50% or better results in a significant amount of misclassifications as seen in Figure 5.12.

5.7.2 Tracking Results

We evaluate the performance of our insect detection methods using two methods. First, we use the MOT metrics discussed in Section 2.4.2. These results can be seen in Table 5.5. Both the MHT and the constrained K-means results for Datasets A, B, and C are shown in Figures 5.16, 5.17, and 5.18. Dataset D is not visualized

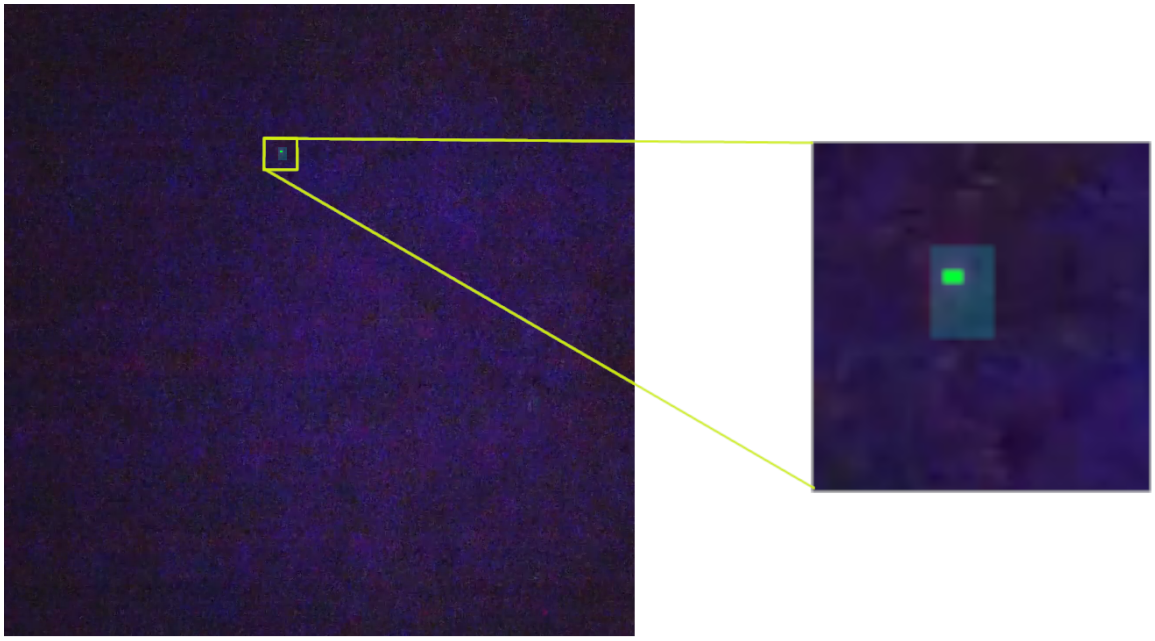

Figure 5.12: A small insect detection inside a bounding box. Here the IOU is below the traditionally used IOU threshold of 0.5.

because there are no flight logs for that dataset, which precludes the application of the association method based on constrained K-means.

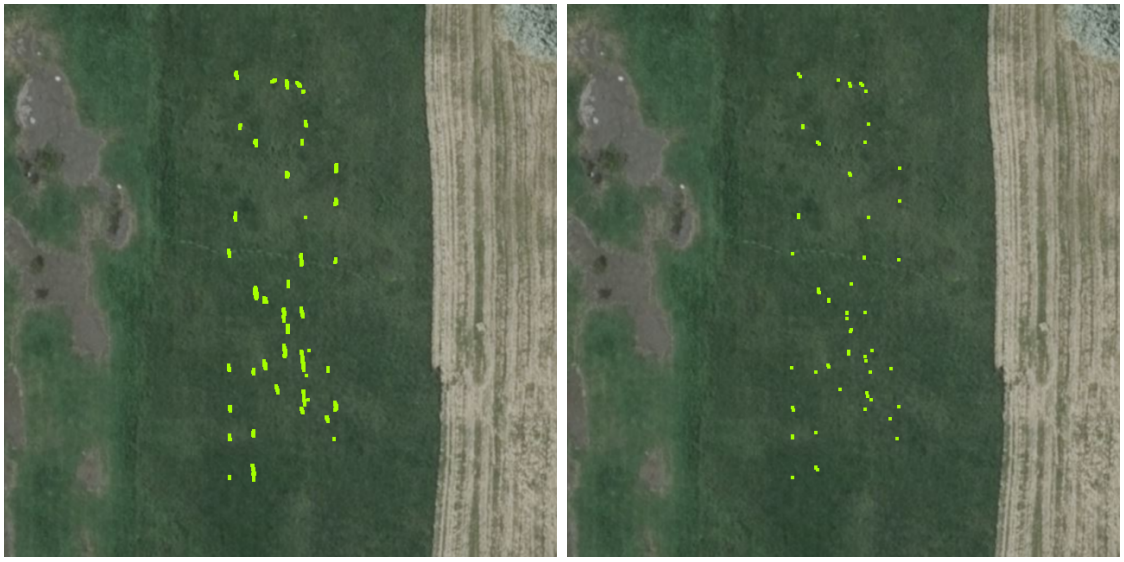

Figure 5.13: Insect detections before and after being clustered when using K-means.

Table 5.5: MOT metrics [43] of the Constrained K-means and MHT results. Dataset D does not include a flight log, so K-means was not a viable method for tracking.

	Const. K-Means				**MHT**			
Dataset	**A**	**B**	**C**	**D**	**A**	**B**	**C**	**D**
MOTA	0.26	0.49	0.46	N/A	0.66	0.56	0.55	0.67
Mostly Tracked	17	30	37	N/A	14	30	33	111
Partially Tracked	5	6	17	N/A	6	5	15	8
Mostly Lost	0	3	43	N/A	2	4	49	6
Number of Fragmentations	22	31	59	N/A	21	23	12	28
ID Switches	58	31	132	N/A	13	8	12	48
ID Precision	0.41	0.47	0.67	N/A	0.78	0.67	0.83	0.66
ID Recall	0.53	0.61	0.49	N/A	0.84	0.86	0.86	0.87
ID F1	0.43	0.52	0.54	N/A	0.81	0.75	0.73	0.73
Ground Truth Insects	22	39	97	125	22	39	97	125

5.7.2.1 Comparing Ground Truth Insects to Real-World Insects using the Hungarian Algorithm

In order to associate ground truth insects with predicted insects, there must be a way to associate them. This problem is commonly referred to as the assignment problem, and an elegant solution is the Hungarian Algorithm [68]. In our implementation, each detection is associated to the nearest ground truth bounding box and if the distance between them is less than 0.5 meters, it is classified as a true positive. If there are additional detections that do not satisfy this criterion, they are classified as false positives. If there are extra ground truth bounding boxes which are not asso-

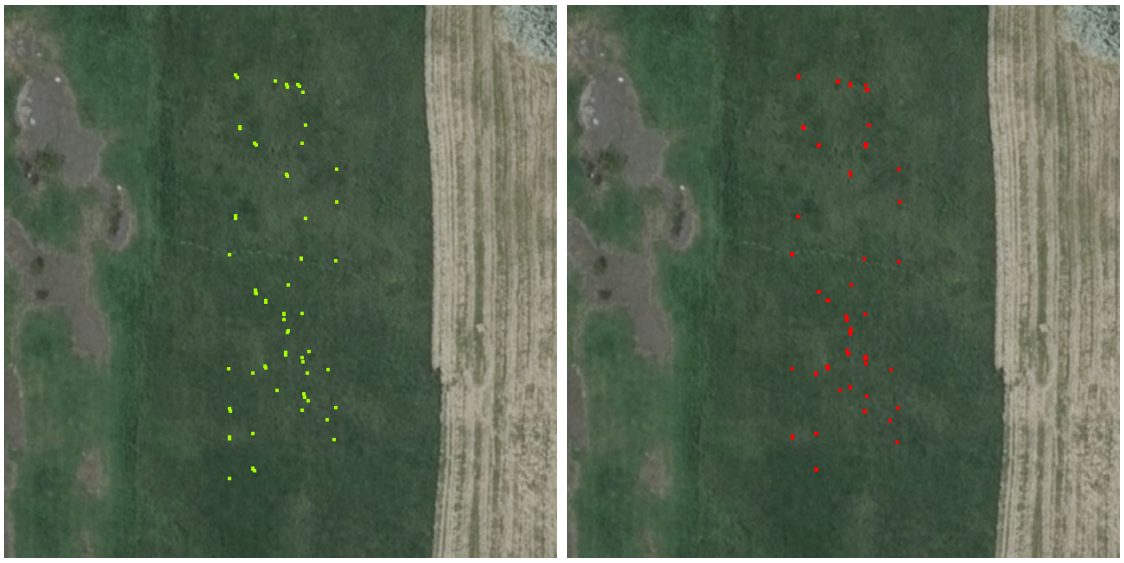

Figure 5.14: Ground truth of insect locations (left) vs. the detected insect locations using K-means (right).

Algorithm 7: Assigning costs to the Hungarian algorithm.

Input : The set x and y of real-world location coordinates of ground-truth insects, GT_x,
The set x and y of real-world location coordinates of grouped detected insects DI_x

Output: Cost matrix for the Hungarian Algorithms, d_w

```
d_w = Empty 2D array of size Count(GT_x) × Count(DI_x);
for x_g, y_g in (GT_x) do
    for x_p, y_p in (DI_x) do
        Compute d_{w_gp} Using Equation 5.19;
```

ciated to any detection, they are classified as false negatives (Figure 5.14). This can be achieved by implementing the following cost metric

$$d_{w_{gp}} = \sqrt{(x_g - x_p)^2 + (y_g - y_p)^2}, \tag{5.19}$$

$$w_{gp} = \begin{cases} d_{w_{gp}} & \text{if } d_{w_{gp}} \leq l_{Hungarian} \\ \infty & \text{otherwise} \end{cases}. \tag{5.20}$$

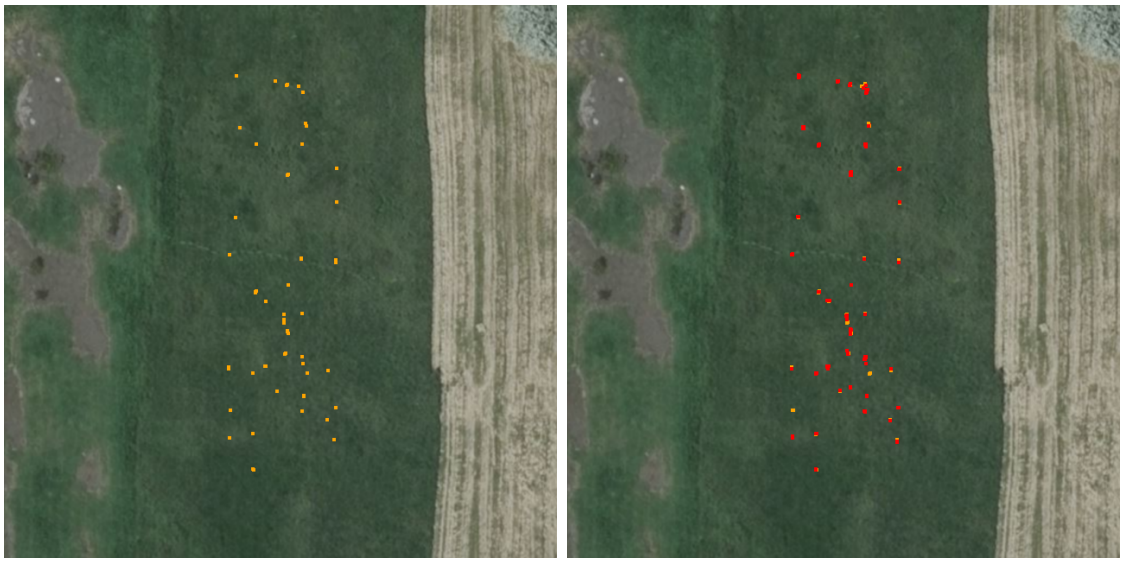

Figure 5.15: Left: Ground truth locations of the insects. Right: MHT insect detections (orange) placed alongside the ground truth locations (red). From dataset C.

Table 5.6: Hungarian algorithm evaluation on constrained K-means and MHT results.

	Const. K-Means			**MHT**		
Dataset	**A**	**B**	**C**	**A**	**B**	**C**
True Positives	17	11	42	19	27	25
False Positives	46	63	55	22	51	52
False Negatives	5	28	25	3	12	15
Precision	0.26	0.14	0.34	0.46	0.34	0.46
Recall	0.77	0.28	0.62	0.86	0.69	0.75
F1	0.40	0.19	0.52	0.60	0.46	0.57

The computation of $d_{w_{gp}}$ is shown in Algorithm 7. The association results can be seen in Table 5.6. The Results for Datasets A, B, and C can be visualized in Figures 5.19, 5.20, and 5.21, respectively. Dataset D is not included because it does not include flight logs.

Figure 5.16: Visualization of results from dataset A.

Figure 5.17: Visualization of results from dataset B.

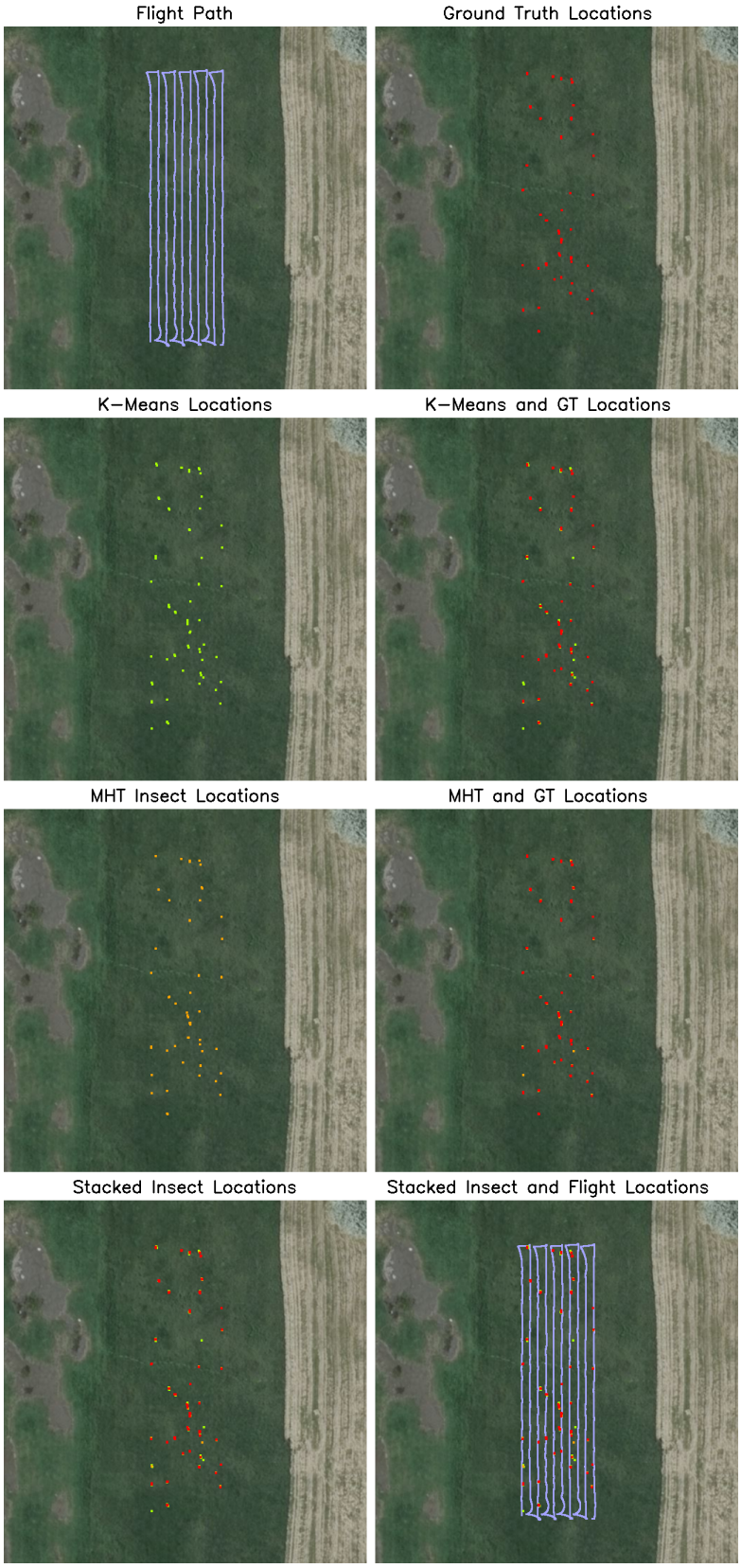

Figure 5.18: Visualization of results from dataset C.

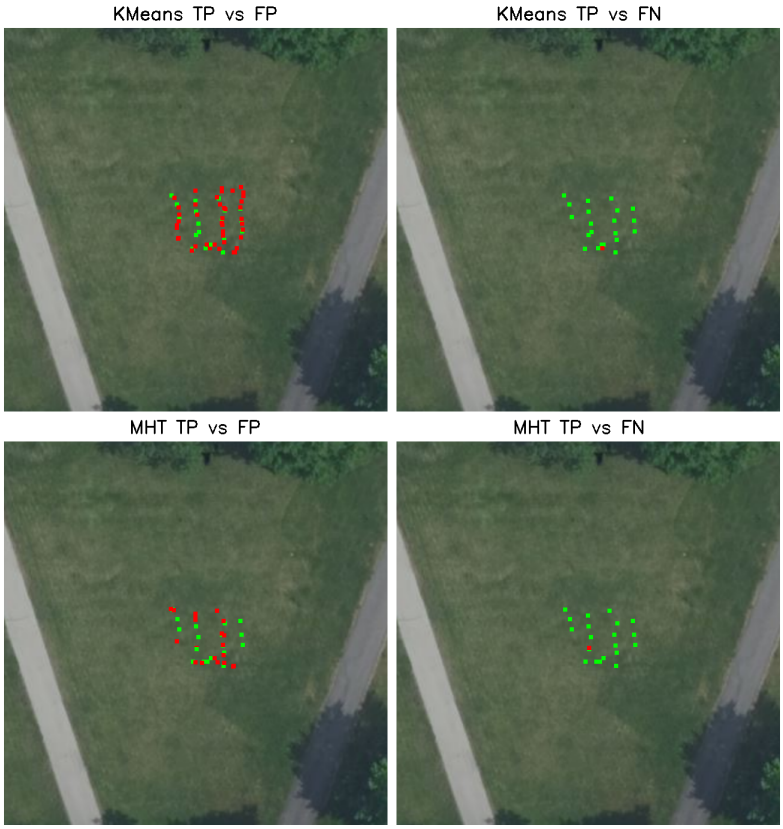

Figure 5.19: Visualization of applying the Hungarian method to the results of dataset A.

5.8 Discussion

The results indicate that the CVAE is a significantly improved way to detect insects in the videos collected by the drone compared to existing methods. Because insects are sparsely distributed throughout the datasets, the CVAE does not learn to represent them, so they simply do not appear in the reconstruction images, while background information does. Previous methods, although arguably less complex than the proposed approach, fail to separate the insects from background noise (especially leaves). Interestingly, the results were worse when trying to find an insect using the entire dataset to train the CVAE (these results are omitted for the sake of conciseness). Interestingly, dataset D had the best results. Considering the number of factors that distinguish dataset D from the other datasets is a direction worth

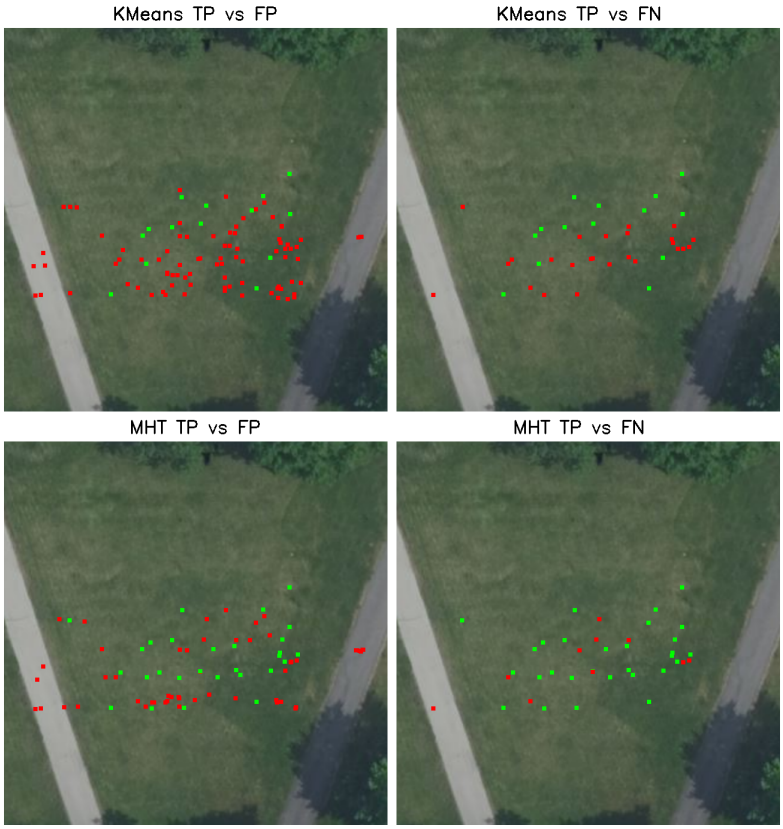

Figure 5.20: Visualization of applying the Hungarian method to the results of dataset B.

exploring in the future to determine the best data collection methods.

5.9 Implementation Details

In this section, we discuss the implementation of the Python code implemented for this project. Table 5.7 lists the variables used in the implementation of the CVAE and their corresponding values. Table 5.8 shows the Anaconda 3[1] environment used by the project. Our proposed method consists of a series of algorithms run as a software pipeline. Each dataset is processed with through these algorithms and the final result produces the latitude and longitude of the detected insect, which are saved to a CSV file that is compared to the hand-labeled ground-truth. All code was

[1]https://www.anaconda.com

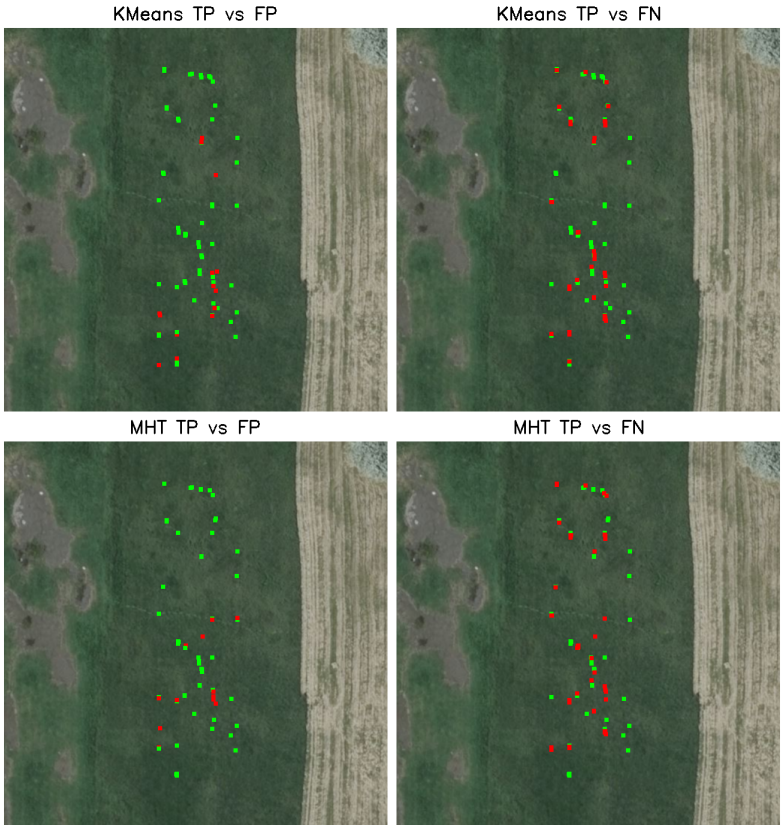

Figure 5.21: Visualization of applying the Hungarian method to the results of dataset C.

written in Python 3.6, with bash scripts acting to automatically connect parts of the code together. The platform used was Ubuntu 18.04. New code was either written by Scott Stewart or sourced from Tensorflow[2], pyMOT[3], or pyMOTMetrics[4].

[2]https://www.tensorflow.org/tutorials/generative/cvae
[3]https://github.com/yoon28/pymht
[4]https://github.com/cheind/py-motmetrics

Table 5.7: Definitions for variables used in the CVAE.

Term	Variable Used	Definition
Subgrid Count	c	17
Channels in subgrid	n_c	3
Pixels in subgrid	n_g	64
size of latent dimension	n_l	650
Number of images in dataset	N	Variable
Pixels in ROI	p_r	720

Table 5.8: Anaconda environment used for all the code discussed in this thesis.

Package Name	**Version**	**Package Name**	**Version**
bleach	3.1.5	pep8	1.7.1
filterpy	1.4.5	pexpect	4.8.0
future	0.18.2	pickleshare	0.7.5
glib	2.65.0	pillow	7.2.0
google-auth	1.30.0	pip	20.3.1
imageio	2.9.0	ply	3.11
ipykernel	5.3.4	py	1.9.0
ipython	5.8.0	py-opencv	3.4.2
jpeg	9d	pygments	2.6.1
json5	0.9.4	pytables	3.4.4
markupsafe	1.1.1	python	3.6.9
matplotlib	3.3.1	python-dateutil	2.8.1
mht	0.0.1	pywavelets	1.1.1
mkl	2019.5	qtpy	1.9.0
motmetrics	1.1.3	requests-oauthlib	1.3.0
nltk	3.4.4	scikit-image	0.17.2
notebook	6.1.3	scikit-learn	0.23.2
numba	0.51.1	scipy	1.5.2
numexpr	2.7.1	tensorboard	2.5.0
numpy	1.19.5	tensorflow-gpu	2.4.1
packaging	20.4	wheel	0.36.2
pandas	1.1.1	yaml	0.2.5
path	15.0.0		

CHAPTER 6
CONCLUSION

The goal of this project is to accurately find insects in aerial videos while requiring as little human input as possible. Anomaly detection offers a promising approach to address this problem. In this thesis, we present an effective strategy for insect detection based on anomalous pixel detection algorithms. Our detection method relies solely on the expected background of videos observed by the drone, so it should be adaptable to other types of environments. We were also able to associate multiple frame-by-frame detections into a set of unique insect identifiers with reasonable accuracy. Finally, we were able to use the location data from the drone in conjunction with a Kalman filter to determine the heading of the drone. This allowed us to calculate the latitude and longitude of the real-world insect locations. Overall, we found a significant improvement in our results over previous approaches to the anomalous insect problem.

There are limitations to the proposed work. While the proposed method is a significant improvement over existing methods, there is still room for improvement when detecting insects. While MHT is better than the baseline constrained k-means strategy, it still struggles to associate insect detections in more complex scenarios involving multiple insects.

6.1 Future Work

The proposed methods represent significant improvements to the current state of the art in insect detection methods. However, there are a number of improvements to be made.

The first potential improvement is to the data collection procedure. Using an RTK-GPS that allows centimeter accurate locations could provide an improved

ground truth for future data collection. In fact, one was purchased and tested for this project, but the COVID-19 pandemic prevented data collection, and the unit died after its lithium ion battery stopped charging due to disuse.

Another potential improvement is in the neural network chosen to reconstruct the image. CVAEs have a number of strengths, but a GAN can reconstruct images with higher accuracy, providing enough training data is available.

The third potential area for improvement is by implementing a physical light filter over the lens of the camera. By filtering only for the wavelengths of light the fluorescent powder reflects the noise could be reduced, potentially improving insect detection performance significantly.

Finally, another potential area for improvement is by testing what factors made dataset D have the best results. For example, having the camera off to the side of the insects, rather than over the top of them. Alternatively, the blue fluorescent powder may be easier for the algorithm to find even if it is harder for a human. Finally, the algorithm may benefit from having several insects visible in any given frame.

BIBLIOGRAPHY

[1] B. Stumph, "Detecting invasive insects using unmanned aerial vehicles," Ph.D. dissertation, Marquette University, 2019.

[2] T. Stankus, "Reviews of science for science librarians: "murder hornets:" vespa mandarinia japonica," *Science & Technology Libraries*, vol. 39, no. 3, pp. 244–252, 2020. [Online]. Available: https://doi.org/10.1080/0194262X.2020.1796890

[3] D. M. Kirkpatrick, K. B. Rice, A. Ibrahim, S. J. Fleischer, J. F. Tooker, A. Tabb, H. Medeiros, I. Morrison, William R, and T. C. Leskey, "The Influence of Marking Methods on Mobility, Survivorship, and Field Recovery of Halyomorpha halys (Hemiptera: Pentatomidae) Adults and Nymphs," *Environmental Entomology*, vol. 49, no. 5, pp. 1026–1031, 08 2020. [Online]. Available: https://doi.org/10.1093/ee/nvaa095

[4] J. Sundell, I. Kojola, and I. Hanski, "A new GPS-GSM-Based method to study behavior of brown bears," *Wildlife Society Bulletin*, vol. 34, no. 2, pp. 446–450, 2006.

[5] D.-H. Lee, C.-G. Park, B. Y. Seo, G. Boiteau, C. Vincent, and T. Leskey, "Detectability of halyomorpha halys (hemiptera: Pentatomidae) by portable harmonic radar in agricultural landscapes," *Florida Entomologist*, vol. 97, 09 2014.

[6] J. An and Cho, "Variational autoencoder based anomaly detection using reconstruction probability," in *2015-2 Special Lecture on IE*, 2015.

[7] J. F. S. Gomes and F. R. Leta, "Applications of computer vision techniques in the agriculture and food industry: a review," *European Food Research and Technology*, vol. 235, no. 6, pp. 989–1000, 2012.

[8] J. Hemming and T. Rath, "PA–precision agriculture: Computer-vision-based weed identification under field conditions using controlled lighting," *Journal of agricultural engineering research*, vol. 78, no. 3, pp. 233–243, 2001.

[9] M. P. Arakeri *et al.*, "Computer vision based fruit grading system for quality evaluation of tomato in agriculture industry," *Procedia Computer Science*, vol. 79, pp. 426–433, 2016.

[10] P. A. Dias, A. Tabb, and H. Medeiros, "Multispecies fruit flower detection using a refined semantic segmentation network," *Ieee robotics and automation letters*, vol. 3, no. 4, pp. 3003–3010, 2018.

[11] L. Costa, Y. Ampatzidis, C. Rohla, N. Maness, B. Cheary, and L. Zhang, "Measuring pecan nut growth utilizing machine vision and deep learning for the better understanding of the fruit growth curve," *Computers and Electronics in Agriculture*, vol. 181, p. 105964, 2021.

[12] P. Roy, A. Kislay, P. A. Plonski, J. Luby, and V. Isler, "Vision-based preharvest yield mapping for apple orchards," *Computers and Electronics in Agriculture*, vol. 164, p. 104897, 2019.

[13] N. Stefas, H. Bayram, and V. Isler, "Vision-based monitoring of orchards with UAVs," *Computers and Electronics in Agriculture*, vol. 163, p. 104814, 2019.

[14] F. H. Iost Filho, W. B. Heldens, Z. Kong, and E. S. de Lange, "Drones: Innovative technology for use in precision pest management," *Journal of economic entomology*, vol. 113, no. 1, pp. 1–25, 2020.

[15] K. B. Rice, S. J. Fleischer, C. M. De Moraes, M. C. Mescher, J. F. Tooker, and M. Gish, "Handheld lasers allow efficient detection of fluorescent marked organisms in the field," *PloS one*, vol. 10, no. 6, p. e0129175, 2015.

[16] D. Rojas-Araya, B. W. Alto, N. Burkett-Cadena, and D. A. Cummings, "Detection of Fluorescent Powders and Their Effect on Survival and Recapture of Aedes aegypti (Diptera: Culicidae)," *Journal of Medical Entomology*, vol. 57, no. 1, pp. 266–272, 10 2019. [Online]. Available: https://doi.org/10.1093/jme/tjz142

[17] D. M. Kirkpatrick, K. B. Rice, A. Ibrahim, W. R. Morrison, and T. C. Leskey, "Influence of harmonic radar tag attachment on nymphal halyomorpha halys mobility, survivorship, and detectability," *Entomologia Experimentalis et Applicata*, vol. 167, no. 12, pp. 1020–1029, 2019.

[18] M. Ebrahimi, M. Khoshtaghaza, S. Minaei, and B. Jamshidi, "Vision-based pest detection based on SVM classification method," *Computers and Electronics in Agriculture*, vol. 137, pp. 52–58, 2017. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S016816991631136X

[19] A. N. Alves, W. S. Souza, and D. L. Borges, "Cotton pests classification in field-based images using deep residual networks," *Computers and Electronics in Agriculture*, vol. 174, p. 105488, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0168169919311342

[20] Y. Kaya, L. Kayci, and R. Tekin, "A computer vision system for the automatic identification of butterfly species via gabor-filter-based texture features and extreme learning machine: Gf+ elm," *TEM J*, vol. 2, no. 1, pp. 13–20, 2013.

[21] S. R. Huddar, S. Gowri, K. Keerthana, S. Vasanthi, and S. R. Rupanagudi, "Novel algorithm for segmentation and automatic identification of pests on plants using image processing," in *2012 Third International Conference on Computing, Communication and Networking Technologies (ICCCNT'12)*. IEEE, 2012, pp. 1–5.

[22] K. Goto, K. Kato, S. Nakatsuka, T. Saito, and H. Aizawa, "Anomaly detection of solder joint on print circuit board by using adversarial autoencoder," in *Fourteenth International Conference on Quality Control by Artificial Vision*, vol. 11172. International Society for Optics and Photonics, 2019, p. 111720T.

[23] A. S. Krasichkov, E. B. Grigoriev, M. I. Bogachev, and E. M. Nifontov, "Shape anomaly detection under strong measurement noise: An analytical approach to adaptive thresholding," *Physical Review E*, vol. 92, no. 4, p. 042927, 2015.

[24] O. Rippel, P. Mertens, and D. Merhof, "Modeling the distribution of normal data in pre-trained deep features for anomaly detection," in *2020 25th International Conference on Pattern Recognition (ICPR)*, 2021, pp. 6726–6733.

[25] P. C. Ngo, A. A. Winarto, C. K. L. Kou, S. Park, F. Akram, and H. K. Lee, "Fence GAN: Towards better anomaly detection," in *2019 IEEE 31st International Conference on Tools with Artificial Intelligence (ICTAI)*. IEEE, 2019, pp. 141–148.

[26] J. Kim, K. Jeong, H. Choi, and K. Seo, "GAN-based anomaly detection in imbalance problems," in *European Conference on Computer Vision*. Springer, 2020, pp. 128–145.

[27] M. A. Pimentel, D. A. Clifton, L. Clifton, and L. Tarassenko, "A review of novelty detection," *Signal Processing*, vol. 99, pp. 215–249, 2014.

[28] P. Burlina, N. Joshi, I. Wang *et al.*, "Where's wally now? deep generative and discriminative embeddings for novelty detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 11 507–11 516.

[29] R. Chalapathy, A. K. Menon, and S. Chawla, "Anomaly detection using one-class neural networks," *arXiv preprint arXiv:1802.06360*, 2018.

[30] L. Ruff, R. Vandermeulen, N. Goernitz, L. Deecke, S. A. Siddiqui, A. Binder, E. Müller, and M. Kloft, "Deep one-class classification," in *International conference on machine learning*. PMLR, 2018, pp. 4393–4402.

[31] P. Bergmann, M. Fauser, D. Sattlegger, and C. Steger, "Uninformed students: Student-teacher anomaly detection with discriminative latent embeddings," in

IEEE/CVF Conference on Computer Vision and Pattern Recognition, 2020, pp. 4183–4192.

[32] P. Perera, R. Nallapati, and B. Xiang, "OCGAN: One-class novelty detection using gans with constrained latent representations," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 2898–2906.

[33] D. Abati, A. Porrello, S. Calderara, and R. Cucchiara, "Latent space autoregression for novelty detection," in *IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2019, pp. 481–490.

[34] P. Bergmann, S. Löwe, M. Fauser, D. Sattlegger, and C. Steger, "Improving unsupervised defect segmentation by applying structural similarity to autoencoders," *arXiv preprint arXiv:1807.02011*, 2018.

[35] L. Torrey and J. Shavlik, "Transfer learning," in *Handbook of research on machine learning applications and trends: algorithms, methods, and techniques*. IGI global, 2010, pp. 242–264.

[36] P. Napoletano, F. Piccoli, and R. Schettini, "Anomaly detection in nanofibrous materials by CNN-based self-similarity," *Sensors*, vol. 18, no. 1, p. 209, 2018.

[37] K. He, X. Zhang, S. Ren, and J. Sun, "Deep residual learning for image recognition," in *IEEE conference on computer vision and pattern recognition*, 2016, pp. 770–778.

[38] Y. Fan, G. Wen, D. Li, S. Qiu, M. D. Levine, and F. Xiao, "Video anomaly detection and localization via gaussian mixture fully convolutional variational autoencoder," *Computer Vision and Image Understanding*, vol. 195, p. 102920, 2020. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S1077314218302674

[39] D. Koller, J. Weber, and J. Malik, "Robust multiple car tracking with occlusion reasoning," in *Computer Vision — ECCV '94*, J.-O. Eklundh, Ed. Berlin, Heidelberg: Springer Berlin Heidelberg, 1994, pp. 189–196.

[40] S. Pellegrini, A. Ess, K. Schindler, and L. Van Gool, "You'll never walk alone: Modeling social behavior for multi-target tracking," in *2009 IEEE 12th International Conference on Computer Vision*. IEEE, 2009, pp. 261–268.

[41] W. Luo, J. Xing, A. Milan, X. Zhang, W. Liu, and T.-K. Kim, "Multiple object tracking: A literature review," *Artificial Intelligence*, vol. 293, p. 103448, 2021. [Online]. Available: https://www.sciencedirect.com/science/article/pii/S0004370220301958

[42] S. Blackman, "Multiple hypothesis tracking for multiple target tracking," *IEEE Aerospace and Electronic Systems Magazine*, vol. 19, no. 1, pp. 5–18, 2004.

[43] K. Bernardin and R. Stiefelhagen, "Evaluating multiple object tracking performance: the clear mot metrics," *EURASIP Journal on Image and Video Processing*, vol. 2008, pp. 1–10, 2008.

[44] C. Heindl, "Python mot metrics," 2020. [Online]. Available: https://github.com/cheind/py-motmetrics

[45] J. Kim, S. Kim, C. Ju, and H. I. Son, "Unmanned aerial vehicles in agriculture: A review of perspective of platform, control, and applications," *IEEE Access*, vol. 7, pp. 105 100–105 115, 2019.

[46] R. Casado and A. Bermúdez, "A simulation framework for developing autonomous drone navigation systems," *Electronics*, vol. 10, no. 1, p. 7, 2021.

[47] J. P. Rojas, Devia, Petro, Martinez, Mondragon, Patino, and Colorado, "aerial mapping of rice crops using mosaicing techniques for vegetative index monitoring"," in *2018 International Conference on Unmanned Aircraft Systems (ICUAS)*, June 2018, pp. 846–855.

[48] B. S. Faiçal, H. Freitas, P. H. Gomes, L. Y. Mano, G. Pessin, A. C. de Carvalho, B. Krishnamachari, and J. Ueyama, "An adaptive approach for uav-based pesticide spraying in dynamic environments," *Computers and Electronics in Agriculture*, vol. 138, pp. 210–223, 2017.

[49] P. Lottes, R. Khanna, J. Pfeifer, R. Siegwart, and C. Stachniss, "UAV-based crop and weed classification for smart farming," in *2017 IEEE International Conference on Robotics and Automation (ICRA)*. IEEE, 2017, pp. 3024–3031.

[50] G. Ronchetti, A. Mayer, A. Facchi, B. Ortuani, and G. Sona, "Crop row detection through uav surveys to optimize on-farm irrigation management," *Remote Sensing*, vol. 12, no. 12, p. 1967, 2020.

[51] C. A. Lin, K. Shah, L. C. C. Mauntel, and S. A. Shah, "Drone delivery of medications: Review of the landscape and legal considerations," *The Bulletin of the American Society of Hospital Pharmacists*, vol. 75, no. 3, pp. 153–158, 2018.

[52] J. S. Turner, K. M. Ross, and J. T. Lin, "Sky-high promise (and potential pitfalls) of drones," *Natural Gas & Electricity*, vol. 33, no. 12, pp. 1–6, 2017.

[53] S. Minaee, Y. Y. Boykov, F. Porikli, A. J. Plaza, N. Kehtarnavaz, and D. Terzopoulos, "Image segmentation using deep learning: A survey," *IEEE Transac-*

tions on Pattern Analysis and Machine Intelligence, pp. 1–1, 2021.

[54] N. Otsu, "A threshold selection method from gray-level histograms," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. 9, no. 1, pp. 62–66, Jan 1979.

[55] T. Inoue, S. Choudhury, G. De Magistris, and S. Dasgupta, "Transfer learning from synthetic to real images using variational autoencoders for precise position detection," in *2018 25th IEEE International Conference on Image Processing (ICIP)*. IEEE, 2018, pp. 2725–2729.

[56] T. Song, J. Sun, B. Chen, W. Peng, and J. Song, "Latent space expanded variational autoencoder for sentence generation," *IEEE Access*, vol. 7, pp. 144 618–144 627, 2019.

[57] A. Milan, L. Leal-Taixé, I. Reid, S. Roth, and K. Schindler, "Mot16: A benchmark for multi-object tracking," *arXiv preprint arXiv:1603.00831*, 2016.

[58] P. Wu and W. R. Mebane Jr, "Marmot: A deep learning framework for constructing multimodal representations for vision-and-language tasks," *Manuscript in preparation]. University of Michigan. https://www. patrickywu. com/working papers/marmot wu. pdf*, 2020.

[59] D. Stadler and J. Beyerer, "Improving multiple pedestrian tracking by track management and occlusion handling," in *Proceedings of the IEEE/CVF Conference on Computer Vision and Pattern Recognition*, 2021, pp. 10 958–10 967.

[60] P. Dai, R. Weng, W. Choi, C. Zhang, Z. He, and W. Ding, "Learning a proposal classifier for multiple target tracking (supplementary material)."

[61] A. Siddique, R. J. Mozhdehi, and H. Medeiros, "Deep heterogeneous autoencoder for subspace clustering of sequential data," *arXiv preprint arXiv:2007.07175*, 2020.

[62] K. Wagstaff, C. Cardie, S. Rogers, and S. SchrÃ¶dl, "Constrained K-means clustering with background knowledge." in *International Conference on Machine Learning*, 2001, pp. 577–584.

[63] S. J. Julier and Uhlmann, "New extension of the kalman filter to nonlinear systems," in *AeroSense '97*, vol. 3068, 1997. [Online]. Available: https://doi.org/10.1117/12.280797

[64] tensorflow, "Convolutional variational autoencoder," 2021.

[65] Q. Li, R. Li, K. Ji, and W. Dai, “Kalman filter and its application,” in *2015 8th International Conference on Intelligent Networks and Intelligent Systems (ICINIS)*, 2015, pp. 74–77.

[66] B. Stumph, M. H. Virto, H. Medeiros, A. Tabb, S. Wolford, K. Rice, and T. Leskey, “Detecting invasive insects with unmanned aerial vehicles,” in *2019 International Conference on Robotics and Automation (ICRA)*. IEEE, 2019, pp. 648–654.

[67] H. Ali and Kadhum, “"k-means clustering algorithm applications in data mining and pattern recognition",” in *International Journal of Science and Research (IJSR)*, 2017.

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