# USING MACHINE LEARNING TO IMPROVE FISH, INSECT, AND UAV DETECTION IN PULSED LIDAR DATA

by

Trevor Charles Vannoy

A dissertation submitted in partial fulfillment
of the requirements for the degree

of

Doctor of Philosophy

in

Electrical Engineering

MONTANA STATE UNIVERSITY
Bozeman, Montana

May 2025

©COPYRIGHT

by

Trevor Charles Vannoy

2025

Creative Commons Attribution 4.0 International License

# ACKNOWLEDGEMENTS

Dr. Whitaker, thank you for letting me spend so much time teaching classes and doing outreach events; Dr. Snider, thank you for letting me (or making me) teach your classes; without you two allowing me to teach and do outreach, I would have graduated sooner but had much less fun. Dr. Shaw, thank you for trusting me around your lasers—you probably should not have let a computer person near a lidar. Dr. Maher, thank you for helping me frame my research contributions. I would like to thank all my fellow lab members for their help and their willing (and unwilling) participation in random philosophical discussions; without your camaraderie and distraction, I would have graduated sooner but had much less fun. Last but not least, I would like to thank all the students who helped me complete my research: Wyatt Weller, John Fike, Nat Sweeney, Kyle Rust, Jackson Belford, and Joseph Aist.

This material (AFRL-2025-1396) is based on research sponsored by Air Force Research Lab through a subcontract with S2 Corporation (via FA8650-16-C-1954) and under agreement number FA8650-21-1-1176. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation therein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Lab or the U.S. Government.

TABLE OF CONTENTS

1. INTRODUCTION ..... 1

Pulsed Lidar ..... 2
Lidar Systems ..... 3
Machine Learning ..... 4
Binary Classification ..... 5
Imbalanced Learning ..... 7
Undersampling ..... 8
Oversampling ..... 9
Boosting ..... 10
Non-stationary Data ..... 10
Summary of Projects and Contributions ..... 11

2. MACHINE LEARNING-BASED REGION OF INTEREST DETECTION IN AIRBORNE LIDAR FISHERIES SURVEYS ..... 14

Contribution of Authors and Co-Authors ..... 14
Manuscript Information ..... 15
Abstract ..... 16
Introduction ..... 16
Airborne lidar for fishery surveys ..... 17
Supervised learning on imbalanced data ..... 18
Lidar systems ..... 20
NOAA lidar ..... 20
MSU lidar ..... 21
Datasets ..... 21
Yellowstone Lake Dataset ..... 22
Gulf of Mexico Dataset ..... 23
Methods ..... 26
Classification algorithms ..... 26
Random Undersampling ..... 27
Boosting ..... 27
Evaluation metrics ..... 28
Preprocessing ..... 30
Surface detection and smoothing ..... 30
Surface correction ..... 31
Depth adjustment ..... 32
Training procedure ..... 33
Parameter tuning ..... 33
Results ..... 35

TABLE OF CONTENTS – CONTINUED

First day training set ..... 35
Yellowstone Lake ..... 35
Gulf of Mexico ..... 38
Randomized training set ..... 41
Yellowstone Lake ..... 41
Gulf of Mexico ..... 44
Discussion ..... 46
Conclusion ..... 49
Future work ..... 50
Acknowledgment ..... 50
Code, Data, and Materials Availability ..... 51

3. COMPARISON OF SUPERVISED LEARNING AND CHANGEPOINT DETECTION FOR INSECT DETECTION IN LIDAR DATA ..... 52

Contribution of Authors and Co-Authors ..... 52
Manuscript Information ..... 53
Abstract ..... 54
Introduction ..... 54
Materials and Methods ..... 57
Lidar System ..... 57
Dataset ..... 61
Preprocessing ..... 66
Supervised Methods ..... 67
Data Sampling ..... 68
Feature Engineering Methods ..... 69
Deep Learning Methods ..... 72
Changepoint Detection ..... 73
Evaluation Metrics ..... 75
Results ..... 76
Supervised Methods ..... 77
Feature Analysis ..... 77
Data Sampling ..... 80
Validation ..... 81
Testing ..... 82
Changepoint Detection ..... 84
Discussion ..... 86
Sources of Uncertainty and Error ..... 86
Effects of Confidence Ratings ..... 87
Model Generalization ..... 88

TABLE OF CONTENTS – CONTINUED

Algorithm Recommendations ..... 90
Future Research Directions ..... 91
Conclusions ..... 92
Acknowledgements ..... 93

4. WINGBEAT-MODULATION LIDAR HARDWARE AND SOFTWARE UPGRADES ..... 94

Removing Transmission-line Reflections ..... 94
Digital PMT Gain Controller ..... 97
System Architecture ..... 98
Hardware Design ..... 98
Software Design ..... 101
Architecture ..... 103
Communication Protocol ..... 103
Usage ..... 103
Connecting to the Gain Controller ..... 104
Setting Gain Values ..... 105
Digitizer Upgrade ..... 105
Digitizer Control Software ..... 106
Typical Usage ..... 107
Pan-tilt Controller Software ..... 111
Architecture and Implementation ..... 111

5. UAV DETECTION AND PROPELLER CHARACTERIZATION ..... 115

Introduction ..... 115
Experiment Prerequisites ..... 116
Beam Diverger ..... 116
Pan-tilt Controller ..... 122
Drone Control Software ..... 124
Data Collection ..... 126
Data Collection Procedure ..... 126
Data Collection Software ..... 128
Dataset Description ..... 130
Data Analysis Methodology ..... 132
Machine Learning ..... 132
Visual Inspection ..... 133
Results and Discussion ..... 134
Visible Inspection ..... 134

TABLE OF CONTENTS – CONTINUED

Supervised Learning ..... 135
Data Sampling ..... 135
Validation ..... 136
Testing ..... 138
Feature Importance ..... 138
Comparison to Visual Inspection Results ..... 141
Aliasing ..... 141
Signal-to-Noise Ratio ..... 142
Visibility of Multiple Propeller Frequencies ..... 143
Conclusion ..... 143
Acknowledgments ..... 144

6. CONCLUSION ..... 146

CUMULATIVE REFERENCES CITED ..... 148

# LIST OF TABLES

Table Page

1. Table 1 Summary of Yellowstone Lake data ..... 23
2. Table 2 Summary of Gulf of Mexico data ..... 24
3. Table 3 Hyperparameter search ranges ..... 35
4. Table 4 Hyperparameter values for the "first day" experiment on Yellowstone Lake ..... 36
5. Table 5 ROI performance metrics for the Yellowstone Lake "first day" experiment ..... 37
6. Table 6 Hyperparameter values for the "first day" experiment on the Gulf of Mexico dataset ..... 39
7. Table 7 ROI performance metrics for the Gulf of Mexico "first day" experiment ..... 39
8. Table 8 Hyperparameter values for the randomized training set experiment on the Yellowstone Lake dataset ..... 42
9. Table 9 ROI performance metrics for the Yellowstone Lake randomized experiment ..... 42
10. Table 10 Hyperparameter values for the randomized training set experiment on the Gulf of Mexico dataset ..... 44
11. Table 11 ROI performance metrics for the Gulf of Mexico randomized experiment ..... 45
12. Table 12 Training runtimes ..... 48
13. Table 13 Bee dataset overview ..... 66
14. Table 14 Best undersampling and augmentation ratios ..... 81
15. Table 15 Validation results ..... 82
16. Table 16 Testing results ..... 84
17. Table 17 Changepoint detection results ..... 86
18. Table 18 Best undersampling and augmentation ratios ..... 136
19. Table 19 Validation results ..... 137

LIST OF TABLES – CONTINUED

Table Page

20. Table 20 Testing results....138

# LIST OF FIGURES


| Figure | | | Page |
|---|---|---|---|
| 1. | Figure 1 | Pulsed lidar data representation | 3 |
| 2. | Figure 2 | Binary confusion matrix | 6 |
| 3. | Figure 3 | Example lidar data from Yellowstone Lake | 22 |
| 4. | Figure 4 | Example lidar data from Gulf of Mexico | 25 |
| 5. | Figure 5 | Neural network architecture | 26 |
| 6. | Figure 6 | Confusion matrix terminology | 28 |
| 7. | Figure 7 | Data preprocessing overview | 30 |
| 8. | Figure 8 | Example lidar data from Yellowstone Lake after processing | 32 |
| 9. | Figure 9 | Yellowstone Lake “first day” ROI cross validation results | 37 |
| 10. | Figure 10 | Yellowstone lake “first day” ROI holdout results | 38 |
| 11. | Figure 11 | Gulf of Mexico “first day” ROI cross validation results | 40 |
| 12. | Figure 12 | Gulf of Mexico “first day” ROI holdout results | 40 |
| 13. | Figure 13 | Yellowstone Lake ROI cross validation results | 43 |
| 14. | Figure 14 | Yellowstone Lake ROI holdout results | 43 |
| 15. | Figure 15 | Gulf of Mexico ROI cross validation results | 45 |
| 16. | Figure 16 | Gulf of Mexico ROI holdout results | 46 |
| 17. | Figure 17 | Example ROI with labels | 49 |
| 18. | Figure 18 | Insect detection intuition | 58 |
| 19. | Figure 19 | Optical schematic of the lidar system | 59 |
| 20. | Figure 20 | Picture of the lidar system during data collection. | 60 |
| 21. | Figure 21 | Pulsed lidar data representation | 61 |
| 22. | Figure 22 | A picture of bee activity around the beehives | 62 |
| 23. | Figure 23 | Example bee images | 64 |
| 24. | Figure 24 | Bee transit times and wingbeat frequencies | 65 |

LIST OF FIGURES – CONTINUED

Figure Page

25. Figure 25 Bee image before and after thresholding ..... 67
26. Figure 26 Features ranked by mutual information ..... 71
27. Figure 27 gfpop graph for detecting bees ..... 74
28. Figure 28 Pairwise scatter plot of the three most important features ..... 78
29. Figure 29 Class-wise distributions of harmonic frequencies ..... 80
30. Figure 30 Recall as a function of confidence rating ..... 88
31. Figure 31 Misclassification examples ..... 89
32. Figure 32 Examples of reflections in lidar data ..... 94
33. Figure 33 PMT response as a function of gain ..... 96
34. Figure 34 PMT response for different cables ..... 96
35. Figure 35 The original gain controller circuit ..... 97
36. Figure 36 Digital gain controller system architecture ..... 98
37. Figure 37 Digital gain controller hardware architecture ..... 99
38. Figure 38 Picture of the digital gain controller ..... 101
39. Figure 39 Digital gain controller software flowchart ..... 102
40. Figure 40 Digital gain controller software architecture ..... 104
41. Figure 41 Class diagram for the `quickset_pan_tilt` package. ..... 112
42. Figure 42 Geometry used to derive the telescope's FOV diameter as a function of distance. ..... 117
43. Figure 43 Plot of the telescope's FOV diameter and the UAV dimensions. ..... 118
44. Figure 44 Geometry used to derive the distance where the telescope FOV and beam diameters intersect. ..... 119
45. Figure 45 Optical schematic of the beam diverger system. ..... 120
46. Figure 46 Beam width as a function of distance for different converging lens focal lengths. ..... 121

LIST OF FIGURES – CONTINUED

Figure Page

47. Figure 47 A Happymodel Crux3 UAV mounted to the pan-tilt controller. .....123
48. Figure 48 A Stan FPV Feather UAV mounted to the pan-tilt controller.....123
49. Figure 49 Diagram of the experiment setup. .....126
50. Figure 50 The five different beam configurations. .....127
51. Figure 51 Data collection flowchart .....129
52. Figure 52 Lidar image with a visible UAV propeller .....131
53. Figure 53 Time-domain and frequency-domain plots of a UAV propeller signal .....132
54. Figure 54 Examples of spectra and their corresponding label. .....133
55. Figure 55 Propeller frequency detectability as a function of angle.....134
56. Figure 56 Propeller frequency detectability based on beam configuration.....135
57. Figure 57 Features ranked by mutual information .....140
58. Figure 58 Example of UAV propeller signal aliasing .....142
59. Figure 59 Example spectrum showing multiple propeller frequencies.....143

# LIST OF LISTINGS

Listing Page

1. Listing 1 An example of collecting a single record of lidar data.....108
2. Listing 2 An example of collecting a multiple records of lidar data.....109
3. Listing 3 Example digitizer configuration file.....110
4. Listing 4 An excerpt of the protocol method dispatch dictionary. .....113
5. Listing 5 Implementation of assembling a command packet using method dispatch.....114
6. Listing 6 Implementation of assembling a command packet using a case statement.....114
7. Listing 7 Typical usage of the `quickset_pan_tilt` package. .....114
8. Listing 8 A minimal example of setting motor throttle values using the YAMSPy library. .....124
9. Listing 9 A minimal example of getting motor RPM values using the YAMSPy library. .....125
10. Listing 10 A minimal example of getting motor RPM values using the YAMSPy library. .....125

# ABSTRACT

The work in this dissertation focuses on detecting fish, insects, and unoccupied aerial vehicles (UAVs) in imbalanced lidar data. The highly imbalanced data produced by pulsed lidar systems makes detecting the objects of interest challenging and time-consuming; moreover, the detection process has traditionally been performed manually. By combining techniques from signal processing, machine learning, and imbalanced classification, I created automated data analysis pipelines that addressed the difficulties of manually analyzing imbalanced lidar data. Using my data analysis techniques, I was able to make contributions in all three application areas. In the fish detection application, I was able to automatically detect 86% of regions that contained fish. In the insect detection application, reliably identifying insects is one of the primary challenges in using lidar for long-term non-invasive insect monitoring. My methods identified 95% of the insect in the dataset, helping further lidar's readiness for insect monitoring. Lastly, my machine learning methods were able to detect 97% of the UAVs in the dataset, while we were only able to visually detect 43% of the UAVs. My contributions in each field will enable researchers that are not signal processing and machine learning experts to utilize automated techniques in their work, allowing researchers to spend more time applying their domain expertise to the problems they are actually interested in solving. By making contributions in three unique fields, I show that my data analysis pipeline is generally applicable for many detection applications.

# 1

# INTRODUCTION

Lidar is commonly used for a wide variety of remote sensing applications: atmospheric sensing, autonomous driving, geographic surveying, and object detection—just to name a few. The work in this dissertation focuses on three applications: fish detection, insect detection, and unoccupied aerial vehicle (UAV) detection and characterization. These applications all used pulsed lidar systems that produced large amounts of measurements. The vast majority of these measurements did not contain the object of interest. This highly imbalanced data makes object detection challenging and time-consuming. The purpose of my work is to develop automated object-detection techniques which reduce the need for time-consuming manual data analysis and allow researchers to spend more time doing their domain science.

To expand the capabilities of automated object-detection techniques in pulsed lidar data, I combined techniques from signal processing, machine learning, and imbalanced classification. I use a combination of signal processing and application-domain knowledge to extract features that can help uniquely identify the object. I then feed the extracted features into supervised machine learning techniques for the final object classification. Given labeled input-output pairs, supervised machine learning excels at finding patterns in the data and correctly distinguishing between classes. By using a combination of traditional signal processing, domain knowledge, and machine learning, my methods were able to automatically detect fish, insects, and UAVs.

The following sections provide a brief background on pulsed lidar and machine learning, followed by a summary of my projects and contributions.

# Pulsed Lidar

The data used in this dissertation were collected with pulsed lidar systems. Pulsed lidar systems measure object distance by measuring the round-trip delay of a short pulse of light [1]. When a pulse reflects off an obstacle, light is reflected back to the receiver, where the received intensity is measured. By sampling the received light, we can measure the received intensity as a function of distance: each sample in time corresponds to a distance determined by the time the light took to travel to the obstacle and back, i.e. distance $= c \times (t_n - t_0)/2$ where $c$ is the speed of light, $t_n$ is the time of the $n^{th}$ sample, and $t_0$ is when the pulse was sent.

Depending on the system's sampling rate and the lidar's range, each pulse can have upwards of 1000 samples, or distance bins. Multiple pulses are sent out at the pulse repetition frequency (PRF); for a stationary lidar, each pulse measures the same path, but at different points in time; for an airborne lidar, each pulse measures a different path in space because the lidar is moving. Pulsed lidar data is usually represented as a 2D image, as shown in Fig. 1, where the axes represent the pulse number and distance bin, and the magnitude of each pixel is the received intensity.

While the lidar is usually visualized as a 2D time-range image, the majority of the analysis methods presented in Chapters 3 and 5 operate on the rows of the images instead of the entire images; that is, for each range bin, we analyze the time-series formed by consecutive laser pulses. Analyzing the time-series for each range bin allows us to compute a Fourier transform and analyze the spectrum at that range bin; in the case of insect detection, the spectrum tells us the insect's wingbeat frequency; in the case of drone detection, the spectrum shows the signature of the rotating propellers and vibrating drone. In contrast, the fish-detection methods in Chapter 2 operate on individual lidar pulses, which are the columns in the images; knowing which lidar pulse a fish was detected in gives us the geographical

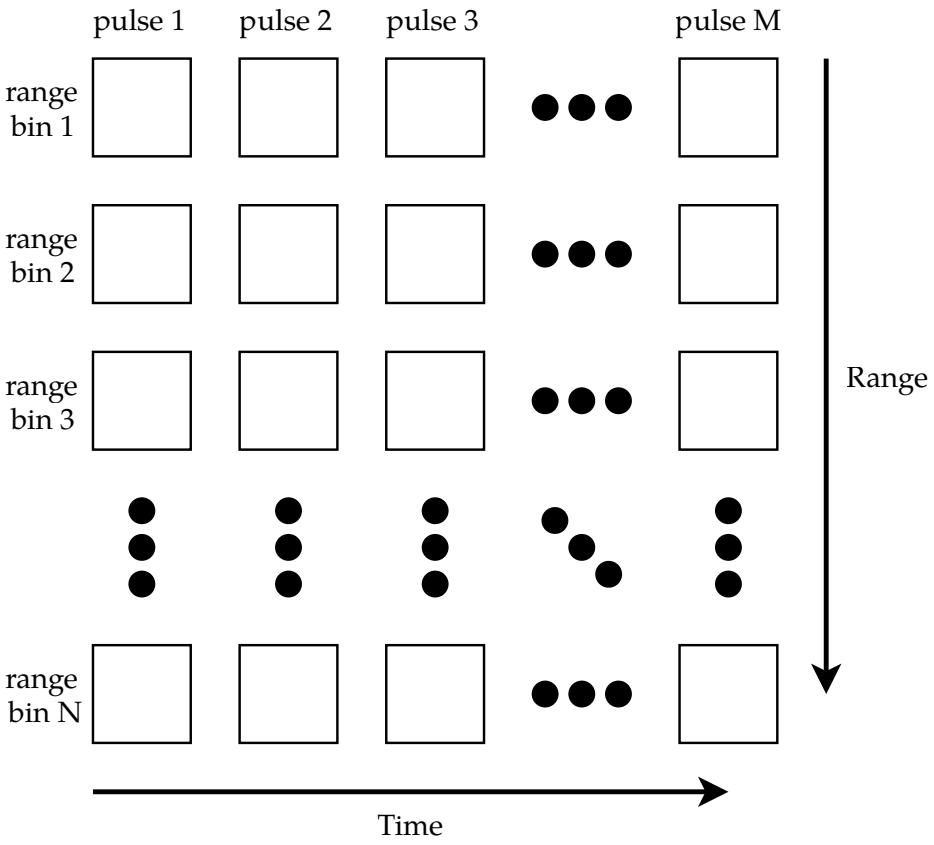



Figure 1: A generic example of how the data were stored. Each column in the matrix represents one pulse of the lidar, and each row represents one range bin. Each matrix element is one sample.

location of the fish, which is what we are most interested in.

Lidar Systems

The work in this dissertation used three different pulsed lidar systems. In Chapter 2, data from the National Oceanic and Atmospheric Administration (NOAA) fish lidar and the Montana State University (MSU) fish lidar were used; these lidar systems are flown in small aircraft and have relatively low PRFs of 30 Hz and 100 Hz, respectively.

The work in Chapters 3, 4, 5 use the wingbeat-modulation lidar developed by MSU's Optical Remote Sensor Laboratory [2]; this lidar system has a higher PRF of approximately 4 kHz, which enables the Fourier transform data analysis methods briefly described above.

All of these instruments are described in more detail in the following chapters.

## Machine Learning

Broadly speaking, machine learning is the study and implementation of algorithms that use data to "learn" underlying patterns and processes; machine learning is primarily data-driven, as opposed to model-based. Machine learning can be roughly split into three main fields:

1. supervised learning
2. unsupervised learning
3. reinforcement learning.

Supervised learning learns how to map input data to output labels by training on example input-output pairs. This is done by "training" the algorithm on a *training* dataset. Since supervised algorithms have access to the ground truth labels for each input, they can be trained to minimize the misclassification rate. Supervised learning is by far the most common type of machine learning. Common supervised learning algorithms include support vector machines (SVM) [3], Naïve Bayes [4], linear and quadratic discriminant analysis [5], logistic regression [4], decision trees [6], neural networks [7], and k-nearest neighbors [8]. The primary drawback of supervised learning is the need for ground truth labels. Ground truth labels are not always available or easy to create. For the applications presented in this thesis, creating ground truth labels took a significant amount of time.

Unsupervised learning, on the other hand, does not have access to ground truth labels. Unsupervised learning algorithms are designed to find patterns and structure in the input data without prior knowledge of the patterns. Clustering algorithms are the most common type of unsupervised learning algorithms. In many real-world applications, unsupervised

learning is useful because we don't have access to the ground truth labels. Common clustering algorithms include k-means [9], Gaussian mixture models [10], DBSCAN [11], hierarchical clustering [12], and spectral clustering [13].

Reinforcement learning is akin to playing a game: an actor makes a decision, and is rewarded accordingly. In reinforcement learning, we do not have access to the correct answer at every step of the game, but we have a way to assess how good each action is. Eventually, after many trials, the algorithm will learn what to do based upon its previous experience—similar to how a toddler learns to not touch a hot stove.

Machine learning algorithms can also be categorized by the type of output they predict. Regression algorithms predict continuous quantities, whereas classification algorithms predict categorical quantities. The work presented here focuses uses classification because object detection is a binary classification problem. In particular, the problems in this work fall under *imbalanced binary classification*; the datasets have relatively few examples of the objects, thus making the classes highly imbalanced. The following sections give an overview of binary classification and imbalanced learning.

Binary Classification

Binary classification is a special case of classification where there are only two classes. In this dissertation, the two classes are always "object" or "no object". Figure 2 shows a binary confusion matrix, from which all classification metrics can be computed.

Accuracy is the most common classification metric, which is defined as

$$\frac{TP + TN}{TP + TN + FP + FN}. \tag{1}$$

Conceptually, accuracy is the percentage of correct predictions. While accuracy is simple and easy to interpret, it is not informative for imbalanced classification problems where one class has very few instances in the dataset. For example, if there are 99 instances in the majority

| | | **Predicted class** | |
|---|---|---|---|
| | | **p** | **n** |
| **Actual class** | $\mathbf{p'}$ | True Positive ($TP$) | False Negative ($FN$) |
| | $\mathbf{n'}$ | False Positive ($FP$) | True Negative ($TN$) |

Figure 2: Generic binary confusion matrix. For our purposes, class $\mathbf{p'}$ are the objects (fish, insects, or drones), and class $\mathbf{n'}$ are non-object instances; class $\mathbf{p}$ are predicted objects, and $\mathbf{n}$ are predicted non-object instances.

class, and 1 instance in the minority class, a classifier can achieve 99% accuracy by simply labeling everything as the majority class; when we are interested in detecting the minority class, as is the case in this dissertation, a 99% accuracy metric is misleading. Consequently, we use other classification metrics such as recall, precision, $F_\beta$, and the Matthews correlation coefficient (MCC) [14].

Recall is a measure of how many true objects were correctly predicted, and is defined as

$$\text{recall} = \frac{TP}{TP + FN}. \tag{2}$$

A recall of 1 means that all objects were identified.

Precision, on the other hand, is a measure of how many predicted objects were actually objects. Precision is defined as

$$\text{precision} = \frac{TP}{TP + FP}. \tag{3}$$

A high precision value means the classifier did not mislabel many non-objects as objects, while a low precision value means that the classifier mislabeled many non-objects as objects.

The $F_\beta$ score is the weighted harmonic mean of precision and recall, which weights recall $\beta$-times higher than precision. The $F_\beta$ is defined as

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}}. \tag{4}$$

The $F_\beta$ score provides a single metric that can be used to evaluate classification performance.

MCC also provides a single metric for evaluating classification performance, and it has been shown to be reliable for imbalanced classification [15]. MCC is a metric that gives a high score if the method is able to achieve an accurate value in each of the four categories of the confusion matrix, and is defined as

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP + FP) \cdot (TP + FN) \cdot (TN + FP) \cdot (TN + FN)}}. \tag{5}$$

The MCC ranges between $-1$ and $+1$. Positive values indicate a positive correlation between the predictions and the true labels, with $+1$ being perfect prediction. A value of 0 indicates that the prediction is no better than random predictions. Negative values indicate a negative correlation between the predictions and the true labels, with $-1$ indicating complete disagreement between the predictions and the true labels.

Imbalanced Learning

When training on data with a large class imbalance—such as the fish, insect, and drone detection datasets used in this work—many traditional machine learning techniques will fail to identify the minority class(es) [16–18]. Consider one formulation of the optimization

problem that must be solved for a non-separable Support Vector Machine [19]:

$$\min \|\beta\| \quad \text{subject to} \begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \zeta_i \ \ \forall i \\ \zeta_i \geq 0, \ \sum \zeta_i \leq \text{constant} \end{cases}$$

where $\beta$ and $\beta_0$ are the normal vector and y-intercept of the optimized hyperplane (decision boundary), $x_i$ is measurement vector $i$ with associated binary label $y_i = \pm 1$, and $\zeta_i$ is a slack variable that allows for individual points to be on the "wrong" side of the hyperplane. Note that in this formulation, each input vector exerts the same amount of force on the hyperplane. This can cause the majority class to overwhelm the minority class, resulting in a classifier that skews heavily towards classifying new points as part of the majority class.

Several techniques have been developed to improve results in situations where the classes are imbalanced [20]. One common method is to weight the classes according to their distributions by imposing a higher cost for misclassifying vectors from the minority class. Two other methods are sampling and boosting [21]. RUSBoost [22], which is used in this dissertation, combines random undersampling and boosting to improve classification performance on imbalanced data. The primary methods used to deal with class imbalance in this work are undersampling, oversampling, and class weighting, because they can all be applied to existing classification algorithms without modification.

Undersampling Random undersampling [17] is a common method for working with imbalanced datasets, as it is a simple way to help balance class proportions. Given a dataset $S_0 = S_{maj} \cup S_{min}$ composed of a majority class $S_{maj} \subset S_0$ and a minority class $S_{min} \subset S_0$, random undersampling removes a random subset $S_{remove} \subset S_{maj}$ of the majority class to create a new dataset $S_1 \subset S_0 = (S_{maj} \setminus S_{remove}) \cup S_{min}$. Random undersampling's main drawback is that removing examples from the majority class results in loss of information [23]; however, random undersampling has been show to work well in practice [24]. In addition

to reducing class imbalance, random undersampling also reduces training time by reducing dataset size.

Although random undersampling appears to be the most common undersampling technique in practice, there are many other undersampling techniques that are more "intelligent" than random undersampling. Examples include Tomek Links [25], Condensed Nearest Neighbor Rule [26], and Evolutionary Undersampling [27]. For an excellent overview of undersampling techniques, see Alberto Fernández's book, *Learning From Imbalanced Data Sets* [28].

Oversampling Oversampling is the process of creating additional examples of the minority class. Oversampling can broadly be categorized into groups: techniques that operate in the original data-space, and techniques that operate in feature-space [29]. Techniques that operate in the original data-space transform the original data, which are usually images, to create plausible synthetic examples; common transformations include adding noise, shifting the image, and rotating the image, for example. For some types of data, label information may not be apparent in the original data-space, in which case oversampling in a feature-space is a better solution.

The simplest oversampling method is random oversampling, which creates identical copies of minority class instances. The main problem with random oversampling is that it does not add any additional information to the dataset. The most popular feature-space oversampling technique is the Synthetic Minority Over-Sampling Technique (SMOTE) [30]. The basic idea behind SMOTE is to create synthetic instances by interpolating between minority instances that are nearest neighbors. Most oversampling techniques are based on SMOTE [28]. As an example, Adaptive Synthetic Sampling (ADASYN) [31] modifies SMOTE by generating more synthetic examples minority instances that are harder to learn, i.e. they are closer to the decision boundary. For a listing and description of popular SMOTE

modifications, see *Learning From Imbalanced Data Sets* [28].

Boosting Boosting algorithms are iterative algorithms that increase the weights of misclassified points and decrease the weights of correctly classified points after each iteration [21]. This increases the probability that the misclassified examples will be correctly classified in subsequent iterations. AdaBoost [32] is one of the most common boosting techniques. In each iteration, Adaboost trains a weak learner and updates the weights of samples based upon the weak learner's hypothesis. After all iterations are done, the weak learners' hypotheses are weighted and combined to form the final hypothesis.

Non-stationary Data

One of the primary data analysis challenges in the fish, insect, and drone detection applications is that the background "clutter" changes over time. "Clutter" is defined as unwanted reflections from things like clouds, trees, or any object that is not a desired object. In a stationary environment with a moving object, clutter can be reduced by high-pass or band-pass filtering consecutive pulses, which removes reflections that are roughly constant between pulses. When the environment is non-stationary, the clutter is rarely the same between pulses.

*Non-stationary* data is particularly troublesome for machine learning algorithms. Most machine learning algorithms assume that the data is stationary: the data comes from a fixed probability distribution. In machine learning, non-stationary distributions are known as concept drift [33]. If the model is not updated as the data distribution changes, new data points will be misclassified.

## Summary of Projects and Contributions

My work focused on three different application areas: fish detection, insect detection, and UAV detection and characterization. All three of these applications have imbalanced datasets, which necessitated the use of the imbalanced learning techniques presented in the previous Imbalanced Learning section. The basic goal of all three applications is determining which data points have the object we're looking for (fish, insect, or UAV).

Although each application shared the same basic imbalanced machine learning techniques, the feature extraction techniques varied and were driven by the specifics of each application. For the fish detection problem, no feature extraction was performed due to the lack of physics-informed features that can be used to easily identify fish. In contrast, the insect detection and UAV detection applications heavily utilized feature extraction based upon physical intuition. In these applications, much of the intuition revolves around the fact that insect wings and UAV propellers oscillate. The lidar system [2, 40] used in the insect and UAV detection applications was designed to measure oscillations, whereas the lidars used in the fish detection application were not.

The work in this dissertation starts with fish detection in Chapter 2. Detecting fish has numerous important applications, including estimating biomass and removing invasive species. Traditionally, detecting fish in lidar data has been done manually [34–36]. Previous efforts to automate lidar processing for fisheries have used a combination of spatial filtering and thresholding to locate schools of mackerel [37] and swarms of zooplankton [38]. These attempts have been limited by the difficulty in optimizing filter parameters and threshold levels for the situation of interest. Due to the limitations of previous efforts, visual inspection was still the state-of-the-art. By using machine learning, we were able to bypass the manual parameter optimization process and instead let the machine learning algorithms optimize their own parameters. Although my methods were not able to reliably identify individual

fish schools, the methods were able to reliably identify regions of interest that humans can manually analyze. Manually analyzing the detected regions on of interest in the Gulf of Mexico dataset would have taken approximately 9.4 hours. Manually analyzing the entire dataset took around 14 hours. Only analyzing the regions of interest detected by my methods would reduce analysis time by 33%. Reducing the data analysis time will allow fisheries personnel to accomplish their work faster. Overall, my work showed that machine-learning-based fish-detection methods are a viable solution that should be explored further.

The dissertation then moves on to insect detection in Chapter 3. Reliably identifying insect signals in a vast array of imbalanced data is one of the greatest challenges in the field of entomological lidar. If insect signals can be reliably identified, entomological lidar can be used for long-term, non-invasive monitoring of insects, which is important for understanding biodiversity declines, insect abundance, and insect behavior[39]. Without automated methods, data analysis is a major bottleneck in entomological lidar applications. In this project, I contributed methods that can reliably detect insect signals using feature extraction and machine learning. Additionally, my in-depth analysis of which features are most salient for insect detection will help future researchers develop insect detection algorithms. My work is a step towards long-term insect monitoring, which is paramount to understanding insect behavior, abundance, and biodiversity loss. Overall, this work represents an advancement in understanding the utility of various features and machine learning methods for entomological lidar data analysis.

While transitioning from insect detection to UAV characterization, I made various upgrades to MSU's patented [40] wingbeat-modulation lidar [2] used in the insect detection and UAV characterization projects. Chapter 4 describes these upgrades, which involved significant hardware and software development.

Expanding upon the instrumentation and data analysis techniques developed in Chapter 3, UAV detection and characterization in Chapter 5 is the last project in this

dissertation. This work focuses on measuring UAVs' spectral signatures and using these signatures for detecting and characterizing UAVs. I show that my automated detection techniques can reliably detect UAVs in imbalanced lidar data. Additionally, I show that the wingbeat-modulation lidar method can extract spectral signatures caused by a UAV's rotating propellers and vibrating parts, and in some cases the propeller frequencies can be directly extracted. This work shows that the wingbeat-modulation method has potential to help distinguish between small UAVs and other flying objects, such as birds, which is a common problem being worked on by radar researchers [41]. With further development, the wingbeat-modulation lidar method can be utilized as part of a comprehensive UAV detection and characterization system.

Overall, my development and application of signal processing and machine learning techniques help advance the state-of-the-art in lidar-based object-detection applications.

14

# MACHINE LEARNING-BASED REGION OF INTEREST DETECTION IN AIRBORNE LIDAR FISHERIES SURVEYS

## Contribution of Authors and Co-Authors

Manuscript in Chapter 2

Author: Trevor C. Vannoy

Contributions: conceptualization, formal analysis, investigation, methodology, software, validation, visualization, writing—original draft, writing—review & editing

Co-Author: Jackson Belford

Contributions: investigation, formal analysis, methodology, software, writing—original draft, writing—review & editing

Co-Author: Joseph N. Aist

Contributions: investigation, formal analysis, methodology, software, writing—original draft, writing—review & editing

Co-Author: Kyle R. Rust

Contributions: investigation, formal analysis, methodology, software, writing—original draft, writing—review & editing

Co-Author: Michael R. Roddewig

Contributions: conceptualization, data curation, investigation, methodology, software, writing—original draft, writing—review & editing

Co-Author: James H. Churnside

Contributions: conceptualization, data curation, investigation, methodology, resources, writing—original draft, writing—review & editing

Co-Author: Joseph A. Shaw

Contributions: conceptualization, data curation, funding acquisition, investigation, methodology, project administration, resources, writing—original draft, writing—review & editing

Co-Author: Bradley M. Whitaker

Contributions: conceptualization, funding acquisition, methodology, project administration, resources, supervision, writing—original draft, writing—review & editing

Manuscript Information

Trevor C. Vannoy, Jackson Belford, Joseph N. Aist, Kyle R. Rust, Michael R. Roddewig, James H. Churnside, Joseph A. Shaw, Bradley M. Whitaker

Journal of Applied Remote Sensing

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

SPIE
Volume 15, Issue 3
https://doi.org/10.1117/1.JRS.15.038503
Copyright: © 2021 SPIE

## Abstract

Airborne lidar data for fisheries surveys often do not contain physics-based features that can be used to identify fish; consequently, the fish must be manually identified, which is a time consuming process. To reduce the time required to identify fish, supervised machine learning was successfully applied to lidar data from fishery surveys to automate the process of identifying regions with high probability of containing fish. Using data from Yellowstone Lake and the Gulf of Mexico, multiple experiments were run to simulate real-world scenarios. Although the human can not be fully removed from the loop, the amount of data that would require manual inspection was reduced by 61.14% and 26.8% in the Yellowstone Lake and Gulf of Mexico datasets.


## Introduction

Setting annual catch limits of fish is difficult because individual stocks vary widely as a result of environmental factors such as the availability of food, predation levels, and larval survival in addition to fishing pressure. Knowledge of these stocks is critical to setting appropriate catch limits, but this knowledge is often imprecise. For example, a 2003 paper in the journal Nature claimed that "large predatory fish biomass today is only about 10% of pre-industrial levels." [42] The conclusion was based on a Catch Per Unit Effort (CPUE) analysis, in which commercial fish landings and the time spent fishing are combined to estimate fish biomass. The analysis was controversial [43, 44], and speaks to the desirability of an estimate of biomass that does not depend on the fishery in question.

The two main fishery-independent survey types are controlled catches and acoustics. In controlled catches, it is the effort that is controlled. The same gear is used the same way in the same place at the same time of year each year to build up a time series of catches that can be related to increases or decreases in the fish stock. The objective is to statistically

sample the entire habitat, unlike the commercial fleet that goes where the fish are in greatest abundance. With acoustics, the ship can continuously sample along the track to obtain more data, increasing the statistical significance. Calibration factors allow conversion of acoustic backscatter to biomass. Generally, acoustic surveys also employ selective fishing to verify species identification and obtain information about the size, health, and reproductive status of the fish. These, and other less common techniques, are all ship based and are limited to the speed of a surface vessel, and these long surveys can be very expensive.

Airborne lidar for fishery surveys

Increasing the speed at which surveys can be conducted requires aerial surveys [45, 46], which have mostly been done visually or with camera systems. However, fish can be detected much deeper in the water column by lidar [47], and airborne lidar has been explored for fisheries surveys in oceans [35, 48–50] and lakes [51, 52]. Related applications of airborne lidar include mapping marine debris [53, 54], plankton layers [34, 55, 56], bubbles [57, 58], internal waves [54] in the ocean, underwater thermal vents[59], and water turbidity[36]. Initial research has started assessing the potential of spaceborne lidars for oceanic profiling of plankton and other biological and particulate matter [60–68].

All airborne lidar applications produce large amounts of data. Unlike lidar bathymetry [69–71] and ocean lidar profiling, [36, 72] most lidar data for fishery surveys do not contain physics-based features that can be used to easily identify fish or other objects of interest. As a result, identifying and marking the positions of fish in lidar images can take 10-20 minutes per hour of lidar data. Data with no fish are faster to process than data with large numbers of fish, but that is about average. As a concrete example, the Gulf of Mexico survey [35] analyzed in this work contains approximately 55 hours of lidar data; visually inspecting this data took about 14 hours. Consequently, there is a need for automated methods that can identify regions of fish and reduce the amount of data that requires manual inspection.

Previous efforts to automate lidar processing for fisheries have used a combination of spatial filtering and thresholding to locate schools of mackerel [37] and swarms of zooplankton [38]. These have been limited by the difficulty in optimizing filter parameters and threshold levels for the situation of interest. The situation of interest includes the return strength and spatial variability of the background water return, as well as the strength and spatial characteristics of the target species. Due to the limitations of previous efforts, visual inspection is still the state-of-the-art. In an effort to develop an automated processing technique, this paper investigates the feasibility of using supervised machine learning to identify regions containing fish.

Supervised learning on imbalanced data

Supervised learning is a type of machine learning that learns how to map input data to output labels by training on example input-output pairs. As a result, supervised algorithms can be trained to minimize misclassification rates. Unsupervised algorithms, in contrast, are designed to find associations between input data and possible output groups. Without a knowledge of class labels, the training procedure cannot directly optimize for prediction accuracy [73]. For this purpose, our work focuses on supervised learning techniques. The wide variety of available supervised learning algorithms makes it infeasible to perform an exhaustive search. Therefore, we focus on comparing results obtained from a few well-known algorithms—support vector machines (SVM) [3], decision trees [6], linear discriminant analysis (LDA) [5], and neural networks [7]—and RUSBoost [22], an algorithm designed specifically for imbalanced data.

We acknowledge that the ability of supervised classifiers to produce more accurate results comes at a cost: they require labeled data for the training process. Because of this, it is impossible to develop a completely autonomous detection system using only supervised classification. Prior to completing the training process, some data need to be manually

inspected and labeled so the classifiers have a ground truth to learn from. For example, during a multi-day lidar campaign, a human may manually inspect the lidar results from the first day. The labels from the first day can then be used to create a classifier for automated detection during subsequent days. It is also important to note that since ground truth labels are prone to human error, the reported classification results may be imperfect. However, this does not always have negative consequences. For example, classifiers sometimes find a fish that a human missed. Such situations negatively affect performance metrics, but ultimately result in more fish detections.

When training on data with a large class imbalance, like lidar data from fishery surveys where there are very few fish examples, many traditional machine learning techniques will fail to identify the minority class(es) [16–18]. Consider one formulation of the optimization problem that must be solved for a non-separable Support Vector Machine [19]:

$$\min \|\beta\| \quad \text{subject to} \begin{cases} y_i(x_i^T\beta + \beta_0) \geq 1 - \zeta_i \ \ \forall i \\ \zeta_i \geq 0, \ \sum \zeta_i \leq \text{constant} \end{cases}$$

where $\beta$ and $\beta_0$ are the normal vector and y-intercept of the optimized hyperplane (decision boundary), $x_i$ is measurement vector $i$ with associated binary label $y_i = \pm 1$, and $\zeta_i$ is a slack variable that allows for individual points to be on the "wrong" side of the hyperplane. Note that in this formulation, each input vector exerts the same amount of force on the hyperplane. This can cause the majority class to overwhelm the minority class, resulting in a classifier that skews heavily towards classifying new points as part of the majority class.

Several techniques have been developed to improve results in situations where the classes are imbalanced [20]. One common method is to weight the classes according to their distributions by imposing a higher cost for misclassifying vectors from the minority class. Two other methods are sampling and boosting[21]. RUSBoost[22], which is used in this

paper, combines random undersampling and boosting to improve classification performance on imbalanced data. The primary methods used to deal with class imbalance in this study are undersampling and class weighting, as they can both be applied to existing classification algorithms without modification.

## Lidar systems

Currently, there are no commercial lidar systems available for this application. Fish have been detected by commercial bathymetric lidars, but only as an artifact that must be eliminated during processing [74]. Terrestrial lidars not designed for bathymetry typically use laser wavelengths that do not penetrate significantly into water. However, the technology for a lidar to detect subsurface fish and plankton layers is much simpler than a bathymetric lidar, and several have been constructed in the United States [49, 51, 75], Russia [76–78], and China [79, 80]. Two of these are considered in this paper.

### NOAA lidar

The NOAA lidar was a down-looking, aircraft-based instrument. The light source for the transmitter of the lidar was a frequency-doubled, Q-switched Nd:YAG laser. The laser itself emitted linearly polarized, 532 nm light in 12 ns pulses at a repetition rate of 30 Hz. The laser light was directed through a polarization beam splitting cube (103 polarization extinction ratio), beam-steering mirrors, and a negative lens to ensure the light reaching the water surface was within the American National Standards Institute (ANSI) standards for exposure to laser light [81]. The pulse energy at the output of the transmission optics was 100 mJ. On the receiver side, the lidar employed two channels for measuring the return light, one for the component of the return with the same polarization as the transmitter (co-polarized return), and the other for its orthogonal polarization (cross-polarized return). For each channel, the received light was collected by a telescope, filtered by a 1 nm bandwidth interference

filter, and detected by a photomultiplier tube (PMT). The 17 mrad field of view of each telescope was set to match the transmitter beam divergence. A plastic-film linear polarizer was installed at the front of each telescope to select the appropriate linear polarization state. The photomultiplier signal was amplified through a logarithmic amplifier and digitized at 1 GHz. For this study, the lidar was mounted in the floor of a small twin-engine aircraft and flown at an altitude of approximately 300 m over the ocean surface. The lidar system was pointed approximately 15° from nadir to minimize contribution from air-water interface reflections.

MSU lidar

The MSU lidar was developed in 2014 and shares many similarities with the NOAA lidar. It used a 532 nm pulsed Nd:YAG laser with a 26.8 mJ pulse energy and 100 Hz repetition frequency. The lidar had a selectable 5 or 15 mrad FOV which at the typical flight altitude of 300 m gave a 1.5 or 4.5 meter diameter spot size, respectively. Two orthogonally polarized receiver telescopes were used to simultaneously measure the co- and cross-polarization return; photomultiplier tubes and synchronized 800 MSPS digitizers converted the returned optical energy to digital numbers, which were stored for later analysis [51]. This lidar was used extensively in Yellowstone National Park to map lake trout spawning locations [52], recorded the first known lidar detection of an underwater thermal vent [59], and characterized the lidar attenuation coefficient [36].

## Datasets

Two airborne lidar datasets were used in this study: one comprising lake trout in Yellowstone lake [52], and the other comprising flying fish, schools of unidentified fish, jellyfish, and plankton in the Gulf of Mexico[35].

Yellowstone Lake Dataset

Discovered in 1994 [82], invasive lake trout are now the top carnivore in Yellowstone Lake, Wyoming, USA [83, 84]—threatening the ecological balance of Yellowstone National Park. The Yellowstone Lake survey [52] was conducted to determine if airborne lidar could be used to detect the invasive lake trout. This dataset was selected for this work due to the application's ecological significance and availability of the data.

The survey was conducted with daytime flights during lake trout spawning season in 2015 and 2016 [52] using the airborne lidar system described in the MSU lidar section. Roddewig et al. [52] manually identified fish in the lidar data, which we used as ground truth labels. A summary of the Yellowstone Lake dataset is shown in Table 1, and a representative example of lidar data is shown in Fig 3.

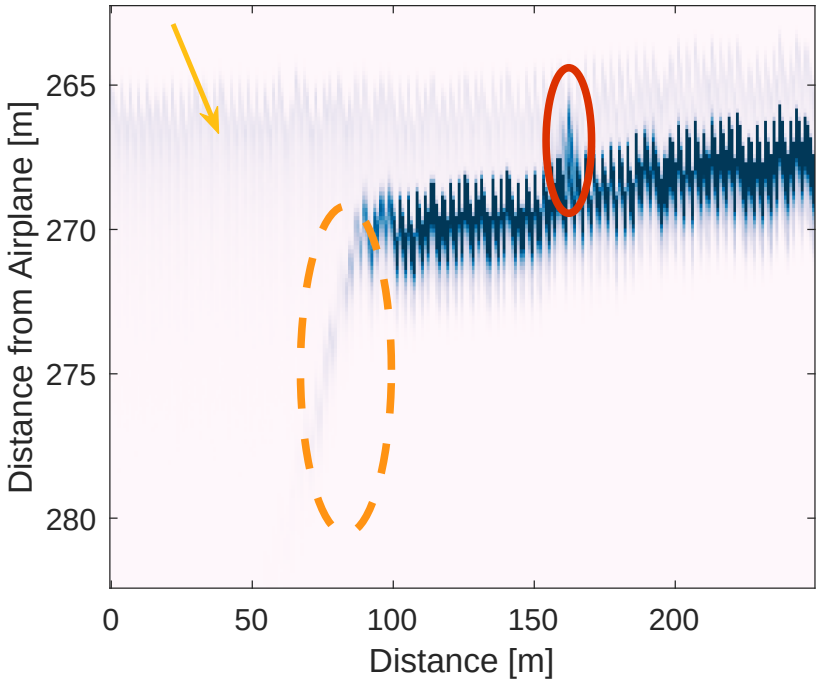



Figure 3: Example cross-polarized lidar data from Yellowstone Lake. Darker values indicate higher reflected radiance; radiance values were compressed to increase contrast. The light patch indicated by the yellow arrow is the surface of the water. The start of an underwater shelf is indicated by the orange dashed ellipse. The red ellipse highlights a typical fish hit, which appears as a vertical spike above the shelf.

Table 1: Summary of data collected at Yellowstone Lake[51], showing the number of instances (lidar shots), dimensions (samples per shot), and the number of instances containing fish. The percentage of fish instances are shown in parentheses. A single fish often shows up in multiple adjacent instances.

| | Instances | Dimensions | Fish Instances |
|---|---|---|---|
| 2015 Flight | 108778 | 2048 | 401 (0.369%) |
| 2016 Flight | 192201 | 2048 | 227 (0.118%) |

Gulf of Mexico Dataset

The Gulf of Mexico study [35] was an investigation into the distribution of near-surface fish after the Deepwater Horizon oil spill. Of particular interest was any effect on the distribution of flying fish, which were thought to be vulnerable because they pass through the oil floating on the surface. This dataset was selected for this work because flying fish created a large number of individual fish returns.

The survey was conducted over 12 days with images being taken both at night and during the day, using the lidar system described in Sec. 2. The data contained instances of flying fish, schools of fish, jellyfish, and plankton layers, which were manually labeled in a previous study [35]. A summary of the Gulf of Mexico dataset is shown in Table 2, and a representative example of lidar data is shown in Fig 4.

The original data was in the form of 1000x1000 pixel PNG images, with multiple images for each day; for each PNG image, object labels were denoted by start and end columns in the image. The original data were transformed such that each day became a separate MATLAB mat file. The labels were placed in a matrix, where each row corresponded to one of the 4 different objects of interest, and each column indicated whether an object was present. To be more consistent with the Yellowstone Lake data, plankton labels were ignored, and the fish, fish school, and jellyfish labels were combined into a single binary label.

Table 2: Summary of data collected over the Gulf of Mexico[35], showing the number of instances (lidar shots), dimensions (samples per shot), and the number of instances containing fish, schools of fish, jellyfish, and plankton layers, respectively. The percentages of each instance type are shown in parentheses. Objects of interest often occur across multiple adjacent instances. Some instances contain multiple types of objects of interest. Day 11 is excluded due to lack of ground truth labels.

| | Instances | Dimensions | Fish Instances | Fish School Instances | Jellyfish Instances | Plankton Layer Instances | All Fish Instances |
|---|---|---|---|---|---|---|---|
| Day 1 | 747284 | 1000 | 7232 (0.968%) | 217 (0.0290%) | 0 | 106854 (14.3%) | 7449 (0.997%) |
| Day 2 | 469363 | 1000 | 383 (0.0816%) | 325 (0.0692%) | 69 (0.0147%) | 44633 (9.51%) | 777 (0.166%) |
| Day 3 | 637132 | 1000 | 576 (0.090%) | 103 (0.0162%) | 0 | 74951 (11.8%) | 679 (0.107%) |
| Day 4 | 501563 | 1000 | 236 (0.047%) | 286 (0.057%) | 36 (0.00718%) | 41812 (8.34%) | 588 (0.111%) |
| Day 5 | 255121 | 1000 | 314 (0.123%) | 76 (0.0298%) | 0 | 8800 (3.45%) | 390 (0.153%) |
| Day 6 | 509049 | 1000 | 226 (0.0444%) | 87 (0.0171%) | 0 | 8541 (1.68%) | 313 (0.0615%) |
| Day 7 | 370976 | 1000 | 630 (0.170%) | 31 (0.00836%) | 0 | 20502 (5.53%) | 661 (0.178%) |
| Day 8 | 648664 | 1000 | 1919 (0.296%) | 126 (0.0194%) | 0 | 9087 (1.40%) | 2045 (0.315%) |
| Day 9 | 574674 | 1000 | 697 (0.121%) | 618 (0.108%) | 0 | 102382 (17.8%) | 1315 (0.229%) |
| Day 10 | 563762 | 1000 | 163 (0.0289%) | 67 (0.0119%) | 0 | 27754 (4.92%) | 230 (0.0408%) |
| Day 12 | 682304 | 1000 | 494 (0.0724%) | 206 (0.0302%) | 0 | 5857 (0.858%) | 700 (0.103%) |

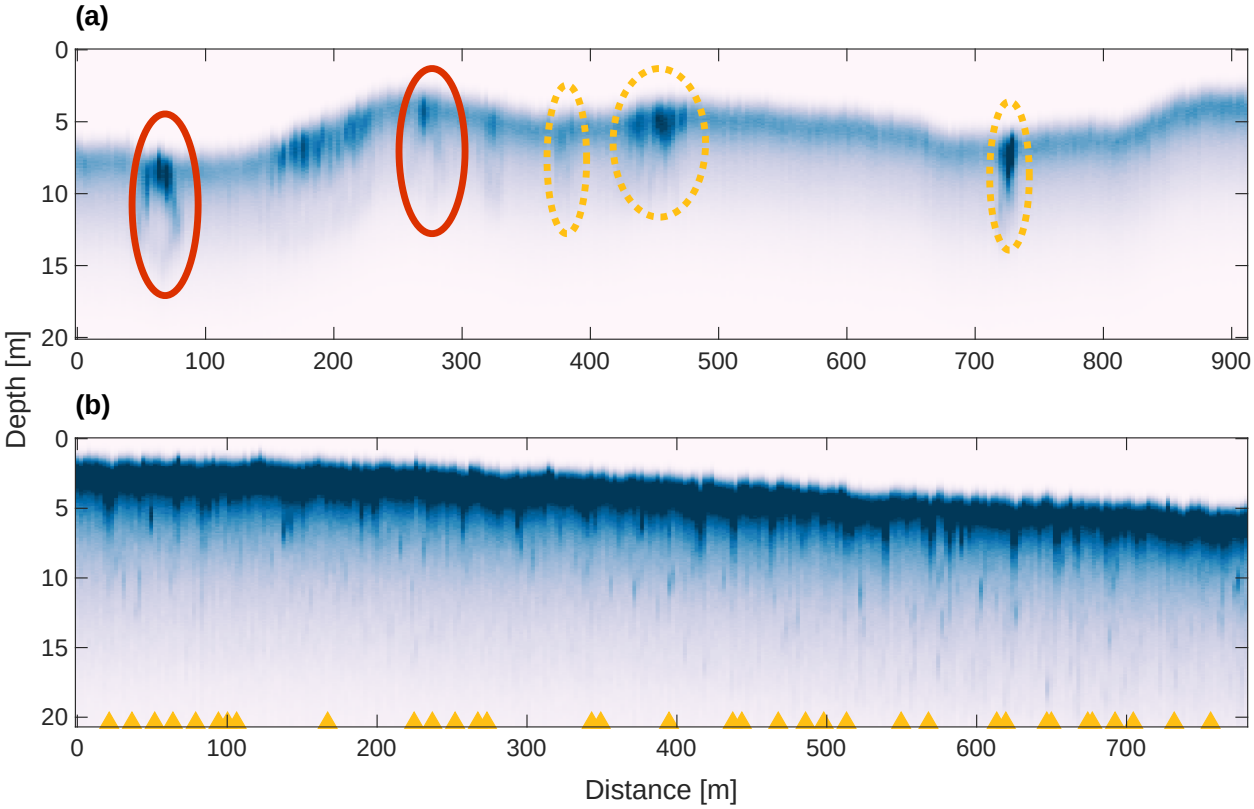



Figure 4: Example cross-polarized lidar data from the Gulf of Mexico. **(a)** Data from day 2. Hollow jellyfish aggregations [85] are annotated with red ellipses, while fish schools are highlighted with yellow dotted ellipses. **(b)** Data from day 1. The yellow triangles indicate lidar shots where a single fish was manually found and labeled. The colormap's maximum was scaled to increase contrast.

## Methods

Two sets of experiments were run on the Yellowstone Lake and Gulf of Mexico datasets: one where classifiers were trained on the first day of each dataset and tested on the remaining days, and one where classifiers were trained on 80% of each dataset and tested on the remaining 20%. The first experiment was intended to simulate a multi-day campaign where the researchers want to train a classifier after the first day, and then use that classifier to predict fish regions during subsequent days. Although the second experiment required more manually labelled data than the first experiment, it can be used to train a classifier for subsequent campaigns in the same location using the same lidar instrument; the second experiment also has the potential to find fish that the manual labels missed.

The following sections describe the algorithms, evaluation metrics, and procedure used in the experiments.

### Classification algorithms

As stated in Sec. 2, the following classifiers were used in the experiments: SVM [3], decision trees [6], LDA [5], neural networks [7], and RUSBoost [22]. SVM, decision trees, and LDA are all common classifiers so we refer the reader to the corresponding references for more details. The neural network architecture that was used is shown in Fig. 5.

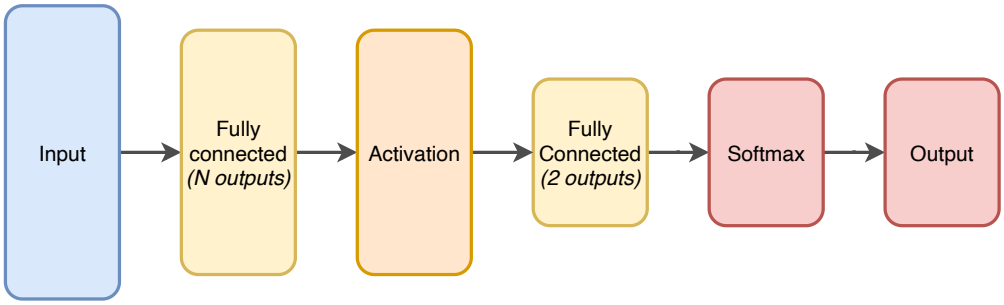



Figure 5: MATLAB's `fitcnet` neural network architecture. The size of the first fully connected layer and the activation function were tuned, as described in Sec. 2. The second fully connected layer has 2 outputs, corresponding to the 2 classes—"no fish" and "fish".

Unlike the other algorithms used in this paper, RUSBoost is an ensemble technique specifically designed for imbalanced learning. RUSBoost introduces random undersampling into the AdaBoost.M2 [32] boosting algorithm, with the goal of providing better performance than undersampling and boosting provide individually. Concretely, RUSBoost randomly undersamples the majority class at the beginning of each iteration of AdaBoost. Random undersampling and boosting are described in the following paragraphs.

Random Undersampling Random undersampling [17] is a common method for working with imbalanced datasets, as it is a simple way to help balance class proportions. Given a dataset $S_0 = S_{maj} \cup S_{min}$ composed of a majority class $S_{maj} \subset S_0$ and a minority class $S_{min} \subset S_0$, random undersampling removes a random subset $S_{remove} \subset S_{maj}$ of the majority class to create a new dataset $S_1 \subset S_0 = (S_{maj} \setminus S_{remove}) \cup S_{min}$. Random undersampling's main drawback is that removing examples from the majority class results in loss of information [23]; however, random undersampling has been show to work well in practice [24]. In addition to reducing class imbalance, random undersampling also reduces training time by reducing dataset size; given the size of the Gulf of Mexico dataset in Sec. 2, reducing training time is advantageous.

Boosting Boosting algorithms are iterative algorithms that increase the weights of misclassified points and decrease the weights of correctly classified points after each iteration [21]. This increases the probability that the misclassified examples will be correctly classified in subsequent iterations. AdaBoost [32] is one of the most common boosting techniques. In each iteration, Adaboost trains a weak learner and updates the weights of samples based upon the weak learner's hypothesis. After all iterations are done, the weak learners' hypotheses are weighted and combined to form the final hypothesis.

All of the classifiers used in this study are available in MATLAB R2021a. The classifiers were implemented with the following functions: `fitclinear` for the SVM, `fitcdiscr` for

LDA, `fitctree` for the decision tree, `fitcnet` for the neural network, and `fitcensemble` for RUSBoost. The base learner for RUSBoost was a decision tree. By default, MATLAB's implementation of RUSBoost undersamples the majority class to give a perfectly balanced class ratio.

Evaluation metrics

In this work, classification performance was measured with recall, precision, and the $F_3$ score, which is a particular combination of recall and precision. All of these metrics range between 0 and 1, with 1 being the best. These metrics can be understood in terms of the confusion matrix, shown in Fig 6.

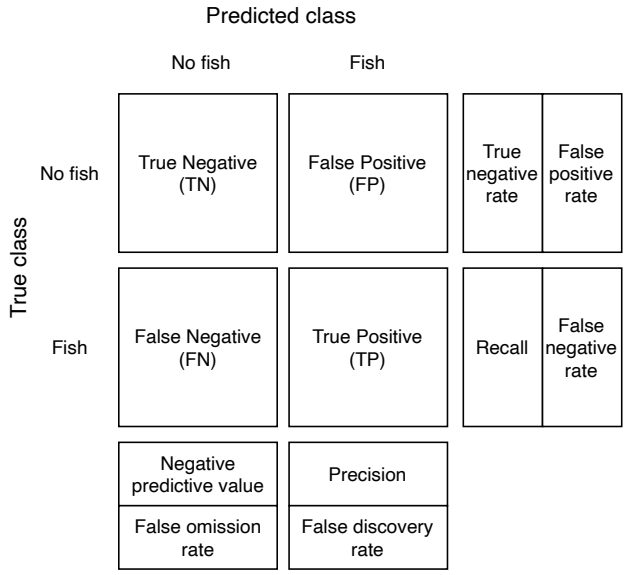



Figure 6: Confusion matrix terminology. "No fish" is considered the negative class, and "fish" is considered the positive class.

Recall is a measure of how many true fish were correctly predicted, and is defined as

$$\text{recall} = \frac{TP}{TP + FN}. \tag{6}$$

A recall of 1 means that all fish were identified.

Precision, on the other hand, is a measure of how many identified fish were actually fish. Precision is defined as

$$\text{precision} = \frac{TP}{TP + FP}. \tag{7}$$

A high precision value means the classifier did not mislabel many lidar shots as containing fish, while a low precision value means that the classifier mislabeled many non-fish as fish.

Since there are very few fish in the datasets, finding the majority of the fish is more important than accidentally classifying non-fish as fish; however, having a very low precision value is not desirable, as that would mean researchers would have to manually inspect and discard large amounts of unimportant data.

The $F_\beta$ score is the weighted harmonic mean of precision and recall, which weights recall $\beta$-times higher than precision. Since recall is more important than precision in our application, the $F_3$ score is used, which weights recall 3-times higher than precision. The $F_\beta$ and $F_3$ scores are defined as

$$F_\beta = (1 + \beta^2)\frac{\text{precision} \cdot \text{recall}}{\beta^2 \cdot \text{precision} + \text{recall}} \tag{8}$$

$$F_3 = 10 \cdot \frac{\text{precision} \cdot \text{recall}}{9 \cdot \text{precision} + \text{recall}}. \tag{9}$$

The $F_3$ score provides a single metric that can be used to evaluate classification performance.

Recall, precision, and $F_3$ score were computed per-shot and per-region. Per-shot metrics were computed directly using the ground truth and predicted labels for each shot. For the per-region metrics, labels were grouped into 1000-shot windows. For the ground truth labels, a region was considered a region of interest (ROI) if at least 1 fish label was present. For the predicted labels, the number of fish labels needed for an ROI was tuned as described in Sec. 2.

Preprocessing

Since this manuscript focuses on introducing machine learning to fish detection in airborne lidar data, we decided to perform minimal preprocessing on the raw data; most of the preprocessing steps used here are standard practice in previous studies [49, 52]. No feature extraction was performed.

Fig. 7 shows an overview of the preprocessing procedure. The first two preprocessing steps—surface detection and correction—were performed to compensate for changes in airplane elevation and ensure that a given row in the lidar image nominally corresponded to a given water depth. The depth adjustment step was performed to reduce the dimensionality of the data and to focus on regions most likely to contain meaningful lidar data. Save for the surface detection and smoothing, all processing was performed on the cross-polarized data because it provides better contrast between fish and the background water column.

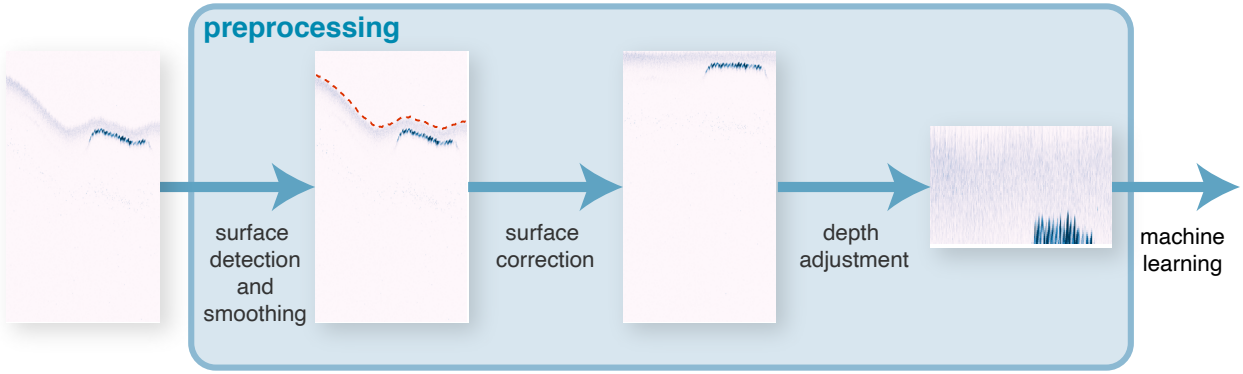



Figure 7: Data preprocessing overview. The original lidar data (far left) was first run through an algorithm that detected and smoothed the surface of the water (dashed red line). Once the surface was detected, surface correction was applied to compensate for changes in airplane elevation, and data above the surface of the water were removed. Finally, the depth of the image was adjusted based on the lidar's penetration depth or prior knowledge about the expected depth of fish.

Surface detection and smoothing The surface of the water in each lidar shot was detected by finding the maximum return in the copolarized channel, and then moving up towards the

lidar until 25% of the maximum return was reached:

$$s_{raw}[m] = \max_n \{n \mid y_n[m] \leq 0.25 \cdot \max(\mathbf{y}[m])\}, \tag{10}$$

where $s_{raw}[m]$ is the surface index for shot $m$, $\mathbf{y}[m]$ is the vector of return values for shot $m$, and $y_n[m]$ is return value at range bin $n$. This is similar to what was done in a previous study by Roddewig et al. [51]. As noted in the previous study [51], this algorithm might fail when the volumetric scattering from the water below the surface is stronger than the surface return [86]; however, the lidar instruments used in this study are not designed for high-accuracy bathymetry, so an occasional error in surface detection is acceptable. Previous studies [78, 87, 88] have shown that the surface return preserves polarization, which is why the copolarized channel was used for surface detection.

Upon visual inspection, the surface location detected by Eq. 10 occasionally varied more drastically between adjacent shots than would be expected. To compensate for this, the surface was smoothed with a 10-tap moving average filter:

$$s_{smooth}[m] = \frac{1}{10} \sum_{l=0}^{9} s_{raw}[m-l]. \tag{11}$$

A filter length of 10 was chosen visually to give a line that smoothly followed the surface.

The Yellowstone data contained atmospheric returns that were incorrectly detected as the water surface. To overcome this, the first 600 and 512 range bins were skipped during surface detection in the 2015 and 2016 data, respectively. For the sake of simplicity, the number of range bins to skip was chosen via visual inspection for each dataset.

Surface correction Once the surface was found, each lidar shot was shifted up so the surface started at the first row, then the lidar shot was padded with zeros at the bottom, maintaining the original number of rows. Surface correction was performed to ensure that

each row in the image corresponded to a specific distance beneath the surface, regardless of the altitude of the aircraft. Thus the surface correction allowed each row to be interpreted by the machine learning algorithm as a specific feature, in this case the intensity of the lidar return corresponding to a specific depth in the water.

Depth adjustment After surface correction, the height of the image was reduced based on the penetration depth of the lidar and the expected fish depth. In the Yellowstone Lake data, the image was reduced to 60 rows, which corresponded to a depth of approximately 8.65 meters. The Gulf of Mexico data was reduced to 150 rows, which corresponded to a depth of approximately 17.3 meters. The depth adjustment accomplished three main purposes. First, the adjustment reduced the dimensionality of the problem, which decreases the complexity of the machine learning algorithms. Second, the adjustment considered the physical limitations of the depth penetration of the lidar systems used in this study [36, 49], as well as the expected physical locations of fish. Finally, truncating the bottom rows ensured that the algorithm is not influenced by the manually padded zeros introduced in the surface correction process. An example of Yellowstone data after preprocessing is shown in Fig 8.

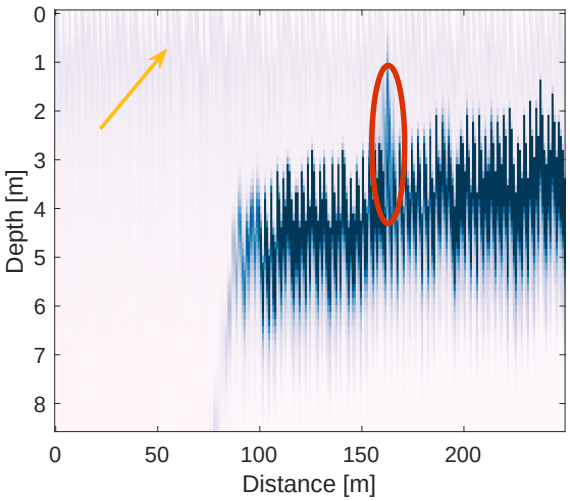



Figure 8: Lidar data from Fig 3 after preprocessing. The height-corrected surface of the water is indicated by the green arrow, and a fish hit is highlighted by the blue ellipse. Radiance values were compressed to increase contrast.

Training procedure

After preprocessing was performed, the cross-polarized data were split into training and testing sets. Rather than randomly selecting individual shots for the training and testing sets, the data were split by region of interest; we did this for two reasons: 1) since fish often span multiple shots, we did not want to split a single fish hit between the training and testing sets; and 2) splitting the data by region of interest allows us to tune the number of predicted labels required for a region to be considered a region of interest. The data splitting procedure was the same for both the Yellowstone and Gulf of Mexico datasets.

When creating the training and testing data, each day was split into regions of 1000 adjacent shots. This matches the typical window size used during visual analysis. In general, the last region in each day contained less than 1000 shots because the number of shots in the day was not a multiple of 1000.

For experiments that used the first day as training data, the first day was split by region into 3 folds for cross validation. The folds were randomly selected, and they were stratified such that the class proportions in each fold were approximately the same.

For the experiments that randomly selected the training and testing data, 80% of the regions were randomly selected as training data, while the remaining 20% were put into the testing set. The training and testing sets were stratified such that the class proportions in each were approximately the same. The training set was then split by region into 3 folds for cross validation. The regions in each fold were randomly selected, and the folds were stratified to maintain class proportions.

Parameter tuning Parameter tuning was performed using 3-fold cross validation on the training sets. In addition to tuning each classifier's hyperparameters, we also tuned the undersampling percentage and the number of predicted labels needed for a region to be considered a region of interest.

**Undersampling** Using the default parameters for each classifier, we tuned the undersampling ratio by performing a grid search between $[0, 0.95]$ in increments of 0.05. For each grid point, the classifiers were trained using 3-fold cross validation. The $F_3$ score was recorded for each fold; the scores were then averaged to give an $F_3$ score for the grid point. The undersampling ratio that resulted in the maximum $F_3$ score was chosen for each classifier.

**Model hyperparameters** After tuning the undersampling ratios, we tuned each classifier's hyperparameters using MATLAB's Bayesian optimization function, `bayesopt`. For classifiers that supported a cost matrix, the cost of missing a fish was also tuned. Table 3 shows the hyperparameters that were tuned for each classifier. We used the following `bayesopt` settings:

- `IsObjectiveDeterministic` = true
- `AcquisitionFunctionName` = expected-improvement-plus
- `MaxObjectiveEvaluations` = 20

When a classifier achieved a lower $F_3$ score after hyperparameter tuning, the model parameters were set back to their defaults and only the cost of missing a fish was tuned. When tuning hyperparameters, we undersampled the cross validation training sets by the optimal ratios found via the previous grid search.

**ROI label tuning** As will be seen in Sec. 2, the classifiers predicted many false positives. Often, the false positive predictions were in regions that contained fish. When assessing the classifier's ability to correctly predict fish-containing regions of interest, one needs to know when to flag a region as a predicted region of interest. Towards this end, we tuned the number of predicted fish labels per region that are needed to flag a region as a true positive. After training the tuned classifiers from the previous steps on the cross validation training sets, the number of positive labels needed per region was swept between 1 and 100. The number of labels that resulted in the highest average $F_3$ score was chosen for each classifier.

Table 3: Hyperparameter search ranges for the classifiers used in this study. Most of the model-specific parameter search ranges were set to the default search ranges in MATLAB R2021a. $|S|$ indicates the number instances in the dataset. A † indicates that the search values were logarithmically spaced. $\lfloor \cdot \rfloor$ is the floor function.

| SVM | | LDA | | Neural Net | |
|---|---|---|---|---|---|
| Lambda | $\frac{1}{\|S\|}[10^{-5}, 10^{5}]^{\dagger}$ | Delta | $[10^{-6}, 10^{3}]^{\dagger}$ | Layer size | $[10, 50] \in \mathbb{Z}$ |
| Regularization | {ridge, lasso} | Gamma | $[0, 1]$ | Activation | {relu, sigmoid, tanh} |
| FN cost | $[1, 20] \in \mathbb{Z}$ | FN cost | $[1, 20] \in \mathbb{Z}$ | | |
| **Decision Tree** | | **RUSBoost** | | | |
| Max number of splits | $[1, \|S\| - 1]^{\dagger} \in \mathbb{Z}$ | Learning cycles | $[10, 500]^{\dagger} \in \mathbb{Z}$ | | |
| Min leaf size | $[1, \lfloor \|S\|/2 \rfloor]^{\dagger} \in \mathbb{Z}$ | Learning rate | $[10^{-3}, 1]^{\dagger}$ | | |
| Split criterion | {gdi, deviance} | Max number of splits | $[1, \|S\| - 1]^{\dagger} \in \mathbb{Z}$ | | |
| FN cost | $[1, 20] \in \mathbb{Z}$ | Min leaf size | $[1, \lfloor \|S\|/2 \rfloor]^{\dagger} \in \mathbb{Z}$ | | |
| | | Split criterion | {gdi, deviance} | | |
| | | FN cost | $[1, 20] \in \mathbb{Z}$ | | |

## Results

Experimental results obtained on both experiments are reported below for both datasets. Due to space considerations, performance metrics are only reported for the ROI results. The per-shot results generally identified extra fish per region; a representative example is shown in Sec. 2.

### First day training set

As was discussed in Sec. 2, the first set of experiments were designed to simulate the real-world scenario of training a classifier after the first flight, then using that classifier during subsequent flights during the same campaign. This section shows the results of those experiments on the Yellowstone Lake and Gulf of Mexico datasets.

Yellowstone Lake Table 4 shows the hyperparameters obtained by following the procedure in Sec. 2 using the 2015 Yellowstone Lake flight as the training data. The

Table 4: Hyperparameter values for the "first day" experiment on Yellowstone Lake. A $*$ indicates that the parameter values were left at their default values because hyperparameter tuning did not improve classification results.

| SVM | | LDA | | Neural Net | |
|---|---|---|---|---|---|
| Lambda | $9.193 \times 10^{-6*}$ | Delta | 0.0007181 | Layer size | 33 |
| Regularization | ridge* | Gamma | 0.99884 | Activation | relu |
| FN cost | 1* | FN cost | 20 | Undersampling | 0.9 |
| Undersampling | 0 | Undersampling | 0.45 | # labels for ROI | 9 |
| # labels for ROI | 1 | # labels for ROI | 25 | | |
| **Decision Tree** | | **RUSBoost** | | | |
| Max number of splits | 2 | Learning cycles | 11 | | |
| Min leaf size | 53 | Learning rate | 0.59087 | | |
| Split criterion | gdi | Max number of splits | 2 | | |
| FN cost | 19 | Min leaf size | 3 | | |
| Undersampling | 0.95 | Split criterion | gdi | | |
| # labels for ROI | 43 | FN cost | 16 | | |
| | | Undersapmling | 0.45 | | |
| | | # labels for ROI | 43 | | |

SVM never predicted any fish during hyperparameter tuning, so the default parameter values were used for the final training.

Figure 9 shows the ROI cross validation results obtained on the 2015 flight. The parameters in Table 4 were used when training the classifiers. As seen in Fig. 9, the SVM was uninformative. LDA, the neural network, and RUSBoost properly identified all fish-containing regions. The decision tree only missed one fish-containing region.

The neural network achieved the highest $F_3$ score during cross validation; consequently, in a real-world scenario, the neural network would be chosen to make predictions during the following days of the campaign. However, as seen in Fig. 10 and Table 5, the neural network did not achieve the highest $F_3$ score on the testing data (2016 flight). LDA achieved the best $F_3$ score on the testing data, but achieved the second best score on the training data. As seen

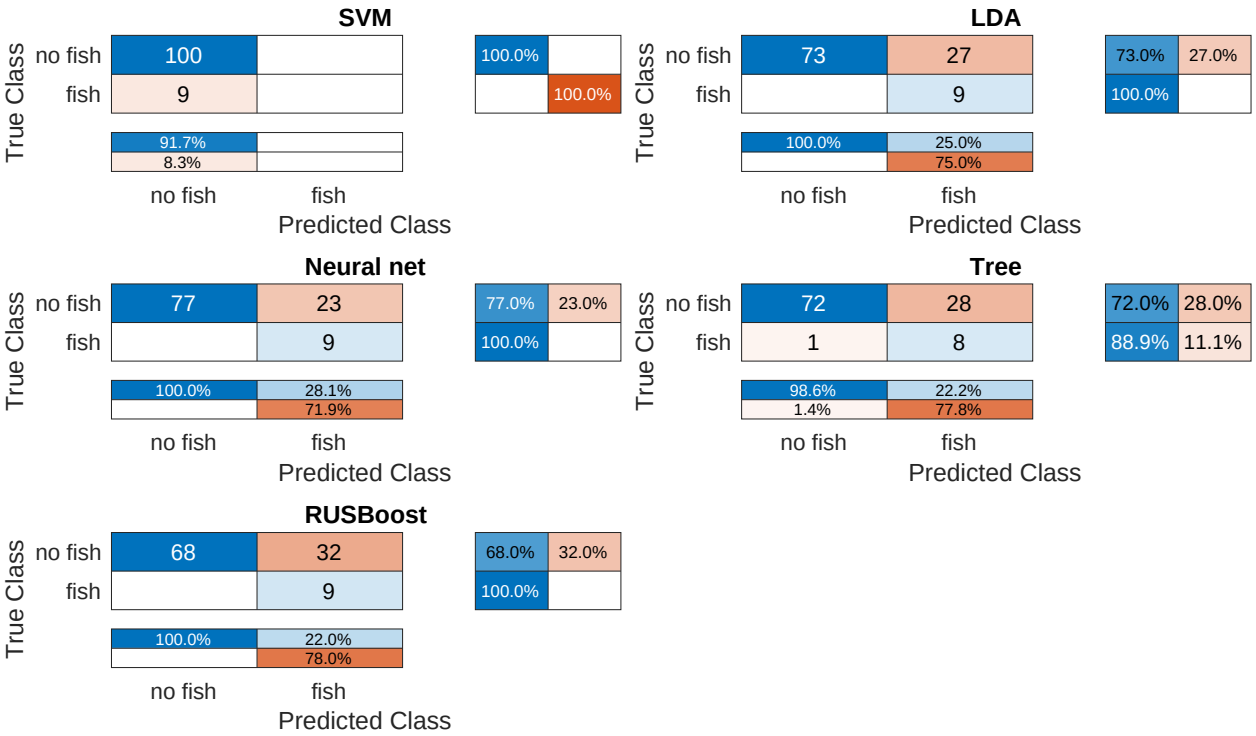



Figure 9: ROI cross validation results for the 2015 Yellowstone Lake flight.

in Table 5, LDA did not discard as many regions as the neural network did; compared to LDA, the neural network would have saved manual analysis time in a real-world campaign.

Table 5: $F_3$ scores achieved on the cross validation and holdout sets for the "first day" experiment on the Yellowstone Lake dataset. Additionally, the percent of regions that were discarded by each classifier are reported. The best $F_3$ scores are marked in bold.

| | SVM | LDA | Neural Network | Decision Tree | RUSBoost |
|---|---|---|---|---|---|
| Cross validation $F_3$ | 0 | 0.7692 | **0.7965** | 0.6838 | 0.7377 |
| Holdout $F_3$ | 0 | **0.3968** | 0.3333 | 0.3922 | 0.3876 |
| % regions discarded | 100 | 58.03 | 61.14 | 70.47 | 56.48 |

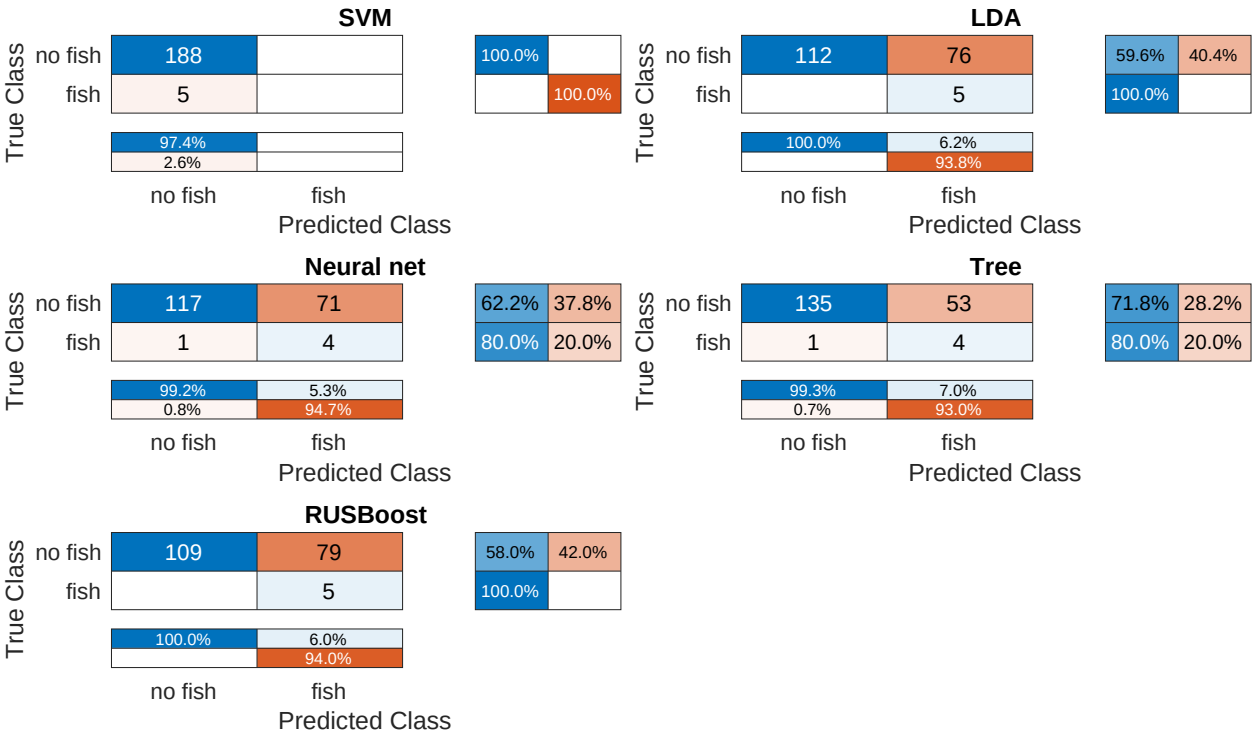



Figure 10: ROI holdout results for the 2016 Yellowstone Lake flight.

Gulf of Mexico Table 6 shows the hyperparameters obtained using day 1 of the Gulf of Mexico dataset as the training data. Once again, as seen in Fig. 11, the SVM was nearly uninformative. The cross validation performance of the other classifiers, however, are more varied than they were for the Yellowstone experiment in Sec 2. As seen in Fig. 11 and Table 7, RUSBoost achieved the highest $F_3$ score on the training data; however, this came at the cost of misclassifying 242 (87.1%) regions that did not contain fish. The decision tree, which had the second highest $F_3$ score, only misclassified 147 (52.9%) regions that did not contain fish. It is worth noting that day 1 contains more fish-containing regions than fishless regions, in spite of the fact that fish make up only 0.997% of the dataset (Table 2).

On the testing set (days 2–12), the decision tree achieved a higher $F_3$ score than RUSBoost did, as shown in Table 7. This is because the tree achieved a higher precision, which can be seen in Fig. 12; however, this came with the tradeoff of incorrectly identifying an additional 572 regions as containing fish, which would result in more manual analysis

Table 6: Hyperparameter values for the "first day" experiment on Gulf of Mexico dataset.

| SVM | | LDA | | Neural Net | |
|---|---|---|---|---|---|
| Lambda | $8.5538 \times 10^{-8}$ | Delta | $1.1273 \times 10^{-6}$ | Layer size | 42 |
| Regularization | lasso | Gamma | 0.81328 | Activation | tanh |
| FN cost | 11 | FN cost | 14 | Undersampling | 0.95 |
| Undersampling | 0 | Undersampling | 0.8 | # labels for ROI | 1 |
| # labels for ROI | 46 | # labels for ROI | 1 | | |
| **Decision Tree** | | **RUSBoost** | | | |
| Max number of splits | 693 | Learning cycles | 349 | | |
| Min leaf size | 347 | Learning rate | 0.77313 | | |
| Split criterion | gdi | Max number of splits | 735 | | |
| FN cost | 20 | Min leaf size | 4 | | |
| Undersampling | 0.8 | Split criterion | gdi | | |
| # labels for ROI | 1 | FN cost | 16 | | |
| | | Undersapmling | 0.8 | | |
| | | # labels for ROI | 1 | | |

time. Ultimately, RUSBoost's performance was not significantly worse than the decision tree's performance, and thus would still have been useful during a real-world campaign.

Table 7: $F_3$ scores achieved on the cross validation and holdout sets for the "first day" experiment on the Gulf of Mexico dataset. Additionally, the percent of regions that were discarded in the testing data by each classifier are reported. The best scores $F_3$ scores are marked in bold.

| | SVM | LDA | Neural Network | Decision Tree | RUSBoost |
|---|---|---|---|---|---|
| Cross validation $F_3$ | 0 | 0.3210 | 0.5568 | 0.7911 | **0.9143** |
| Holdout $F_3$ | 0 | 0 | 0.4354 | **0.7983** | 0.7419 |
| % regions discarded | 100 | 100 | 73.1915 | 12.7660 | 26.8085 |

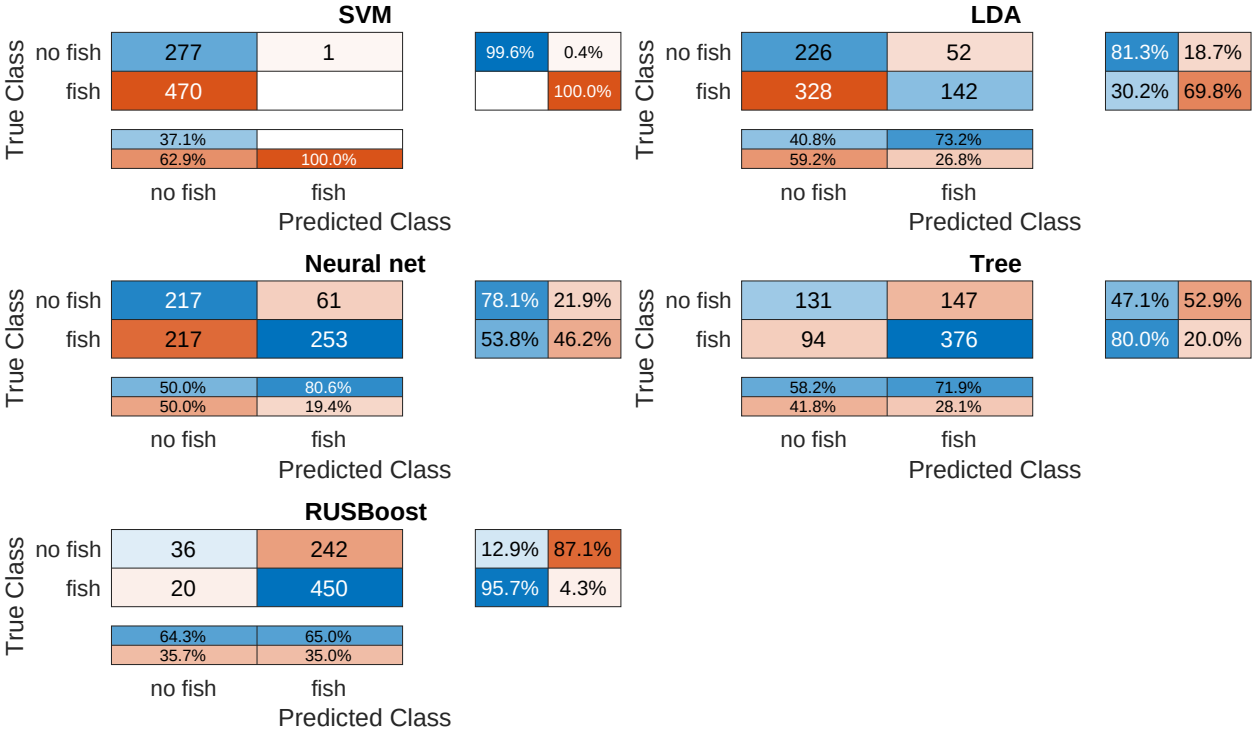



Figure 11: ROI cross validation results for day 1 of the Gulf of Mexico dataset.

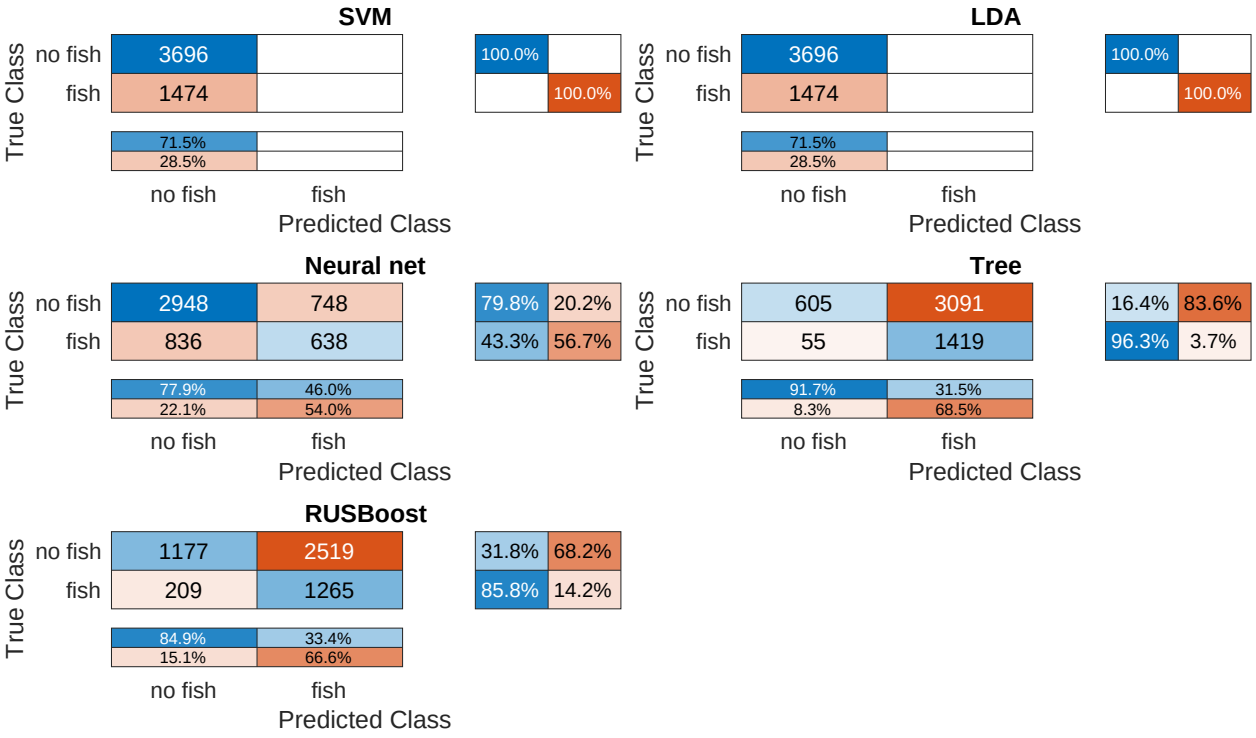



Figure 12: ROI holdout results for days 2–12 of the Gulf of Mexico dataset.

Randomized training set

The second set of experiments followed a more traditional approach: the training and test sets were randomly chosen with an 80/20 split. Compared to the "first day" experiments in Sec. 2, the training sets for the following experiments contained more data, and thus more examples of fish. Generally, training with more data produces more accurate models. The classifiers trained in these experiments could be used in future campaigns that use the same lidar instrument at the same location as the previous campaign. The results of these experiments are reported for both datasets in the following sections.

Yellowstone_Lake Table 8 shows the hyperparameters obtained by following the procedure in Sec. 2. The SVM did not perform better after hyperparameter tuning, so the default parameters were used. RUSBoost also did not perform better after the initial hyperparameter tuning. The RUSBoost hyperparameters were then set to defaults, and the false negative cost was tuned; this resulted in a higher $F_3$ score than using default parameters with a false negative cost equal to 1.

Figure 13 shows the cross validation results obtained using classifiers trained with the parameters in Table 8. As shown in Table 9, LDA achieved the highest $F_3$ score, followed by RUSBoost. RUSBoost, however, achieved a perfect recall, whereas LDA missed one fish-containing region.

Interestingly, as seen in Fig. 14, LDA missed all fish regions in the testing set, whereas RUSBoost found all of them. The neural network was the only other classifier to find any fish regions. If the best classifier was chosen based purely upon the cross validation $F_3$ scores, LDA would have been chosen, only to prove unsuccessful during testing.

Table 8: Hyperparameter values for the randomized training set experiment on the Yellowstone Lake dataset. A $*$ indicates that the parameter values were left at their default values because hyperparameter tuning did not improve classification results.

| SVM | | LDA | | Neural Net | |
|---|---|---|---|---|---|
| Lambda | $4.1497^{-6*}$ | Delta | 0.0014063 | Layer size | 41 |
| Regularization | ridge* | Gamma | 0.43906 | Activation | tanh |
| FN cost | 1* | FN cost | 18 | Undersampling | 0.9 |
| Undersampling | 0 | Undersampling | 0.45 | # labels for ROI | 16 |
| # labels for ROI | 1 | # labels for ROI | 31 | | |
| **Decision Tree** | | **RUSBoost** | | | |
| Max number of splits | 41 | Learning cycles | 100* | | |
| Min leaf size | 2 | Learning rate | 1* | | |
| Split criterion | deviance | Max number of splits | 1* | | |
| FN cost | 18 | Min leaf size | 1* | | |
| Undersampling | 0.95 | Split criterion | gdi* | | |
| # labels for ROI | 100 | FN cost | 9 | | |
| | | Undersampling | 0.45 | | |
| | | # labels for ROI | 68 | | |

Table 9: $F_3$ scores achieved on the cross validation and holdout sets for the randomized train/test split experiment on the Yellowstone Lake dataset. Additionally, the percent of regions that were discarded in the testing data by each classifier are reported. The best $F_3$ scores are marked in bold.

| | SVM | LDA | Neural Network | Decision Tree | RUSBoost |
|---|---|---|---|---|---|
| Cross validation $F_3$ | 0 | **0.6790** | 0.5988 | 0.5946 | 0.6316 |
| Holdout $F_3$ | 0 | 0 | 0.3125 | 0 | **0.5263** |
| % regions discarded | 100 | 75 | 76.67 | 96.67 | 66.67 |

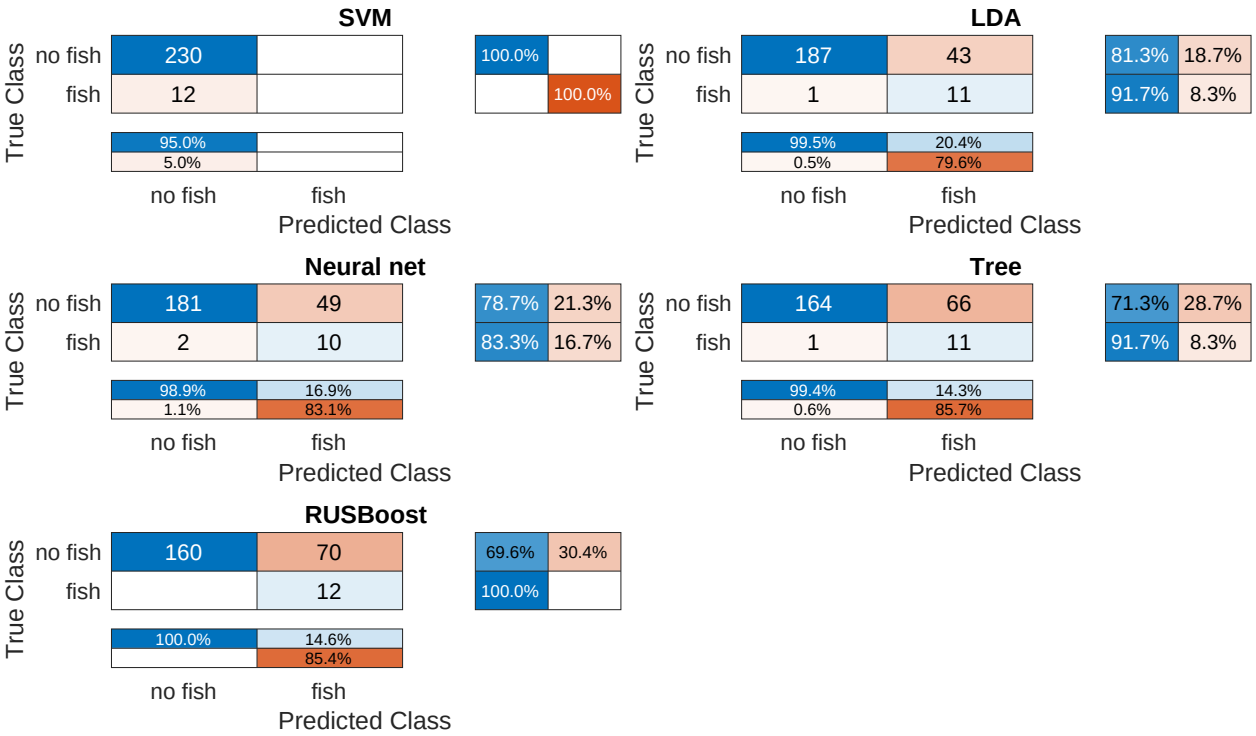



Figure 13: ROI cross validation results for the Yellowstone Lake randomized training set.

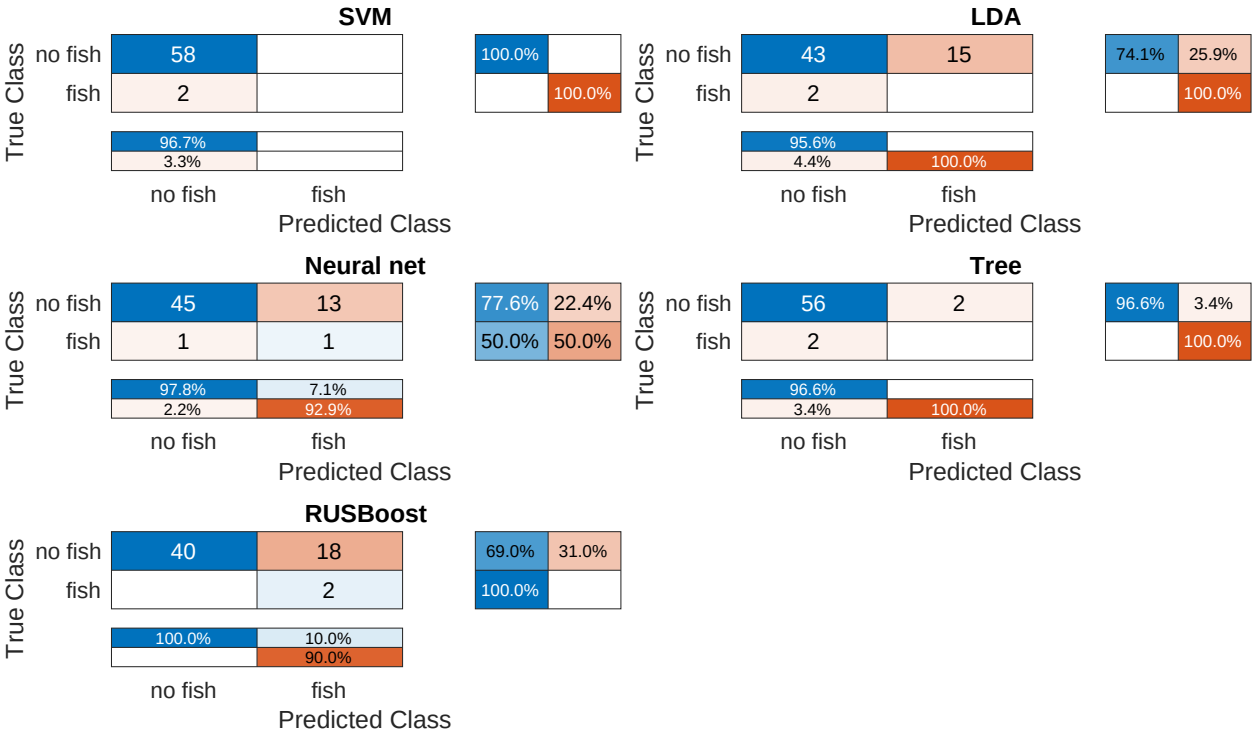



Figure 14: ROI holdout results for the Yellowstone Lake randomized testing set.

Gulf of Mexico Table 10 shows the hyperparameters obtained via tuning on the training set. The default LDA hyperparameters performed better than all the hyperparameters examined during tuning, so only the false negative cost was tuned. Figure 15 and Table 11 show that the decision tree achieved the highest $F_3$ score on the training set, while RUSBoost performed second best. As in the "first day" experiment on the Gulf of Mexico data, the decision tree's performance came at the cost marking more fishless regions as containing fish, which would increase the time required for manual inspection.

Table 10: Hyperparameter values for the randomized training set experiment on the Gulf of Mexico dataset. A $*$ indicates that the parameter values were left at their default values because hyperparameter tuning did not improve classification results.

| SVM | | LDA | | Neural Net | |
|---|---|---|---|---|---|
| Lambda | $3.8629 \times 10^{-9}$ | Delta | $0^*$ | Layer size | 44 |
| Regularization | lasso | Gamma | $0^*$ | Activation | tanh |
| FN cost | 20 | FN cost | 20 | Undersampling | 0.95 |
| Undersampling | 0 | Undersampling | 0.8 | # labels for ROI | 1 |
| # labels for ROI | 46 | # labels for ROI | 1 | | |
| **Decision Tree** | | **RUSBoost** | | | |
| Max number of splits | 4835 | Learning cycles | 401 | | |
| Min leaf size | 7 | Learning rate | 0.79965 | | |
| Split criterion | gdi | Max number of splits | 200 | | |
| FN cost | 20 | Min leaf size | 11 | | |
| Undersampling | 0.8 | Split criterion | gdi | | |
| # labels for ROI | 1 | FN cost | 6 | | |
| | | Undersampling | 0.8 | | |
| | | # labels for ROI | 1 | | |

Looking at the holdout results in Fig. 16 and Table 11, we see that the decision tree outperforms RUSBoost again, with all other classifiers obtaining significantly lower $F_3$ scores. In line with the cross validation results, the decision tree falsely classified 88.5% of fishless regions in the testing set as containing fish. RUSBoost misclassified 63.3% of fishless regions,

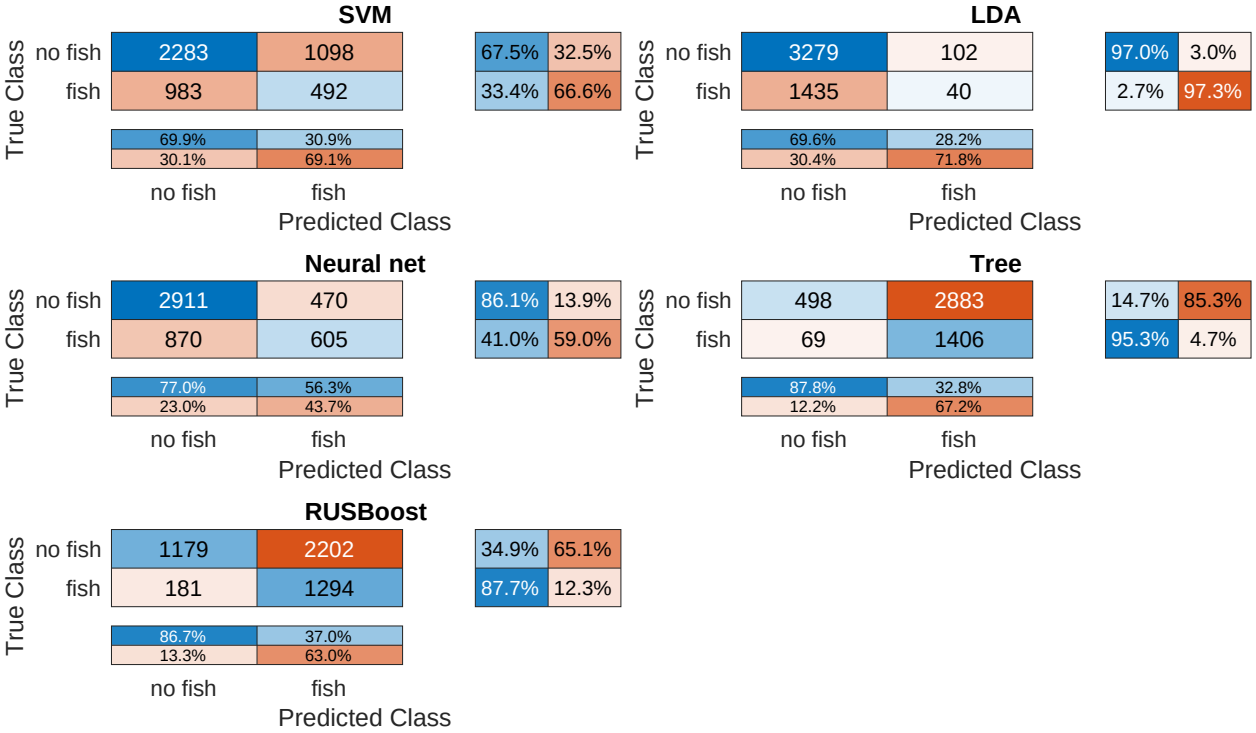



Figure 15: ROI cross validation results for the Gulf of Mexico randomized training set.

while the neural network only misclassified 8.5% of fishless regions. The neural network's low false positive rate is beneficial in reducing the amount of manual inspection required, but this came with the tradeoff of missing more than half of the fish-containing regions.

Table 11: $F_3$ scores achieved on the cross validation and holdout sets for the randomized train/test split experiment on the Gulf of Mexico dataset. Additionally, the percent of regions that were discarded in the testing data by each classifier are reported. The best $F_3$ scores are marked in bold.

| | SVM | LDA | Neural Network | Decision Tree | RUSBoost |
|---|---|---|---|---|---|
| Cross validation $F_3$ | 0.3310 | 0.0298 | 0.4216 | **0.8005** | 0.7716 |
| Holdout $F_3$ | 0 | 0.0210 | 0.3535 | **0.8166** | 0.7702 |
| % regions discarded | 100 | 98.02 | 83.84 | 8.57 | 29.43 |

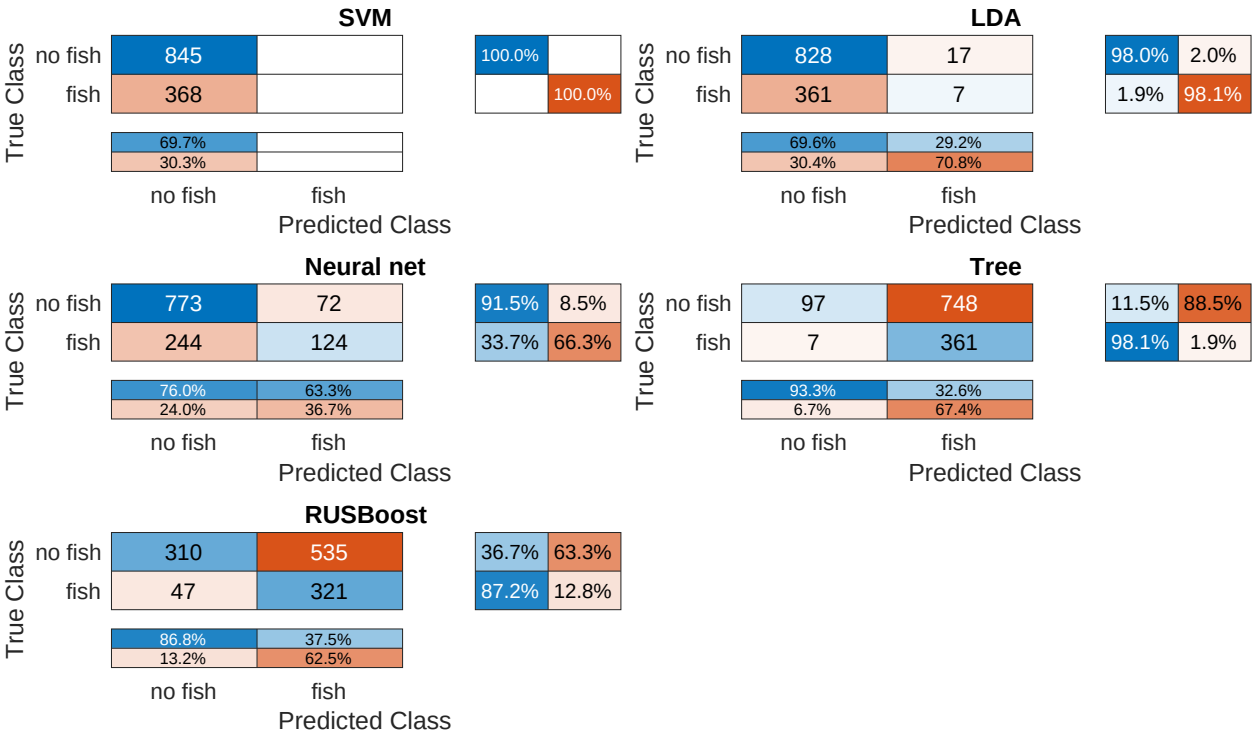



Figure 16: ROI holdout results for the Gulf of Mexico randomized testing set.

## Discussion

As seen in Sec. 2, no classifier performed best across all the experiments, and the classifier that performed best during cross validation often did not perform best on the testing set. This highlights the fact that classifiers are not guaranteed to perform well on new data; in the context of the lidar data analyzed in this paper, reduced performance on the testing data could be due to the fact that fish-containing regions often do not have the same characteristics or data distribution as one another. Additionally, the classifiers we tested traditionally assume that all data instances are independent; due to the nature of our data, this assumption is not entirely true: adjacent instances are spatially correlated. However, in spite of not always achieving similar performance on the testing data, several of the classifiers were able to identify the majority of fish-containing regions in the testing sets.

Although no classifier definitively came out on top, we can see several patterns and make

recommendations. On the Yellowstone data, RUSBoost and the neural network performed well across both experiments. While LDA and the decision tree also performed well, the fact that they missed both fish-containing regions in the randomized testing set could be a cause for concern if they were used in future campaigns at Yellowstone Lake.

On the Gulf of Mexico experiments, the decision tree and RUSBoost came out on top in both experiments. The decision tree tended to classify more regions as containing fish than RUSBoost did; on one hand, this increases recall, which is desirable, but it also increases the number of false positive regions that would need to be inspected. RUSBoost did not find as many fish, but it was able to discard more fishless regions. In a real-world Gulf of Mexico campaign, the choice between using RUSBoost or a decision tree would rely on the relative importance of finding all fish versus reducing manual labor. Furthermore, if one is most interested in reducing reducing manual labor, and can tolerate missing a significant number of fish-containing regions, the neural network would be a good choice because it consistently discarded over 60% of the training data in both experiments on both datasets.

Overall, based on the results seen in this study, we would recommend using either RUSBoost, a decision tree, or a neural network to detect fish regions in aerial lidar surveys. We can categorically not recommend using an SVM, as it was almost always uninformative; this is likely caused by the high class imbalance in fishery surveys. If reduced training time is important, one should consider using a decision tree; RUSBoost and neural networks had longer training times, as shown in Table 12. It is worth nothing that regardless of which classifier is used, the classifier needs to be trained on data from the specific survey being analyzed; in general, the results from classifiers trained on data from different locations or different instruments will not transfer to new locations or instruments.

Comparing the results between the “first day” training set (Sec. 2) and the larger, randomized training set (Sec. 2), we see that using the larger training set produced better results on the testing set, as expected. The cross validation results on the randomized

Table 12: Training runtimes in seconds. The runtimes are totals for the entire tuning procedure described in Sec. 2. The training was run on a Linux computer with an i9-9900k and 32 Gigabytes of RAM.

| | Yellowstone Lake | | Gulf of Mexico | |
|---|---|---|---|---|
| | "First day" training time | 80% split training time | "Fist day" training time | 80% split training time |
| SVM | 30.38 | 63.78 | 129.55 | 670.96 |
| LDA | 16.83 | 18.73 | 173.62 | 935.13 |
| Neural Net | 334.74 | 883.19 | 9377.30 | 46401.00 |
| Decision Tree | 22.64 | 45.99 | 578.54 | 4684.90 |
| RUSBoost | 177.80 | 235.19 | 2888.80 | 8074.30 |

training set were lower than those on the "first day" training set; this makes sense because the randomized training sets were more diverse, resulting in harder-to-learn data distributions. Nonetheless, the results show that our methods can be useful in both real-world scenarios simulated in Sec. 2 and Sec. 2.

Figure 17 shows a representative example of the predicted labels for a region of interest. The predicted labels found one fish, missed one fish, and predicted several false fish. This behavior should prompt a human to take a closer look, which is why it was labelled as a region of interest.

As an example of the time savings that our method can achieve, consider RUSBoost's holdout results on days 2–12 in Fig. 16. RUSBoost predicted 1386 (26.8%) regions as not containing fish. 1386 regions is approximately 13.5 hours of lidar data. With an average analysis time of 15 minutes for an hour of lidar data, 1386 regions would take approximately 4.6 hours to analyze. Inspecting the Gulf of Mexico data took about 14 hours total; saving 4.6 hours of manual labor is approximately a 33% time savings.

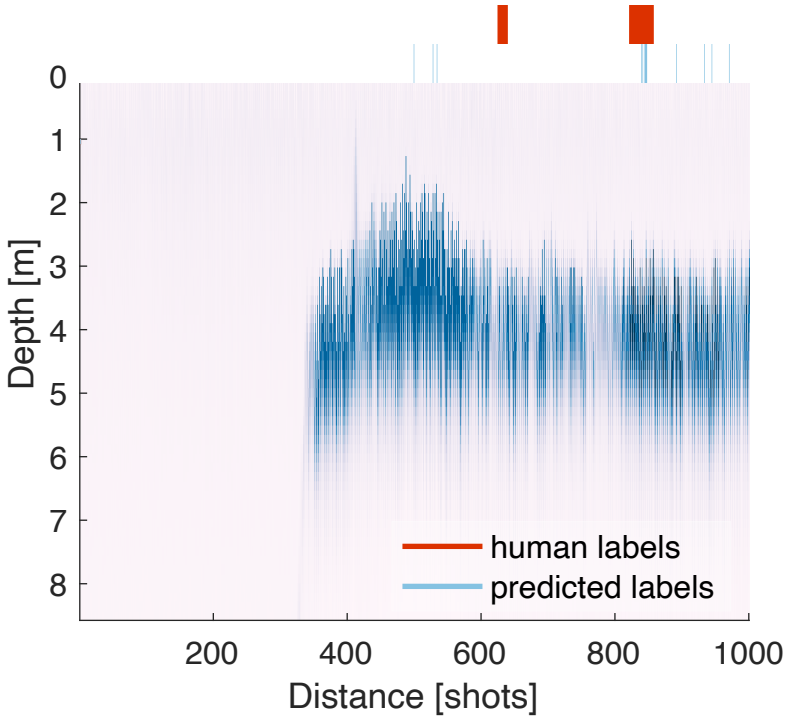



Figure 17: An example ROI from the 2016 Yellowstone flight.

## Conclusion

In this study, we demonstrated the feasibility of using supervised machine learning techniques to detect regions that have a high probability of containing fish. We devised two experiments to simulate real-world scenarios: one where a classifier is trained after the first day of a campaign and used during the subsequent days, and one where a classifier is trained on a previous campaign in preparation for a new campaign at the same location. In both cases, we found that several classifiers were able to correctly detect the majority of fish-containing regions while reducing the amount of data that would require manual inspection. This is a significant improvement over manual inspection, which is the current state-of-the-art in the field. Additionally, we made recommendations on which classifiers should be used for future lidar-based fishery surveys.

Future work

Given the plethora of machine learning techniques available, there are many avenues for future work. One of the primary directions will be to extract features from the data, as feature engineering might improve results compared to using the raw data. Using classifiers, such as recurrent neural networks, that model the time-dependency between samples is also a promising direction for further research. Another interesting area of future work is to aggregate multiple classifiers into an ensemble. For example, aggregating the outputs of the RUSBoost, decision tree, and neural network classifiers used in this study could result in improved performance.

In addition to further exploration of supervised learning techniques, we will direct future research efforts towards real-time and unsupervised fish detection methods. Real-time methods would allow researchers to detect fish during flight, enabling personnel on the ground to carry out further operations, e.g., collecting more data or removing invasive species. Unsupervised methods would eliminate the need for ground truth labels, resulting in further time savings. Although there is still room for improvement, our results motivate further application of machine learning techniques in fishery surveys and other lidar-based remote sensing applications.

## Acknowledgment

We gratefully acknowledge funding from the U.S. Air Force Research Laboratory through subcontract with S2 Corp. Additionally, we thank the reviewers and editors for their many insightful comments and suggestions that have greatly improved the quality of this manuscript.

## Code, Data, and Materials Availability

The authors are coordinating with data librarians at Montana State University to archive the Yellowstone and Gulf of Mexico fish lidar datasets. The machine learning software as been published on GitHub and archived via Zenodo and assigned a Digital Object Identifier (DOI). The software can be reached at https://doi.org/10.5281/zenodo.5097031. The Zenodo archive will be updated to include a link to the raw data when it becomes publicly available. Until then, the data is available upon request.

# COMPARISON OF SUPERVISED LEARNING AND CHANGEPOINT DETECTION FOR INSECT DETECTION IN LIDAR DATA

## Contribution of Authors and Co-Authors

Manuscript in Chapter 3

Author: Trevor C. Vannoy

Contributions: conceptualization, data curation, investigation, formal analysis, methodology, software, validation, visualization, supervision, writing—original draft, writing—review & editing

Co-Author: Nathaniel B. Sweeney

Contributions: data curation, investigation, formal analysis, methodology, software, validation, visualization, writing—original draft, writing—review & editing

Co-Author: Joseph A. Shaw

Contributions: conceptualization, funding acquisition, investigation, methodology, resources, visualization, supervision, writing—original draft, writing—review & editing

Co-Author: Bradley M. Whitaker

Contributions: conceptualization, funding acquisition, methodology, project administration, resources, supervision, writing—original draft, writing—review & editing

Manuscript Information

Trevor C. Vannoy, Nathaniel B. Sweeney, Joseph A. Shaw, Bradley M. Whitaker

Remote Sensing

Status of Manuscript:
____ Prepared for submission to a peer-reviewed journal
____ Officially submitted to a peer-reviewed journal
____ Accepted by a peer-reviewed journal
_X_ Published in a peer-reviewed journal

MDPI
Volume 15, Issue 24
https://doi.org/10.3390/rs15245634
Copyright: © 2023 by the authors. Licensee MDPI, Basel, Switzerland. This article is an open access article distributed under the terms and conditions of the Creative Commons Attribution (CC BY) license (https://creativecommons.org/licenses/by/4.0/).

## Abstract

Concerns about decreases in insect population and biodiversity, in addition to the need for monitoring insects in agriculture and disease control, have led to an increased need for automated, non-invasive monitoring techniques. To this end, entomological lidar systems have been developed and successfully used for detecting and classifying insects. However, the data produced by these lidar systems create several problems from a data analysis standpoint: the data can contain millions of observations, very few observations contain insects, and the background environment is non-stationary. This study compares the insect-detection performance of various supervised machine learning and unsupervised changepoint detection algorithms and provides commentary on the relative strengths of each method. We found that the supervised methods generally perform better than the changepoint detection methods, at the cost of needing labeled data. The supervised learning method with the highest Matthew's Correlation Coefficient score on the testing set correctly identified 99.5% of the insect-containing images and 83.7% of the non-insect images; similarly, the best changepoint detection method correctly identified 83.2% of the insect-containing images and 84.2% of the non-insect images. Our results show that both types of methods can reduce the need for manual data analysis.


## Introduction

In spite of their small size, insects play a huge role in ecosystems around the world—outnumbering all other terrestrial species on Earth [89, 90]. Reasons for detecting and monitoring insects are numerous: controlling malaria [91], increasing agricultural yields [92], investigating population decline [93], understanding changes in biodiversity [94, 95], etc. Traditional monitoring methods, such as traps and netting [96], are not capable of the wide-scale monitoring needed to understand and address global insect population decline [97]. These

traditional techniques usually kill the insects, lack precise spatial and temporal resolution, and are labor-intensive. The downsides of traditional techniques, along with the rising importance of wide-scale monitoring, have led to numerous calls for improved monitoring and data analysis methods [39, 95, 98, 99], many of which rely on remote sensing and machine learning.

Automated monitoring techniques aim to increase the ease and scale of insect monitoring; they are typically based on audio [100], radar [101–103], or optical sensors, such as lidar [102, 104–108] or camera traps [109–111]. Given the large amounts of data these automated monitoring techniques generate, machine learning has been applied to help automate data analysis tasks. For example, computer vision has been used in many camera trap studies [99, 112–116]. In lidar studies, machine learning has been used for numerous tasks, such as identifying gravid mosquitoes [117], estimating biodiversity [106], and classifying species [118]. Since all the aforementioned remote sensing techniques have unique advantages and disadvantages, no single sensing modality will be sufficient to solve all insect monitoring problems; moreover, these remote sensing techniques need to be combined with traditional in-situ data collection to fully address ecological challenges [95].

The study presented here focuses on a lidar-based technique, which can be used for high-resolution spatio-temporal monitoring of insects [105]. The lidar instrument was developed by the Optical Remote Sensor Laboratory at Montana State University (MSU) [105]. This paper compares supervised machine learning and unsupervised changepoint detection techniques for detecting insects in pulsed lidar data.

Early entomological lidar development started with groups at Sandia National Laboratories [119] and MSU [40, 120, 121] demonstrating that lidar can detect honeybees; Shaw et al. [120] located unexploded landmines by mapping the locations of honeybees that were trained to locate the landmines. In the following two decades, several entomological lidar systems have been developed using time-of-flight [105], Scheimpflug [107, 122], and fluorescence [123, 124] techniques. For an overview, see the review papers by Brydegaard et

al. [104] and Wang et al. [108]. However, these entomological lidar systems produce a large amount of data, making the data analysis time-consuming and labor-intensive. Beyond this, the measurements contain few insects, and the background environment can vary over time and space.

Due to the variable background in lidar measurements, as well as the fact that most measurements contain no insects, automatically detecting insects is challenging. Previous field studies by Brydegaard et al. [107, 125, 126] detected insect events using variations of the methods described by Malmqvist et al. [127]. This method determines the static background signal via measurements when the laser is off and subtracts this background from the signal measurements. Any samples with an amplitude above a range-dependent threshold are considered as insect events. This method has been successful at finding insect events in several studies [106, 107, 125, 126, 128]; however, other non-insect objects that pass through the beam are likely to be above the detection threshold. Rather than relying on a statistical threshold, Rydhmer et al. trained a convolutional neural network to distinguish insect observations from non-insect observations [129]; this technique is most similar to the learning methods presented in this paper.

Most previous entomological lidar studies that used machine learning classification focused exclusively on feature engineering and supervised learning methods [117, 118, 130, 131]. However, since the insects are being remotely sensed, ground truth labels are difficult to generate or infeasible to collect. For example, ground-truth species labels in field studies are typically unknown because researchers cannot collect the remotely sensed insects; even if insects were collected during a field experiment, correlating individual insects to individual measurements would be challenging. Consequently, some work has explored unsupervised methods, such as clustering of power spectra [106, 125, 129]; changepoint detection is another unsupervised method, which our research group has previously explored [132]. To the best of our knowledge, only two studies have used end-to-end machine learning [118, 129]. It is worth

noting that each entomological lidar instrument has been developed as a unique research instrument, making direct comparisons of data and results difficult.

Although previous entomological lidar studies used a variety of machine learning algorithms, most studies only used one algorithm or compared a small set of algorithms. The main aim of this paper is to compare the insect-detection performance of a wide variety of machine learning techniques. In particular, we test traditional supervised learning, deep learning, and changepoint detection, as each method category has unique advantages and disadvantages. To that end, we present performance comparisons of six supervised and two changepoint methods. For supervised methods, we present four feature engineering and two deep learning [133] approaches. We evaluate and compare these methods on a new entomological lidar dataset, which we openly publish [134].

## Materials and Methods

### Lidar System

We collected data using the wingbeat-modulation lidar system developed by MSU's Optical Remote Sensor Laboratory [105]. Figure 18 shows the intuition behind detecting insects using wingbeat-modulation lidar. As an insect flaps its wings, the amount of reflected light changes, which in turn leads to modulations in the lidar data. By analyzing the changes in reflected light in the frequency domain, one can determine the insect's wingbeat spectrum [104, 105]. The unique signal characteristics of the insect's wingbeats can also be used to distinguish insects from other objects. Figure 18b shows an example image containing one stationary object and two insects: one where reflections from the insect's wings dominate the signal, and one where reflections from the insect's body dominate the signal. Figure 18c shows time series plots of the three targets. The stationary object looks like noise. The reflection from the insect's body creates a noticeable dip in the signal, while the wing's oscillations are superimposed on top of the body reflection. The wing signal in the lower plot

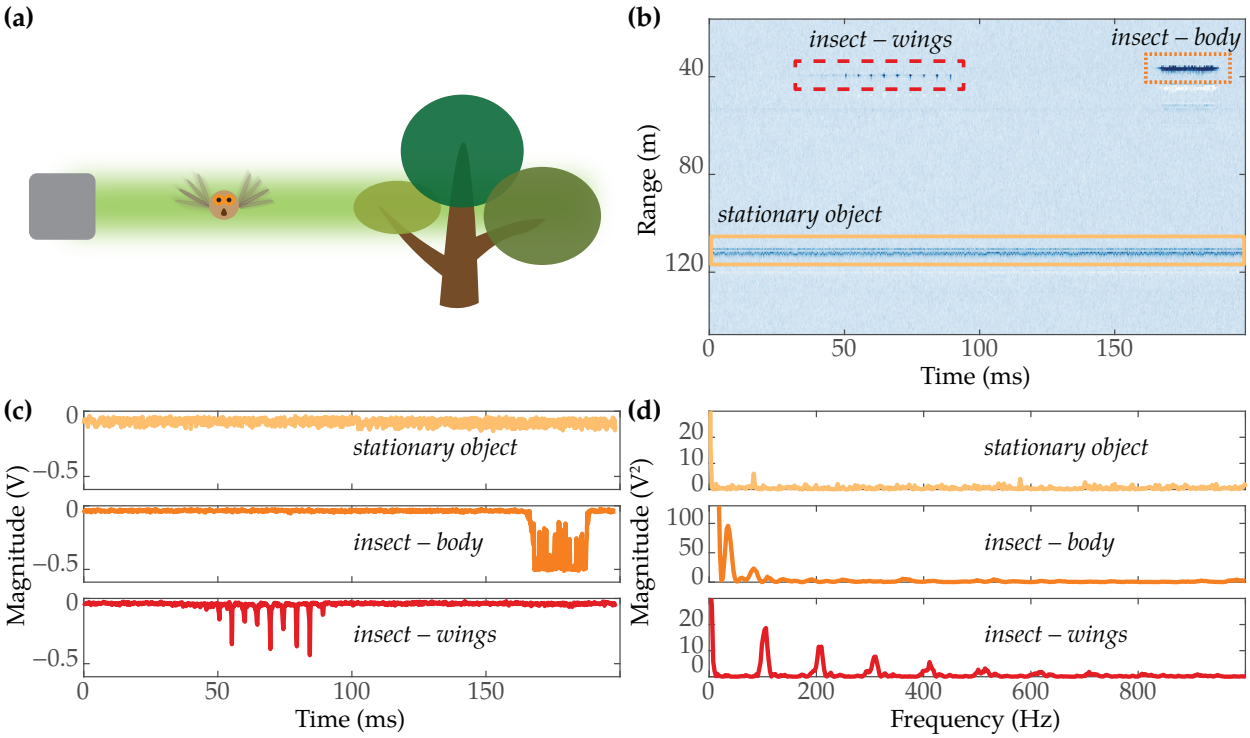



Figure 18: (**a**) When an insect flies through the lidar beam and flaps its wings, the amount of reflected light changes. These changes in received intensity can be distinguished from other stationary objects. (**b**) An image showing a stationary object (yellow rectangle) and two insects: one where only the wings were in the lidar beam (dashed red rectangle), and one where the body of the insect was in the beam (dotted orange rectangle). The white-colored returns beneath the insects are reflections from the lidar's wiring. (**c**) Time series plots showing the voltage magnitude of the three targets in (**b**). (**d**) Frequency spectra for each of the targets.

shows distinct periodic peaks. Figure 18d shows the frequency spectra for each of the targets. The stationary object looks like noise in the frequency domain; the insect's body produced a dominant low-frequency component, while the wing signal produced obvious harmonic structure.

Figure 19 shows an optical schematic of the lidar system, and Figure 20 shows a picture of the lidar system during the experiments. The first component in the transmitting subsystem was the JDS Uniphase NG-10320-100 532 nm laser, which had a pulse width of 0.6 ns, a pulse energy of 3 µJ, and a pulse repetition frequency (PRF) range between approximately 2.9 kHz

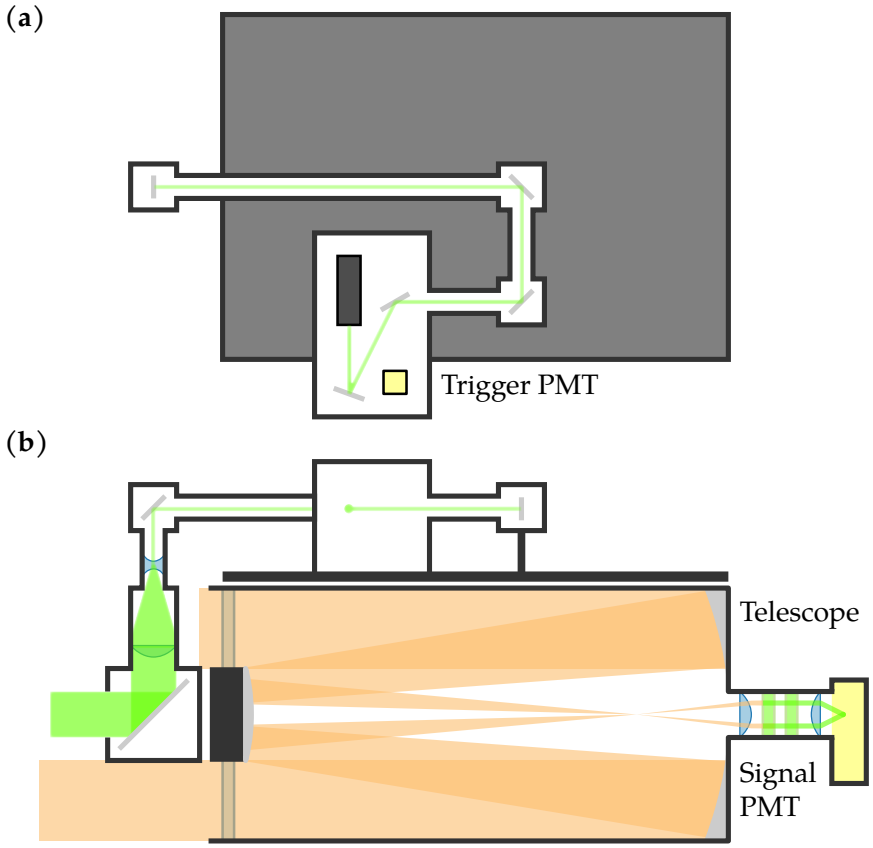



Figure 19: Optical schematic of the lidar system. (**a**) Top view of the transmitting optics. (**b**) Side view of the transmitting and receiving optics. The PMTs are shown in yellow.

to 5.4 kHz. The laser was housed in a light-tight box along with a Hamamatsu H6780-20 photomultiplier tube (PMT), the output of which was used as the trigger input for the analog-to-digital (ADC) converter. A series of mirrors and lens tubes directed the light out of the system at the center of the receiver telescope's central obstruction. Before exiting the system, the beam was expanded to have a 50.8 mm diameter and half-angle divergence of 0.2 mrad.

The first component in the receiving subsystem was the Meade LX200 Classic Schmidt-Cassegrain telescope with a 305 mm diameter and a focal length of 3048 mm. At the back of the telescope, a collimating lens, two laser line filters, and a focusing lens were housed in a lens tube. A Hamamatsu H9305-04 PMT was attached to the end of the lens tube. The lidar




Figure 20: Picture of the lidar system during data collection.

system was mounted on a MOOG QPT-130 pan-tilt mount.

The data were digitized using a Compuscope 14200 PCIe ADC with a 200 MS/s sampling rate, 100 MHz bandwidth, and 14-bit resolution. The 200 MS/s sampling rate limited the system's range resolution to 0.75 m. The output of the H9305-04 PMT was connected to the ADC's input channel, and the output of the H6780-20 PMT was connected to the ADC's trigger input. For each laser pulse, the ADC was triggered to collect 200 samples. Due to the transit time for the light to leave the system, as well as variance in when the ADC started sampling, 178 samples were usable [135]. MATLAB was used to control the ADC and collect 1024 pulses, resulting in images of 178 rows (range bins) $\times$ 1024 columns (pulses). For compatibility with the ADC, MATLAB 2015b was used. Figure 21 shows a generic example of how the data were arranged into a matrix; the data are later visualized as false-color images. Since the laser's PRF was not constant, the total duration of 1024 pulses was not constant; this resulted in images of varying durations.

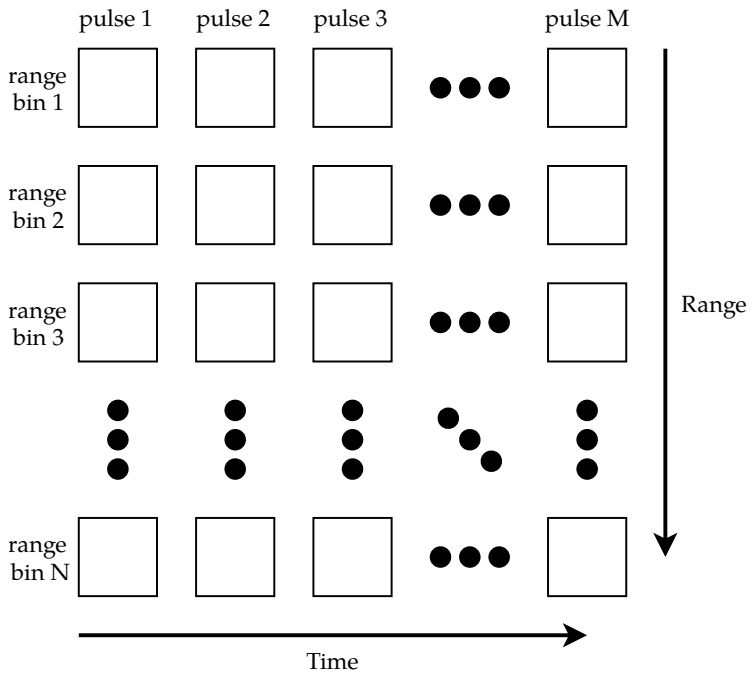



Figure 21: A generic example of how the data were stored. Each column in the matrix represents one pulse of the lidar, and each row represents one range bin. Each matrix element is one sample.

Dataset

The data used for the experiments were collected in June and July 2022 at the MSU horticulture farm. The data comprise 9977 images taken in front of the beehives. As noted previously, the image durations vary because the laser's PRF varied; this can be seen in Figure 23.

For the data collection process, the lidar was mounted in the back of a van, as shown in Figure 20, and pointed in front of the beehives, some of which are shown in Figure 22. The data were collected with the lidar beam pointed in various discrete directions; at each position, multiple images were collected while the lidar's position was fixed. This was done to create a diverse set of measurements to train the machine learning algorithms on; having various levels of bee activity and background clutter helps avoid overfitting. The position of the lidar beam was controlled by changing the pan and tilt angles of the pan-tilt mount

that the lidar was mounted on. The probability of bees flying through the lidar's beam was a function of the beam's position, as well as the various factors that influence bee activity (temperature, time of day, etc.).



Figure 22: A picture of bee activity around some of the beehives. Four out of twelve beehives are shown. Some foliage was present near the beehive entrances, which sometimes shows up as background clutter in the collected images.

After collecting the data, we manually labeled the bounding box of each insect in each image, then converted the bounding boxes into binary labels that indicate whether a row contains an insect. Each potential insect was labeled with a confidence rating between 1 and 4 because some bees were more obvious than others. In general, high confidence ratings were assigned to bee events, which have obvious harmonic structure and a prominent return amplitude in the time domain. Low confidence ratings were given to events with very short transit times, such as those shown in Figure 23c. During the labeling

process, we found 4671 probable bees. All labeled bees, regardless of confidence rating, were included in the dataset; our insect-detection algorithms did not account for confidence ratings. Since we were not able to collect ground-truth data in the field, it is possible that our labels are imperfect: some insects might have been missed, and some non-insects might have been labeled as insects. Our ground truth labels are the primary source of uncertainty. Using imperfect labels during training and validation may result in false insect detections; on the contrary, classifiers may find insects that the ground truth labels missed. The Sources of Uncertainty and Error and Model Generalization sections provide additional discussion on the effect of imperfect labels.

Of the 9977 images, 3498 (35.14%) contain one or more bees. In total, the dataset has 1,775,906 rows, 11,492 (0.647%) of which contain an insect measurement. Due to sampling jitter in the ADC, most insects span multiple range bins, leading to an increase in the number of rows that were labeled as containing insects. The dataset has a large class imbalance, particularly when looking at how many rows contain insects. Large class imbalances can cause machine learning algorithms to ignore the minority class [136], which would result in not detecting most insects. The Data Sampling section discusses the methods we used to alleviate class imbalance.

Figure 23 shows four example images from the dataset containing different types of bee activity and stationary objects. The bee signals vary depending on whether the laser primarily hit the body or the wings of the bee. There is a low chance that the lidar pulses only hit the body, but the return from the body often dominates the signal when the incidence angle is such that the more specular wing reflection is not large. There are also images containing small blips of a bee where the laser might have barely hit the bee for a few pulses. These blips were labeled as bees but with a low confidence. When the intensity of returned light was large, reflections showed up in the data, which resulted in positive voltages at later range bins, as indicated by the orange dashed boxes in Figure 23. These artifacts are undesirable

because they have similar characteristics to the actual returns that caused them. After the data were collected, we discovered that the reflections were caused by an impedance mismatch in the lidar's wiring. By the time we discovered this, we could not recollect the data. The Preprocessing section discusses our method to reduce the reflection artifacts.

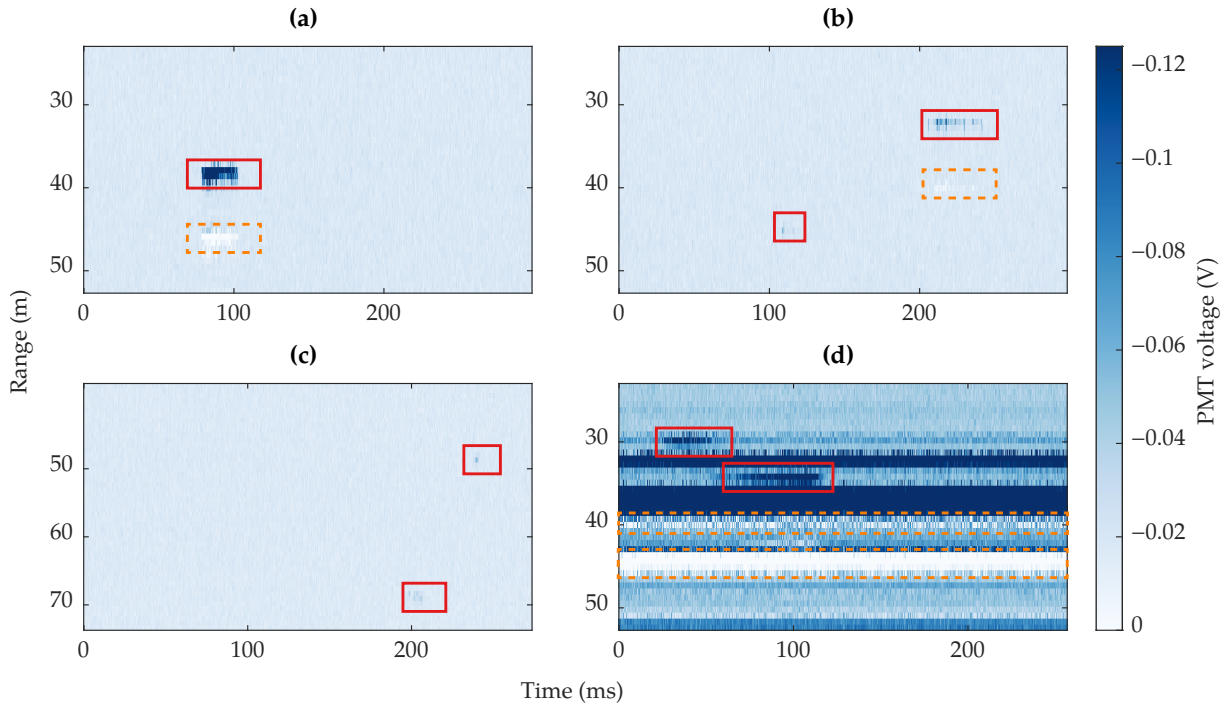



Figure 23: Example images of different types of bees and images found in the dataset. For visualization clarity, the range of the images are cropped, and the colormap range is limited. The bees are surrounded by red rectangles; reflections are surrounded by orange dashed rectangles. (**a**) An example where the bee's body dominated the return signal. (**b**) An example where the wing return dominated the signal (upper right rectangle), and an example of a very short-transit event (lower left rectangle). (**c**) Two examples of weak, short-transit observations. (**d**) An example of two long-transit signals where both the body and wing signals are present; the image also contains stationary, non-bee targets; the stationary objects have associated reflections, indicated by the orange dashed rectangles The duration of the four images are not the same due to the laser's varying PRF.

Figure 24a shows a histogram of the insect transit times observed in the dataset. The shortest transit time was 744 µs, the longest transit time was 307.9 ms, and the average transit time was 36.7 ms. Most bees were in the beam for less than 50 ms, which is about 1/5 of the average image duration of 230 ms. Note that the image durations are not all the same. None

of the recorded transit times can be longer than the associated image's duration, though the object may have been in the beam for longer than the image's duration. Nine events that were labeled as insects had transit times that lasted the entire image duration; most of these were labeled as low-confidence insects.

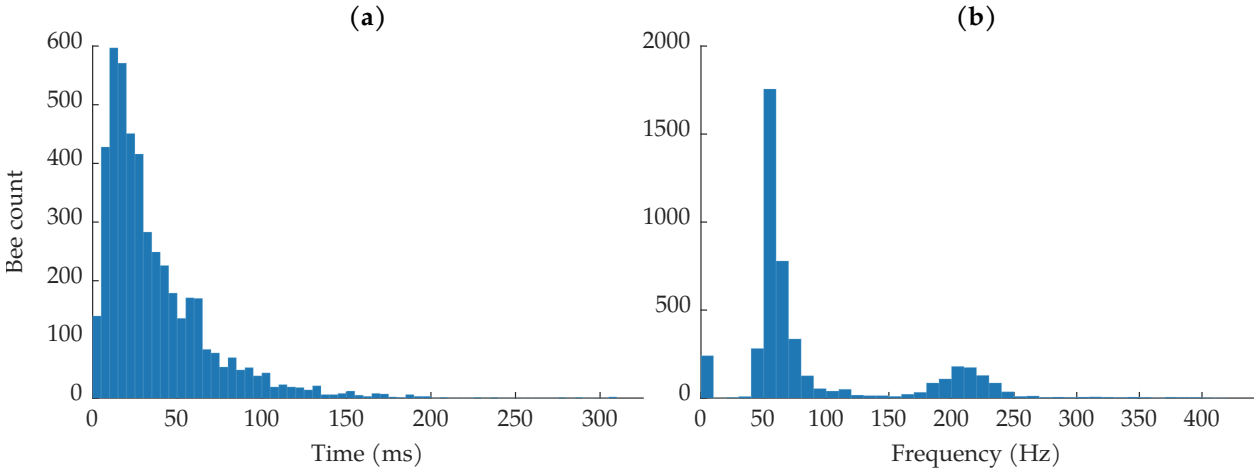



Figure 24: (**a**) Histogram of bee transmit times. (**b**) Histogram of estimated fundamental wingbeat frequencies.

Figure 24b shows the distribution of fundamental wingbeat frequencies. Previous literature has found that honeybees' fundamental wingbeat frequency can range from 150 Hz to 600 Hz and varies with environmental conditions [137]. The most common wingbeat frequencies are in the low 200 Hz region [138, 139]. The fundamental frequency of each insect observation was calculated using the harmonic product spectrum algorithm [140]. Since most insects showed up in multiple range bins due to sampling jitter, the fundamental frequency was estimated on the average of the insect's range bins to increase the quality of the insect signals. Before estimating the fundamental frequency, the insect observations were bandpass filtered to help isolate the expected wingbeat frequencies. We used a 10th-order Butterworth filter with a passband between 50 Hz and 1200 Hz; the upper passband frequency was chosen to ensure that the third harmonics in the spectra were left untouched, so as to not affect the harmonic product spectrum algorithm. The wingbeat frequency distribution shown in Figure

24b is distinctly bi-modal. In agreement with previous literature, we see a strong distribution of wingbeat frequencies between approximately 175 Hz and 250 Hz [137–139]. The other distribution of wingbeat frequencies, between 0 Hz and approximately 125 Hz, came from the bee observations where the body dominated the return signal and very little wingbeat motion was captured.

The data from June were used for training and validation, and the data from July were used for testing. This split simulates the scenario of training algorithms after one field campaign and then using the algorithms during the next campaign. The June data were split into stratified training and validation sets, with 80% for training and 20% for holdout validation. Table 13 shows the number of images, observations, and bees in each set.

Table 13: Number of images, insect-containing images, rows, and insect-containing rows in the training, validation, and testing sets. The percentage of insect-containing images and insect-containing rows are shown in parentheses.

| **Dataset** | **# of Images** | **# of Insect Images** | **# of Rows** | **# of Insect Rows** |
|---|---|---|---|---|
| Training | 3295 | 1048 (31.81%) | 586,510 | 3410 (0.581%) |
| Validation | 823 | 261 (31.71%) | 146,494 | 840 (0.573%) |
| Testing | 5859 | 2189 (37.36%) | 1,042,902 | 7242 (0.694%) |

Preprocessing

As mentioned in the Dataset section, an impedance mismatch caused reflections in the lidar's wiring, which resulted in undesirable artifacts in the data. These artifacts can easily confuse the bee detection algorithms, as they share the same characteristics as the bee that produced them. Since the PMT output current is negative, any positive voltage values were due to reflections. Since the samples in each row form a time-series at a particular range bin, any large changes between subsequent samples would be caused by a moving object at that range bin. Using this knowledge, we thresholded all positive voltage values to the

corresponding row's average value. Figure 25 shows an example image before and after thresholding. The thresholding reduced most of the reflections, but not all, because some of the reflections resulted in small negative voltages near 0. Since we cannot easily distinguish between a small negative voltage caused by a reflection and a small negative voltage caused by a weak return from a bee or other object, we settle for leaving some artifacts in the data. While leaving these artifacts will increase the false positive rate of the algorithms, a more aggressive removal would result in missed insect detections.

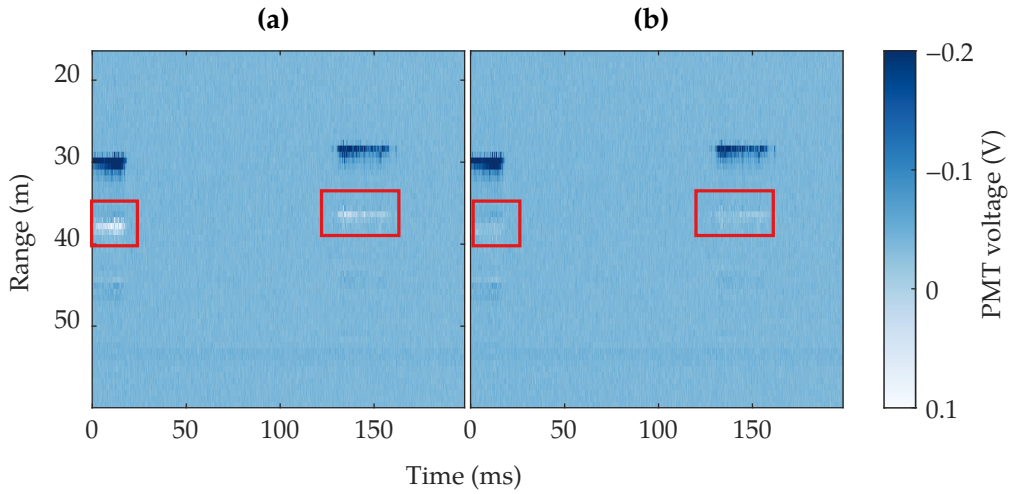



Figure 25: Example of an image before (**a**) and after (**b**) thresholding the PMT ringing to the mean of the corresponding row. The ringing is highlighted by red rectangles. For visualization clarity, the range and colormap are limited.

Supervised Methods

The supervised learning methods we used can be broken into two categories: feature engineering and deep learning. The feature engineering methods use the features described in Feature Engineering Methods section as input, whereas the deep learning methods take the raw data as input. The methods can also be classified by whether they used rows as input or entire images as input. The feature engineering methods, as well as the 1-dimensional deep learning methods, predict which rows contain insects; the 2-dimensional deep learning methods take entire images as input and predict which images contain insects. For the

row-based methods, the row predictions were aggregated to predict which images contain insects.

The overall supervised learning process comprised tuning dataset sampling parameters, tuning model hyperparameters, training, and testing. While our code attempts to make training as reproducible as possible by carefully seeding MATLAB's random number generator, some algorithms were still not deterministic; consequently, we ran the whole process three times for each supervised learning algorithm.

Data Sampling As mentioned previously, our dataset is highly imbalanced. To deal with this, we used combinations of data augmentation and random undersampling [136] while training the row-based methods. Data sampling was not performed for the image-based methods because the image-based dataset is not highly imbalanced.

Random undersampling was used to remove rows from each image that did not contain insects. To increase the number of insect examples used during training, we created synthetic insects. Our augmentation procedure performs 6 transformations: (1) random circular shifting; (2) interpolating the circularly-shifted vector; (3) decimating the circularly-shifted vector; (4) vector reversal; (5) interpolating the reversed vector; (6) and decimating the reversed vector. Random Gaussian noise is added to each synthetic observation. These transformations are performed in order. Consequently, when the number of synthetic observations, $N$, is not a multiple of 6, the last $n = N \pmod 6$ synthetic observations are created using the first $n$ transformations from the list; for example, when the number of synthetic insects is 10, the last 4 synthetic insects are created using the first 4 transformations.

For the row-based methods, we performed a grid search over the following undersampling and data augmentation parameters:

- undersampling ratio: $\{0, 0.25, 0.5, 0.75\}$
- # of synthetic insects per insect observation: $\{0, 1, 10, 100\}$

During the grid search, the Matthew's Correlation Coefficient (MCC) [14] was used to evaluate classification performance on the validation dataset. For each model, the sampling parameters with the highest MCC value were chosen as the final sampling parameters. During the grid search, the models were trained with their default hyperparameters.

Feature Engineering Methods For the feature engineering methods, we trained four different types of models: AdaBoost [32], RUSBoost [141], a linear support vector machine (SVM), and multilayer perceptrons (MLP). AdaBoost and RUSBoost are both boosted decision tree ensembles, while the MLP is a feedforward neural network. We trained MLPs with 1, 3, 5, and 7 hidden layers.

The inputs to these models were 46 hand-selected features, which were extracted from each row in each image. From the time domain, we extracted the following features from each row: range, standard deviation, maximum absolute first difference, maximum absolute second difference, root-mean-square (RMS), impulse factor, crest factor, shape factor, median, median absolute deviation, mean-centered skewness, mean-centered kurtosis, and the observation mean minus the image mean. From the frequency domain, we extracted various statistics from the normalized one-sided energy spectral density (ESD): mean, median, median absolute deviation, standard deviation, mean-centered skewness, and mean-centered kurtosis. Most frequency-domain features were based upon the insect's wingbeat frequency and harmonics, such as the height and location of each harmonic, as well as ratios of each harmonic's features. The fundamental frequency was estimated from the normalized ESD using the harmonic product spectrum algorithm [140]. Before extracting any of the frequency domain features, the signals were bandpass filtered with a 10th order Butterworth filter with a passband between 50 Hz and 1200 Hz.

In addition to time- and frequency-domain features, we used generalized Morse wavelets [142] to compute the continuous wavelet scalogram, from which we extracted

additional features. Across the time axis in the time-frequency plane, we computed the average intensity for each frequency bin, then selected the maximum of those intensity values. Similarly, we computed the standard deviation of each frequency bin and extracted the mean of those values. We also computed the maximum and mean values of the scalogram, the maximum change between adjacent values in the frequency axis, and the average skewness of the frequency bins. For a full list of features, see the feature ranking in Figure 26 and our code [143].

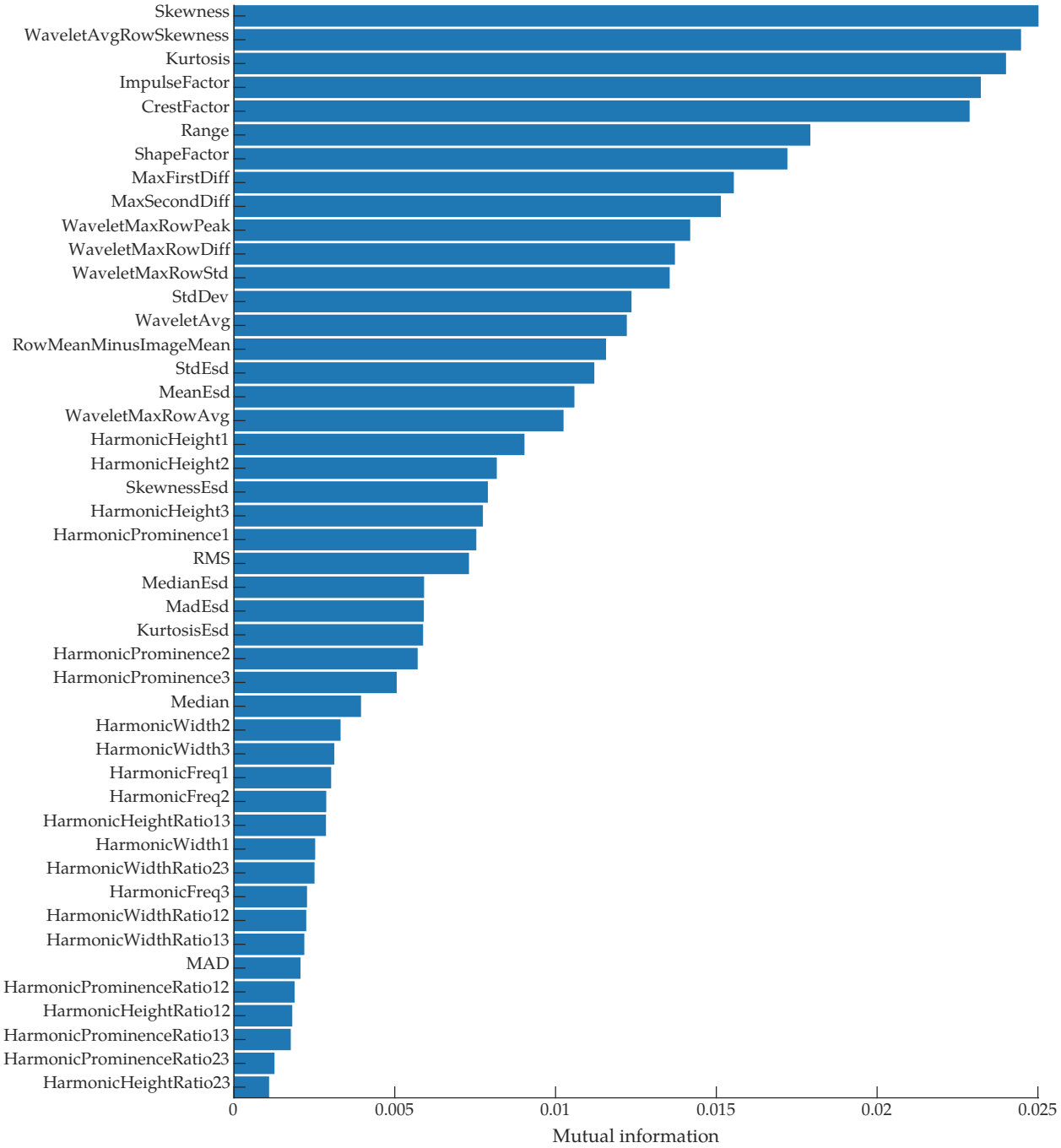



Figure 26: A list of all features ranked by their mutual information with the class labels.

Individual model hyperparameters and the cost of predicting false negatives were tuned using Bayesian optimization [144], using MATLAB's `bayesopt` function. Fifteen iterations were used during parameter tuning. MCC was again used to select the final hyperparameters.

The search range for the false negative cost was integers between 1 and 10. For brevity, view our source code [143] for more information on the model-specific hyperparameter search ranges. The data sampling and model hyperparameter tuning were performed sequentially, rather than together. Our assumption was that the optimal data sampling parameters were independent of the individual model hyperparameters; although this assumption is likely not 100% true, performing the two tuning steps sequentially drastically reduces the parameter search space.

Deep Learning Methods Given the rise of deep learning, we wanted to see if deep learning outperformed our manually engineered features. To this end, we tested two deep learning algorithms: a 2D convolutional neural network (CNN) that took the raw images as input, and a 1D CNN that operated on each row. For both CNNs, we tested models with 1, 3, 5, and 7 convolutional layers. Each convolution layer was followed by a batch normalization layer, a layer of rectified linear unit (ReLU) activation functions, and a dropout layer with a 20% dropout probability to reduce the chance of overfitting. Between the ReLU layer and the dropout layer, the 2D CNNs included a 2-by-2 max pooling layer with a stride of 2. Each model used weighted cross-entropy loss for the output layer. The default filter size for the 1D CNNs was 16, and the default filter size for the 2D CNNs was $16 \times 2$. A rectangular filter size was chosen as the default for the 2D CNNs because insect events are typically wider than they are tall.

All networks were trained using the Adam optimizer [145], though no efforts were made to optimize the training parameters; 20 training epochs and a learning rate of 0.01 were used for all models; the mini-batch size was 2048 and 64 for the 1D and 2D CNNs, respectively. The filter size, number of filters, and false negative cost for each model were tuned using the same procedure described in the previous section. For more architecture and training details, see our source code [143].

Changepoint Detection

The motivation for using changepoint algorithms was that bees show up as abrupt changes in both rows and columns. The experiments were conducted using MATLAB's `findchangepts` function, a primarily statistical changepoint detection algorithm, and gfpop [146, 147], a graph-based changepoint detection algorithm.

The `findchangepts` function is a part of MATLAB's Signal Processing Toolbox and returns the index where the input signal changes most significantly. This algorithm utilizes a parametric global method of dividing the signal into two sections from some division point, calculating an estimate of the desired statistical property, measuring the deviation of the property at each point from the estimate, calculating total residual error, and then varying the division location until the total residual error is at its minimum. In this study, the mean was used as the statistical property. An experimentally determined minimum threshold of 0.0025 was used; this acts as a penalty value for the changepoint and sets the minimum improvement for the total residual error when calculating the changepoints.

The gfpop (Graph Constrained Functional Pruning Optimal Partitioning) algorithm is a graph-based changepoint detection algorithm primarily created for applications in fields, such as medicine, containing large sequences of data with abrupt changes. The algorithm requires the user to input a built-in or manually created graph consisting of nodes representing the states of the signal, edges for the transitions between states, and a loss function to analyze the graph. The edges for the graph have a variety of parameters, although for this application, the only parameter that was tuned was the penalty on the edges for transitioning between states. The graph for the bee dataset, shown in

Figure 27, contained four states representing air, bees, and the increase and decrease from the air to the bee; this graph gave better results than a binary graph of only air and bee states. The edges transitioning from the air to an increase and from the bee to a decrease had a penalty of 0.005. The graph penalties, as well as the procedures described below, were

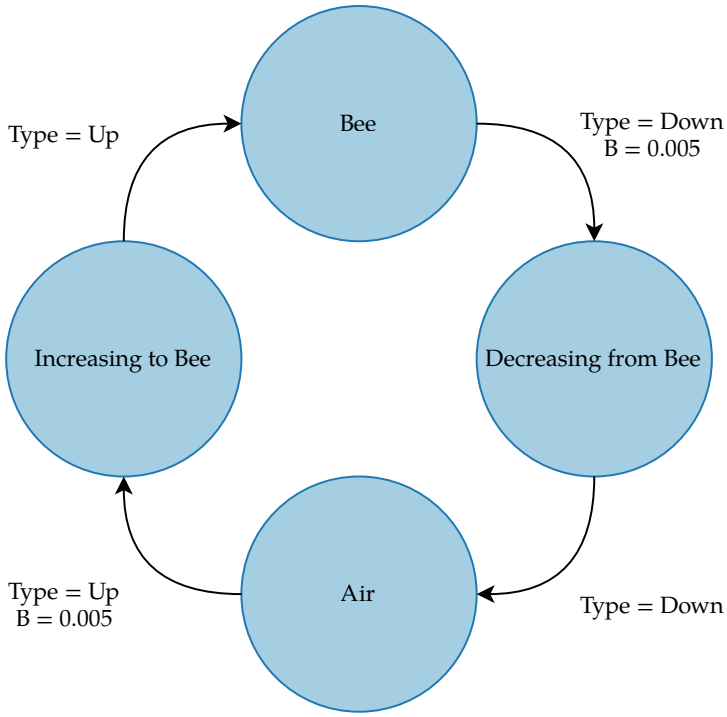



Figure 27: gfpop graph for detecting bees in the dataset. Null edges that allow the algorithm to stay in the same state from point to point are not shown.

manually tuned on a few images from the training dataset.

For the image analysis using the changepoint detection algorithms, both the rows and columns of the bee images were fed into the algorithms. The results were gathered based off of analyzing only the rows, only the columns, and checking both the rows and columns to see if they gave a changepoint within some margin.

The procedure for analyzing rows comprises the following steps: applying a heuristic to filter out rows that obviously do not contain bees, applying the changepoint detection algorithm to the remaining rows, then applying further heuristics to determine if the changepoints were likely associated with a bee signal. Concretely, we first check whether the range of the row is higher than the mean of the row. This was meant to check whether the target row was either empty air or a stationary object, as the empty air has a low range and a low mean, while stationary objects generally have a low range and high mean; if the range

is greater than the mean, the row is marked as potentially containing a bee. The changepoint algorithm is then applied to these marked rows. After the algorithm is applied to the rows, the number of changepoints is checked, and if it is more than 5, the row is thrown out. This is because the stationary objects that pass through the first check often contained upwards of 10 changepoints, while the true bees often only contained around 5. The last check verifies that the voltage of the changepoint is larger than the mean of the row. If it is, then the changepoint is classified as a valid bee.

The procedure for analyzing the columns is similar. First, the algorithm checks the range of each row and compares it to the mean; from those results, the algorithm creates a vector of the valid rows where an insect could be. It then iterates through each column and checks whether the changepoints it detects are in valid rows before checking the value of the changepoint against the mean of the row. If the changepoint fits the same criteria used in the row analysis process, it is classified as a valid bee.

The process for checking both rows and columns involves iterating through the rows in the same manner as before, but after detecting changepoints in a row, it then iterates through those columns, and after finding a changepoint in the column, checks the norm between the changepoint it detected in the row and the changepoint it detected in the column. If they are within a distance of 4, as calculated by the L2 norm, the changepoint is considered to be a bee.

Evaluation Metrics

All the methods used were analyzed based on their recall, precision, and (MCC) [14] scores. It is well known that accuracy is not an informative metric for imbalanced data [148]; thus, accuracy is not used in this paper. Recall is the percentage of the target class that the method was able to identify and ranges from 0 to 1. Precision is the percentage of true positives relative to the total number of positive identifications and ranges from 0 to 1. MCC

is a metric that gives a high score if the method is able to achieve an accurate value in each of the four categories of the confusion matrix. The MCC ranges between $-1$ and $+1$. Positive values indicate a positive correlation between the predictions and the true labels, with $+1$ being perfect prediction. A value of 0 indicates that the prediction is no better than random predictions. Negative values indicate a negative correlation between the predictions and the true labels, with $-1$ indicating complete disagreement between the predictions and the true labels.

When evaluating our methods, we consider recall more important than precision because the focus was to identify as many of the bees as possible. However, a low precision is not entirely outweighed by a high recall: low precision means there were many false positives, which researchers would have to look through to validate whether bees were present or not. Thus, a balance of recall and precision is desirable; a high MCC score indicates a good balance of recall and precision. While the $F_\beta$ score could be used as a single metric that combines recall and precision, we chose to use MCC because it is invariant to which class is labeled "positive", unlike $F_\beta$ [149].

## Results

The work in this paper has two performance goals: find which images contain insects and find the rows (i.e., range bins) the insects are located in. Methods that correctly detect most insect-containing images could be used on future datasets to reduce the number of images humans have to look through to label the insects. Likewise, methods that correctly detect most insect-containing rows would reduce the labeling burden, as well as help automatically compute parameters of interest, such as the number of insects and their wingbeat frequencies. While we ultimately want to detect all insects, we also want to have a reasonably high precision so we do not have to look through many false positives. For the methods that take rows as input, row-based and image-based results are reported. For the 2D CNNs, only

image-based results are reported. The following sections detail the classification performance of the feature engineering, deep learning, and changepoint detection methods, as well as an in-depth analysis of the features described in the Feature Engineering Methods section.

Supervised Methods

Feature Analysis To see which of the extracted features had the most predictive power for discriminating between classes ("Bee" and "No bee"), we computed the mutual information [150] between the features and the class labels. We used the `mutual_info_classif` function from Python's scikit-learn library to compute the mutual information scores; this function is based on a k-nearest-neighbor method of computing mutual information between continuous and discrete variables [151]. The results are shown in Figure 26. The top three features were the time-domain skewness, the average skewness of each row in the wavelet scalogram, and the time-domain kurtosis. Figure 28 shows a pairwise scatter plot of these three features; none of the top three features are separable, but there are clear differences between the "bee" and "no bee" distributions in Figure 28. The following paragraphs interpret the discrimination ability of each of the top three features.

Skewness is a measure of asymmetry about the sample mean, with negative values indicating a long left tail of the distribution and positive values indicating a long right tail of the distribution. In our lidar data, insects flew through the beam for a small fraction of the total 1024 pulses; the pulses that hit insects have a large negative value compared to the rest of the pulses at that range bin, which results in a negative skewness. The majority of rows that do not contain insects have a skewness close to zero, as seen in Figure 28.

The rows of the wavelet scalogram correspond to frequency bins; thus, a row represents how a particular frequency component changes over time. Observations that did not contain any targets have little frequency content, which results in most of the scalogram values being close to 0; this leads to a small skewness value, as can be seen in "No Bee" distribution's

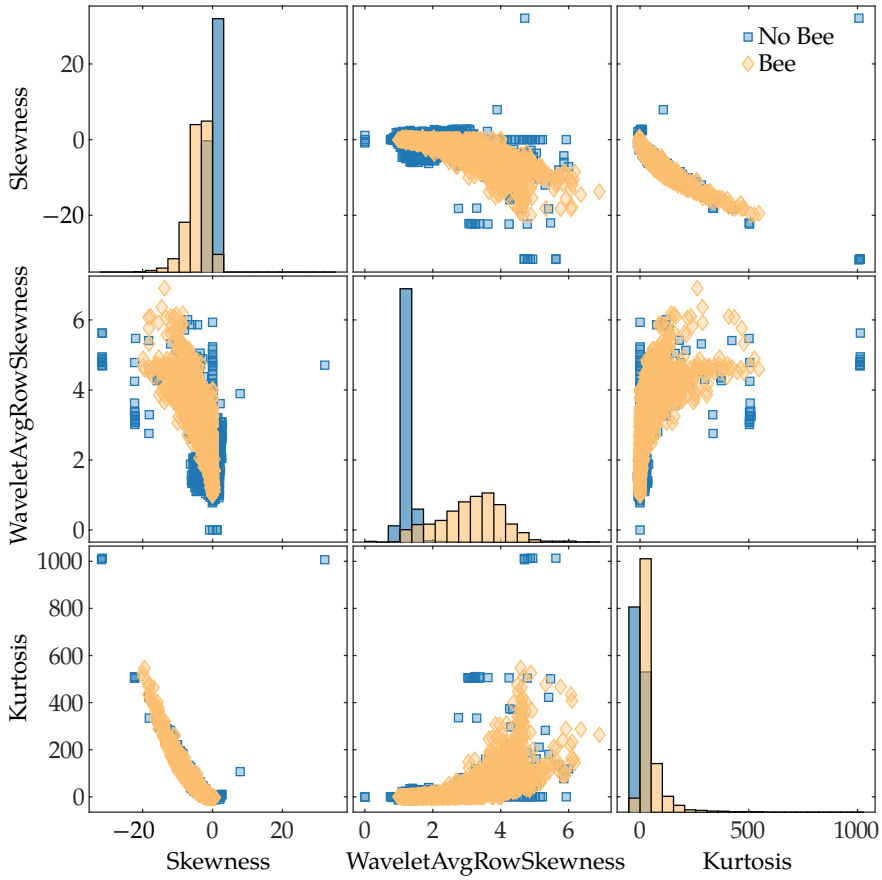



Figure 28: Pairwise scatter plot of the three features with the highest mutual information with the class labels. The main diagonal shows histograms of each feature's distribution. The features are ordered by descending mutual information values from top-to-bottom and left-to-right.

peak in the middle plot in Figure 28. When a bee is in the beam, its wings create frequency content in the return signal; this frequency content shows up in the relevant scalogram rows for a short period of time, which results in positive skewness for those rows. This is why the "Bee" distribution tends to have higher skewness values. However, there are also non-bee observations with high skewness, as seen in Figure 28.

Kurtosis is a measure of a distribution's tail extremity; for the sample kurtosis, tail extremity indicates outliers in the data [152]. Rows that contain no targets are characterized by small amounts of noise and will have kurtosis values near 0, as seen by the peak in the "No bee" kurtosis distribution in Figure 28. Insects flying through the beam for short periods of

time result in returns that differ significantly from their surroundings, i.e., they are outliers; consequently, insect rows will generally have a positive kurtosis value. Other noisy targets, such as moving blades of grass, will also tend to have positive kurtosis values.

The low mutual information between the harmonic frequency values and the class labels, as seen in Figure 26, is surprising given that the wingbeat-modulation lidar method is based upon being able to determine the insects' wingbeat spectra. This low mutual information is due to the significant overlap in the "Bee" and "No bee" harmonic frequency distributions, as shown in Figure 29. The distributions in Figure 29 were computed on each row of the training dataset. As seen in Figure 29a, compared to the "No bee" distribution, there is an extra bump in the "Bee" distribution around 200 Hz, corresponding to the expected bee fundamental wingbeat frequency. The tails of the "No bee" distributions are more smooth, with no increase around the expected wingbeat frequencies. The majority of the "No bee" rows should exhibit no harmonic structure; in this case, an ideal frequency estimation algorithm would not report erroneous harmonic structure. Our fundamental frequency estimation algorithm looks for the largest local maximum in the harmonic product spectrum; if any local maxima exist, our algorithm returned an estimated frequency, otherwise the algorithm returned 0. In Figure 18d, the small peak around 100 Hz in the stationary object's spectrum would be interpreted as a local maxima and would thus be considered that observation's fundamental frequency. This behavior is problematic for the non-bee observations and contributed to the overlap between the distributions.

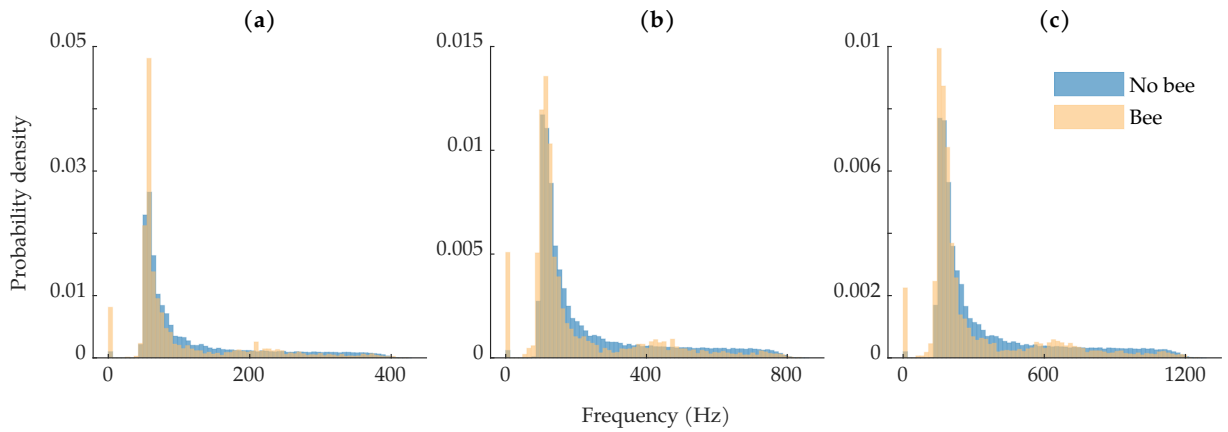



Figure 29: Class-wise distributions of harmonic frequency computed on each row in the training set: (**a**) distribution of the first harmonic frequency; (**b**) distribution of the second harmonic frequency; (**c**) distribution of the third harmonic frequency. The counts at bin 0 are from observations where no local maximum was found in the harmonic product spectrum.

Data Sampling Table 14 shows the undersampling and augmentation ratios that resulted in the best MCC scores on the validation set. The grid search was performed three times for each algorithm, and the results for each run are shown in Table 14. Most algorithms except the 1D CNNs did not benefit from data augmentation; this is partly in contrast to our previous work, where both AdaBoost and RUSBoost benefited from data augmentation [131]. Compared to our previous work, we have extracted more features, which might have helped the classifiers learn the decision boundary without data augmentation. There is no apparent pattern to which algorithms benefited from undersampling. Of the algorithms that selected different optimal undersampling ratios between the three runs, the associated MCC scores did not vary significantly, except for the SVM. However, MCC scores tended to vary significantly over the grid search parameters. As an example, one of the 5-layer 1D CNNs had MCC values range between 0.221 to 0.736. The data sampling results suggest that undersampling and oversampling can affect performance but that the optimal values are not necessarily independent of the model's initial conditions, especially for the CNNs.

Table 14: Undersampling and augmentation ratios that resulted in the best MCC scores on the validation set. The results are shown for each of the three runs.

| **Method** | **Undersampling Ratio** | **Augmentation Ratio** | **MCC** |
|---|---|---|---|
| AdaBoost | [0.50, 0.50, 0.50] | [0, 0, 0] | [0.800, 0.800, 0.800] |
| RUSBoost | [0.25, 0.25, 0.25] | [0, 0, 0] | [0.648, 0.648, 0.648] |
| SVM | [0.75, 0.75, 0.00] | [1, 0, 0] | [0.459, 0.215, 0.359] |
| MLP 1 | [0.00, 0.00, 0.25] | [0, 0, 0] | [0.812, 0.812, 0.826] |
| MLP 3 | [0.00, 0.00, 0.00] | [0, 0, 0] | [0.818, 0.818, 0.818] |
| MLP 5 | [0.00, 0.00, 0.00] | [0, 0, 0] | [0.815, 0.815, 0.815] |
| MLP 7 | [0.25, 0.25, 0.00] | [0, 0, 0] | [0.823, 0.819, 0.818] |
| 1D CNN 1 | [0.00, 0.25, 0.25] | [1, 1, 1] | [0.737, 0.730, 0.736] |
| 1D CNN 3 | [0.00, 0.00, 0.50] | [1, 10, 0] | [0.736, 0.764, 0.764] |
| 1D CNN 5 | [0.25, 0.50, 0.00] | [1, 1, 1] | [0.714, 0.745, 0.738] |
| 1D CNN 7 | [0.25, 0.00, 0.25] | [1, 1, 1] | [0.725, 0.751, 0.763] |

Validation For the row-based methods, the training set was sampled during hyperparameter tuning using the undersampling and augmentation ratios in Table 14; data sampling was not performed for the image-based methods. Table 15 shows the MCC scores that the tuned models obtained on the validation set after hyperparameter tuning. For the row-based methods, the best hyperparameters were chosen based upon the row-based MCC score. For all methods, if the default hyperparameters performed better than all hyperparameter combinations encountered during hyperparameter tuning, the default parameters were used for the final training.

For the row-based methods, the feature-based MLPs obtained the best row-based average MCC scores on the validation set, as seen in Table 15. RUSBoost achieved the highest recall, at the expense of 45.3% precision. The 1D CNNs and the linear SVM all achieved average MCC scores around 0.75.

Table 15: Classification results on the validation set. The mean score from the three runs is shown, and the standard deviation is shown in parentheses. The best results for each metric are shown in bold. Methods marked with a * indicate that the method achieved the same results every run.

| **Method** | **Rows** | | | **Images** | | |
|---|---|---|---|---|---|---|
| | **Recall** | **Precision** | **MCC** | **Recall** | **Precision** | **MCC** |
| AdaBoost * | 0.835 (0.000) | 0.786 (0.000) | 0.809 (0.000) | 0.854 (0.000) | 0.965 (0.000) | 0.870 (0.000) |
| RUSBoost * | **0.954** (0.000) | 0.453 (0.000) | 0.655 (0.000) | **0.977** (0.000) | 0.695 (0.000) | 0.728 (0.000) |
| SVM | 0.740 (0.021) | 0.755 (0.042) | 0.745 (0.016) | 0.794 (0.031) | 0.907 (0.061) | 0.785 (0.032) |
| MLP 1 | 0.802 (0.010) | 0.850 (0.005) | 0.825 (0.003) | 0.851 (0.013) | 0.965 (0.007) | 0.867 (0.005) |
| MLP 3 * | 0.798 (0.000) | 0.851 (0.000) | 0.823 (0.000) | 0.866 (0.000) | 0.962 (0.000) | 0.876 (0.000) |
| MLP 5 * | 0.806 (0.000) | **0.854** (0.000) | **0.829** (0.000) | 0.874 (0.000) | 0.931 (0.000) | 0.858 (0.000) |
| MLP 7 | 0.815 (0.013) | 0.833 (0.020) | 0.823 (0.004) | 0.883 (0.018) | 0.939 (0.016) | 0.871 (0.015) |
| 1D CNN 1 | 0.724 (0.051) | 0.799 (0.068) | 0.757 (0.009) | 0.798 (0.036) | 0.854 (0.083) | 0.746 (0.043) |
| 1D CNN 3 | 0.717 (0.051) | 0.805 (0.079) | 0.757 (0.012) | 0.815 (0.034) | 0.929 (0.036) | 0.816 (0.038) |
| 1D CNN 5 | 0.659 (0.035) | 0.832 (0.060) | 0.738 (0.007) | 0.746 (0.040) | 0.949 (0.010) | 0.783 (0.034) |
| 1D CNN 7 | 0.686 (0.040) | 0.835 (0.063) | 0.754 (0.007) | 0.761 (0.039) | 0.924 (0.048) | 0.775 (0.011) |
| 2D CNN 1 | — | — | — | 0.731 (0.002) | 0.965 (0.000) | 0.782 (0.002) |
| 2D CNN 3 | — | — | — | 0.867 (0.004) | 0.960 (0.009) | 0.876 (0.009) |
| 2D CNN 5 | — | — | — | 0.908 (0.004) | 0.970 (0.009) | 0.912 (0.004) |
| 2D CNN 7 | — | — | — | 0.908 (0.017) | **0.973** (0.008) | **0.914** (0.007) |

Looking at the image-based results in Table 15, we see that RUSBoost found the most insect-containing images, achieving the highest recall. The MLPs also performed well, achieving high precision scores above 93% and MCC values around 0.87. The three largest 2D CNNs achieved the top three average MCC scores, followed by the 3-layer MLP.

Testing Using the best parameters found during the data sampling and hyperparameter tuning phases, each model was trained one final time on the full training and validation datasets. These trained models were then tested on the testing set, which was collected approximately one month after the training data was collected. Table 16 shows the recall, precision, and MCC scores that the models obtained on the testing set.

For the row-based methods, RUSBoost performed the best, achieving a 0.948 recall and

0.752 MCC score. The majority of the methods did not generalize well to the training set, achieving significantly lower recall scores. However, the majority of the row-based methods performed well at identifying which images contained insects. This suggests that the methods were doing some combination of

- Finding the insects but not predicting all the rows that the insects spanned.
- Predicting insects in incorrect rows but doing so in images that contain insects.
- Missing some insects in images that contain multiple insects.

Looking at the image results in Table 16, we see that RUSBoost achieved the highest recall and MCC scores. The MLPs achieved the best MCC scores after RUSBoost. While the 1D CNNs often achieved high average recall, their recall and precision scores often varied significantly between the three runs, as indicated by the standard deviations in the image-based recall and precision columns. The 2D CNNs and AdaBoost did not generalize well to the testing set and achieved the lowest average recall and MCC scores.

Overall, RUSBoost achieved the best recall on both datasets and generalized to the testing set; this might be due in part to the undersampling RUSBoost performs during training, which creates a new undersampled dataset for each learner in the ensemble, rather than a single undersampled dataset for each learner. However, RUSBoost's low precision, particularly on the validation set, is a potential downside; if one were to choose only one row-based model based on row-based validation-set performance, RUSBoost would not have been chosen. The feature-based MLPs performed consistently at detecting insect-containing images in both validation and testing sets; while their row-based testing results did not generalize as well, the consistent image-based performance is a plus for future applications. More discussion of model selection and algorithm recommendations follows in the Discussion section.

Table 16: Classification results on the testing set. The mean score from the three runs is shown, and the standard deviation is shown in parentheses. The best results for each metric are shown in bold. Methods marked with a * indicate that the method achieved the same results every run.

| **Method** | **Rows** | | | **Images** | | |
|---|---|---|---|---|---|---|
| | **Recall** | **Precision** | **MCC** | **Recall** | **Precision** | **MCC** |
| AdaBoost * | 0.296 (0.000) | 0.771 (0.000) | 0.476 (0.000) | 0.418 (0.000) | 0.935 (0.000) | 0.519 (0.000) |
| RUSBoost * | **0.948** (0.000) | 0.600 (0.000) | **0.752** (0.000) | **0.995** (0.000) | 0.837 (0.000) | **0.856** (0.000) |
| SVM | 0.605 (0.132) | 0.636 (0.076) | 0.616 (0.091) | 0.895 (0.028) | 0.759 (0.061) | 0.705 (0.040) |
| MLP 1 | 0.541 (0.002) | 0.838 (0.026) | 0.672 (0.012) | 0.811 (0.001) | 0.919 (0.048) | 0.791 (0.040) |
| MLP 3 * | 0.550 (0.000) | 0.831 (0.000) | 0.674 (0.000) | 0.821 (0.000) | 0.882 (0.000) | 0.768 (0.000) |
| MLP 5 * | 0.488 (0.000) | **0.860** (0.000) | 0.646 (0.000) | 0.753 (0.000) | **0.948** (0.000) | 0.771 (0.000) |
| MLP 7 | 0.663 (0.146) | 0.739 (0.041) | 0.696 (0.099) | 0.891 (0.045) | 0.826 (0.005) | 0.769 (0.028) |
| 1D CNN 1 | 0.600 (0.195) | 0.650 (0.195) | 0.602 (0.010) | 0.828 (0.131) | 0.785 (0.106) | 0.680 (0.058) |
| 1D CNN 3 | 0.697 (0.076) | 0.645 (0.233) | 0.654 (0.106) | 0.892 (0.064) | 0.780 (0.083) | 0.722 (0.038) |
| 1D CNN 5 | 0.700 (0.072) | 0.573 (0.237) | 0.620 (0.126) | 0.907 (0.060) | 0.815 (0.148) | 0.760 (0.128) |
| 1D CNN 7 | 0.329 (0.049) | 0.691 (0.351) | 0.466 (0.166) | 0.688 (0.151) | 0.857 (0.179) | 0.636 (0.060) |
| 2D CNN 1 | — | — | — | 0.291 (0.003) | 0.921 (0.000) | 0.414 (0.002) |
| 2D CNN 3 | — | — | — | 0.539 (0.346) | 0.861 (0.070) | 0.547 (0.184) |
| 2D CNN 5 | — | — | — | 0.488 (0.101) | 0.924 (0.020) | 0.564 (0.086) |
| 2D CNN 7 | — | — | — | 0.544 (0.367) | 0.907 (0.005) | 0.592 (0.267) |

Changepoint Detection

The results for the changepoint detection algorithms were also gathered on both an image basis and a row basis using the testing set. For the image results shown in Table 17, when the two algorithms are compared, MATLAB's `findchangepts` achieves a higher recall than gfpop. For the row results shown in Table 17, both algorithms struggled to achieve a high number of true positives; both algorithms achieved higher precision when analyzing the columns compared to analyzing the rows.

One of the main motivations for using the changepoint detection algorithms was that the total runtime would be less than traditional machine learning algorithms, along with a decrease in required computational power. All the variations of the two algorithms were

run with 32 CPUs and 80 GB of RAM. Since each image was independent of the other, the algorithms could process all the images in parallel before compiling the results together. The `findchangepts` function completed its workload in 21 s, 71 s, and 21 s for the rows, columns, and both, respectively. The gfpop algorithm completed its workload in 834 s, 1205 s, and 828 s for the rows, columns, and both, respectively. In general, these runtimes are far lower than the time it takes to train a machine learning algorithm.

Overall, the changepoint detection methods achieved lower row-based results than the supervised methods, on average. In particular, the changepoint detection methods achieved much lower row-based precision, indicating a higher number of false positives. This is due to the fact that any object that moves through the laser and has a significant return would result in a changepoint. The changepoint detection methods performed much better on the image-based metrics than on the row-based metrics, as seen in Table 17. While the changepoint detection methods still had worse image-based recall than RUSBoost, the gfpop rows and gfpop columns algorithms achieved the second and third best image-based recall, respectively. In spite of their comparable recall, the gfpop algorithms achieved lower image-based precision and MCC scores than the comparable supervised learning methods did; the MATLAB `findchangepts` "rows" and "both" methods, on the other hand, achieved precision comparable to many of the supervised methods. Since changepoint detection is an unsupervised method, the lower performance is not surprising: unsupervised methods do not have a priori knowledge of the class labels like supervised methods do and thus cannot be directly optimized to predict class labels [4]. Although the best supervised methods achieved better results, the changepoint methods achieved similarly high recall without needing to be trained on labeled data. This is a significant advantage for the changepoint detection methods, as labeling data is a time-consuming process.

Table 17: Changepoint detection results on the testing set. The best results for each metric are shown in bold.

| **Method** | **Rows** | | | **Images** | | |
|---|---|---|---|---|---|---|
| | **Recall** | **Precision** | **MCC** | **Recall** | **Precision** | **MCC** |
| MATLAB Rows | 0.073 | 0.097 | 0.078 | 0.832 | 0.842 | **0.741** |
| MATLAB Columns | **0.917** | 0.116 | 0.316 | 0.884 | 0.650 | 0.581 |
| MATLAB Both | 0.034 | 0.049 | 0.031 | 0.750 | **0.855** | 0.694 |
| gfpop Rows | 0.322 | 0.243 | 0.274 | **0.936** | 0.676 | 0.647 |
| gfpop Columns | 0.668 | 0.196 | **0.354** | 0.929 | 0.658 | 0.621 |
| gfpop Both | 0.258 | **0.317** | 0.281 | 0.806 | 0.690 | 0.576 |

## Discussion

### Sources of Uncertainty and Error

Our ground truth labels are the primary source of uncertainty. Since we manually labeled all insects via visual inspection, it is possible that some insects were missed and some non-insects were labeled as insects. Without any independent ground truth to validate our labels, we cannot easily verify that all labeled insects truly were insects; this is especially pertinent for observations that were labeled as low-confidence insects. The digitizer's sampling jitter, which resulted in most insects spanning multiple 0.75 m range bins during a single laser pulse (a physically impossible event), also contributed to uncertainty in the labels. When labeling each insect's bounding box, deciding exactly how many rows the insect spanned was somewhat arbitrary because the effect of the sampling jitter decayed over multiple range bins. This decay and smearing can be seen in Figure 23.

Effects of Confidence Ratings

As noted in the Dataset section, the bee events were labeled with confidence ratings between 1 and 4, with 4 being the highest confidence rating. High-confidence bees had obvious signatures in both the time and frequency domains, whereas medium-confidence bees generally had obvious signatures in only one domain. For example, the two insect examples in Figure 18 are high-confidence bees, whereas the blips in Figure 23b,c are low-confidence bees. Thus, the confidence ratings are tied to labelling uncertainty and data quality. There are 73 rows with a confidence of 1, 1479 rows with a confidence of 2, 5288 rows with a confidence of 3, and 402 rows with a confidence of 4.

Figure 30 shows recall as a function of confidence rating. For the majority of the algorithms, the high-confidence bees are identified more often than low-confidence bees. This makes intuitive sense because high-confidence bees should be the most distinct. Interestingly, many of the supervised learning algorithms had higher recall for bees with a confidence of 1 than a confidence of 2. This suggests that the algorithms may have identified some salient features for low-confidence bees, such as the blips shown in Figure 23. These results could also be due to labeling uncertainty in the bounding boxes, as discussed in the following section, or imprecision of the confidence labels.

Interestingly, the gfpop "rows" and MATLAB "rows" methods exhibit the opposite trend: recall decreases as confidence increases. The "both" methods exhibit the same trend because they are similar to the "rows" methods. It is possible that some of the high-confidence bee rows were discarded by the heuristic that bees typically had fewer than five changepoints. Conversely, low-confidence bees that were in the beam for a short period of time would have few changepoints and would thus pass through that heuristic. The column-based changepoint detection algorithms exhibit the expected behavior. High-confidence bees, especially ones which had a prominent body return signal, exhibit abrupt changes between range bins: range bins before and after the bee have a low return amplitude; this likely contributed to the

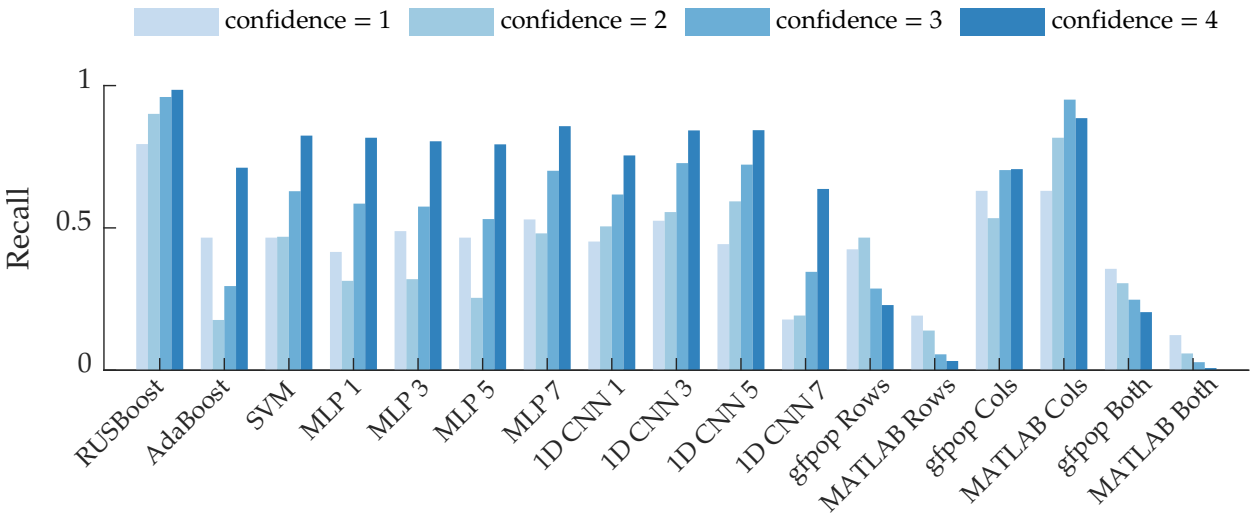



Figure 30: Row-based recall grouped by the confidence ratings of the bee events. The average recall is displayed for the supervised learning algorithms.

behavior of the column-based changepoint methods shown in Figure 30.

Model Generalization

As noted in the Results section, many of the supervised methods did not perform as well on the testing set as they did on the validation set. This suggests some amount of overfitting to the training and validation sets, which were collected at a different time than the testing set. Although the training/validation and testing data were collected at the same location, the conditions were not exactly the same: ambient temperature was different, resulting in different bee activity; height of the foliage in front of the beehives was different; and some of the pan and tilt angles that the data were collected at were intentionally different. These variations resulted in different bee and background characteristics between the two datasets. Consequently, the training/validation and testing dataset distributions are not identical, nor are either of the sets representative of the entire possible data distribution at the data-collection location.

As was seen in Tables 15 and 16, the methods generalized better when classifying images than classifying rows. This suggests that these methods were still detecting the insects but

not detecting every row the insect occupied due to sampling jitter; or the methods simply missed an insect in images that contained multiple insects.

Figure 31a shows an example where a 3-layer MLP correctly found the insect but predicted more rows than were labeled. In this case, only one row was labeled but the insect spanned three rows, which the MLP predicted. This leads to an artificially low row-based precision, as the human labels were imperfect and did not span all the insect's rows. Additionally, the MLP predicted the reflection artifacts as additional bees. Figure 31b shows the opposite case, where the 3-layer MLP predicted fewer rows than were labeled. Although the algorithm still found the bee, the missed rows led to an artificially low row-based recall. The situations demonstrated in Figure 31 show that the label ambiguity caused by the digitizer's sampling jitter and the human-labeled bounding boxes contributed to artificially low row-based performance metrics.

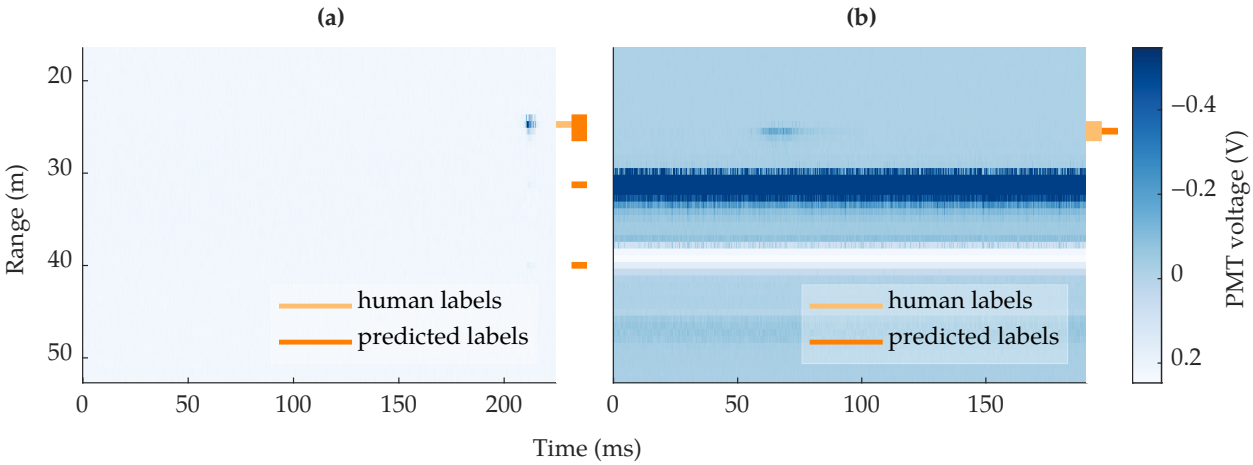



Figure 31: Example images where a 3-layer MLP found the bee but did not achieve perfect recall or precision. (**a**) An example where the algorithm predicted extra rows compared to the human-labeled rows; the algorithm also predicted the ringing artifacts as bees. (**b**) An example where the algorithm predicted one out of three of the human-labeled rows.

Better generalization performance might have been achieved if we randomly split the combined June and July datasets into training/validation/testing sets, rather than using June for training/validation and July for testing. However, we wanted to model a real-world

scenario where labeled data from a previous campaign could be used to train models for a new data collection campaign.

Overall, lack of generalization is one of the primary limitations of both the supervised and changepoint detection methods. It is unlikely that either category of methods will generalize to a new data-collection location or species without needing to be retrained or have some algorithm parameters adjusted. Additionally, recent insights have shown that CNNs with subsampled convolution and pooling operations are sensitive to small image translations [153]; it is possible that our 2D CNNs are sensitive to the bee locations, leading to lack of generalization, but some preliminary analysis suggests this is not the case. Eventually, as more data are collected across various locations, seasons, and species, the methods should start to generalize.

Algorithm Recommendations

For supervised learning, the general rule for model selection is to choose the model with the best validation performance. Based upon the row-based MCC, which was used during hyperparameter tuning, the 5-layer MLP would be chosen as the "best" model. Indeed, the 5-layer MLP also performed okay on the testing set, so this would not be a bad decision. However, RUSBoost, which had the lowest row-based MCC on the validation set, performed best on the testing set; RUSBoost did, however, have the highest row-based recall on the validation set, as well as the highest image-based recall on the validation set. Most methods' generalization ability was better for the image-based results; given this, one might consider using the image-based validation results for final model selection, rather than the row-based results.

In general, for the supervised methods we tested, we recommend using RUSBoost and/or a feature-based MLP, as those methods performed consistently well across the validation and testing sets. The 1D CNNs achieved good average performance on the testing set but had

high variability between runs; this variability indicates that the models were not robust. The 2D CNNs did not generalize to the testing set and also had high variability between runs. With more fine-tuned architectures, it is possible that CNNs could perform and generalize well.

Compared to the changepoint detection methods, the higher performance of the supervised methods comes at a cost: needing labeled data. Needing labeled data is the major limitation of the supervised methods because the primary point of automated detection is not having to manually detect insects. If labeled data already exist, then we recommend using one of the supervised methods. However, if labeled data does not exist, the changepoint detection methods are a promising place to start because they can be tuned on a few images and still achieved high recall of insect-containing images. If finding as many insect images as possible is paramount, and looking through false positives is not a huge burden, then we recommend the gfpop "rows" or "both" methods; these methods achieved image-based recall scores around 93%. If looking through many false positive images is too large of a burden, then we recommend the MATLAB "rows" or "both" algorithms because they achieved precision scores around 85%.

When the data are not labeled, a semi-supervised learning approach could be employed: use a changepoint detection method to create an initial set of imperfect labels, and then use those labels to train one of the supervised learning algorithms. The data could then be fed through the trained supervised algorithm, and any new positive predictions that were not predicted by the changepoint detection method can be inspected and labeled. This process can be repeated until no new insect images are found.

Future Research Directions

While good image-based recall and precision, like that achieved in this paper, would help reduce the amount of non-insect images requiring manual analysis, manual analysis would

still need to be performed on the insect images to obtain quantities of entomological interest, such as the total number of insects observed. Consequently, future research should work towards improving the row-based insect detection performance and reducing sampling jitter; high row-based precision and recall would allow scientists to easily calculate the total number of insects, their spatial and temporal distributions, their wingbeat frequency distribution, etc. Towards this goal, the fundamental frequency estimation algorithm should be improved to detect less spurious harmonics in stationary object and noise observations. This would lead to better class separability in the harmonic features.

The sampling jitter is the biggest impediment to having an automatic count of the actual number of insects. The smearing of insect observations across multiple range bins limits the ability of the row-based methods to accurately detect insects and leads the row-based methods to produce artificially large insect counts. If the sampling jitter cannot be reduced significantly, post-processing methods should be explored. Using an image segmentation CNN is one possible solution, as the model could be trained to predict the insects' bounding boxes.

Including the labels' confidence ratings into the detection process is an interesting avenue for future work. The confidence ratings provide extra probability information that could prove useful. Lastly, we are also interested in trying more unsupervised and semi-supervised methods, such as anomaly detection methods.

## Conclusions

Automatically detecting insects is crucial for the future of "big data" in entomology. In this paper, we present a comparison of supervised learning and changepoint detection methods for detecting insects using wingbeat-modulation lidar. We collected and published a new lidar dataset containing honeybee activity at various times and locations around a line of beehives. We found that both methods would be able to successfully reduce the need for manual data analysis: the supervised learning method with the best MCC score on the testing

set correctly identified 99.5% of the insect-containing images and 83.7% of the non-insect images; the changepoint detection method with the best MCC score on the testing set correctly identified 83.2% of the insect-containing images and 84.2% of the non-insect images. In addition to comparing and recommending algorithms, we provide an analysis of which features were relevant for insect detection, which can help guide future development. Notably, we introduce wavelet-based features, which we have not seen in the previous literature; the wavelet-based features were all ranked in the top 50% of relevant features. While there is still room for improvement, the automated detection methods we present are capable of reducing the amount of manual data analysis required and can provide rough estimates of insect quantity and activity.

## Acknowledgements

This material (AFRL-2023-5181) is based on research sponsored by Air Force Research Lab under agreement number FA8650-21-1-1176. The U.S. Government is authorized to reproduce and distribute reprints for Governmental purposes notwithstanding any copyright notation therein. The views and conclusions contained herein are those of the authors and should not be interpreted as necessarily representing the official policies or endorsements, either expressed or implied, of Air Force Research Lab or the U.S. Government.

Computational efforts were performed on the Tempest High Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure at Montana State University. The authors would also like to acknowledge Caroline Xu and Ryan Ficken for performing prior work on changepoint detection and wavelet feature extraction, respectively. We would also like to acknowledge Caroline Xu, Ryan Ficken, and Walden Marshall for helping collect data. We thank the reviewers for their helpful comments that have improved this manuscript.

# WINGBEAT-MODULATION LIDAR HARDWARE AND SOFTWARE UPGRADES

As stated in the , my drone-detection work expanded upon my insect-detection work. We used the same wingbeat-modulation lidar instrument with some modifications, and we expanded upon my data analysis techniques. This chapter details the following modifications that were made to the lidar to enable the drone-detection experiments: reducing signal reflections, creating a digital PMT gain controller, upgrading to a faster digitizer, and writing software to control pan-tilt controllers.

## Removing Transmission-line Reflections

During the insect experiments, "reflections" of the insects showed up at further distances, as shown in Figure 32 (reproduced from Figure 25). These reflections were essentially inverted copies of the actual insect signal; consequently, our data analysis techniques tended to interpret these reflections as additional insects.

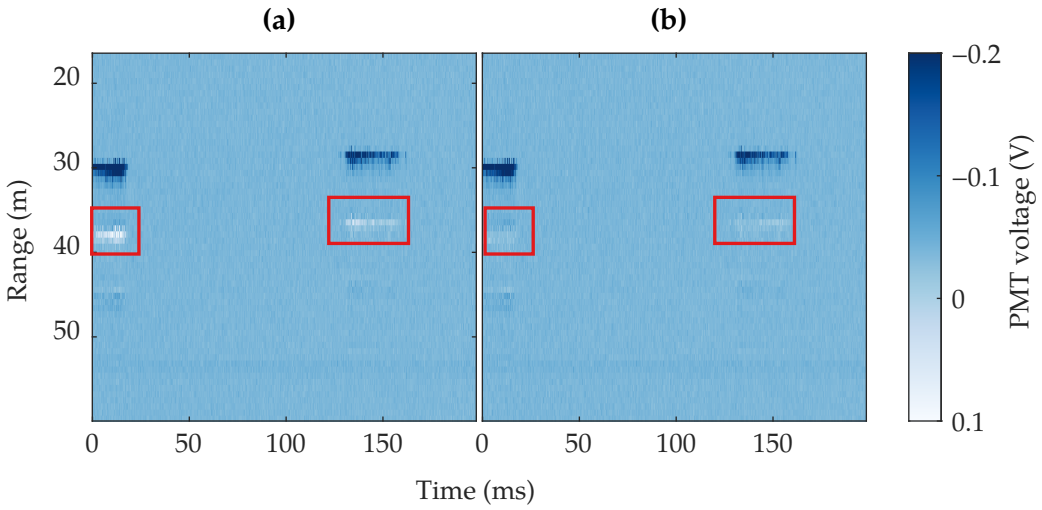



Figure 32: Example of an image before (**a**) and after (**b**) thresholding the reflections. The reflections are highlighted by red rectangles.

Initially, we thought these reflections were caused by the PMT "ringing" when it received a high intensity of returned light. However, PMT ringing should produce a negative voltage because the PMT can only output negative current; the reflections' voltages were always positive, which means ringing was probably not the cause of the reflections. While at an SPIE conference, a colleague asked representatives from Hamamatsu about the reflections we saw in the data. The Hamamatsu engineers said the reflections should not be from the PMT and suggested they might be due to bad wiring.

From electromagnetic theory, we know that an impedance discontinuity will cause an electromagnetic wave to reflect [154]. This reflection will travel back towards the source. The reflection's amplitude may have the opposite sign of the original wave's amplitude. A positive-voltage reflection from an impedance discontinuity could accurately model the reflections we saw in the insect data.

To investigate the cause of the reflections, I looked at the PMT response as a function of gain and with different BNC cables. If the reflections were primarily caused by ringing, reducing the PMT's gain should eventually eliminate the reflection. If the reflection is due to an impedance discontinuity, replacing the cable would remove the impedance discontinuity if the discontinuity is in the cable. Figure 33 shows the PMT output for different gain settings, and Figure 34 shows the PMT output for two different cables.

In Figure 33, each response exhibits ringing (the oscillations that occur before the positive-voltage reflection) around 950 ns, but the ringing's amplitude decreases as gain decreases. The amplitude of the reflection also decreases. These observations show that ringing is present.

In Figure 34, the original cable caused a reflection, whereas the new cable did not. This confirms that reflection was caused by a bad cable. The new cable still exhibits ringing after 940 ns. This shows that ringing and the reflection are distinct phenomena. It is likely that the original BNC cable had an impedance discontinuity somewhere in the middle of the cable

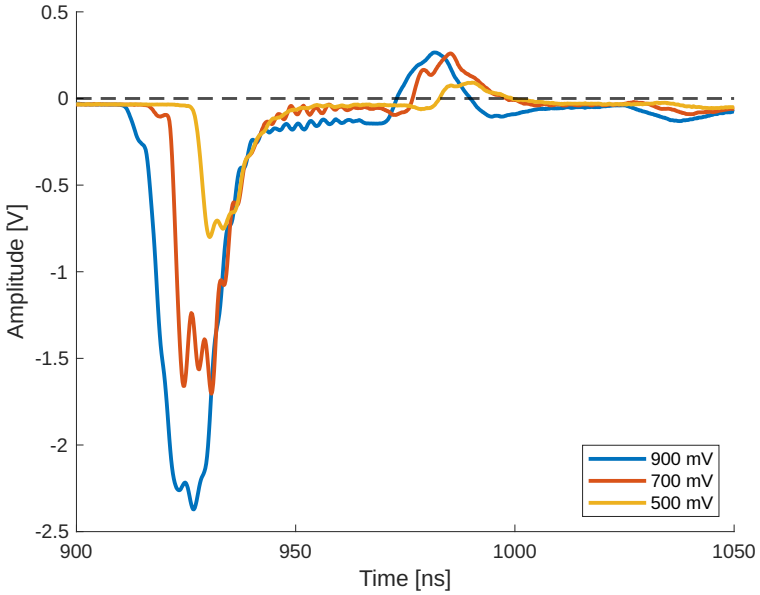



Figure 33: PMT response for multiple gain values using the original cable. As gain decreases, the magnitude of the positive-voltage reflection also decreases, implying that the reflection is due to an impedance discontinuity. Each response exhibits some ringing after 940 ns; however, as gain decreases, the amplitude of the ringing also decreases.

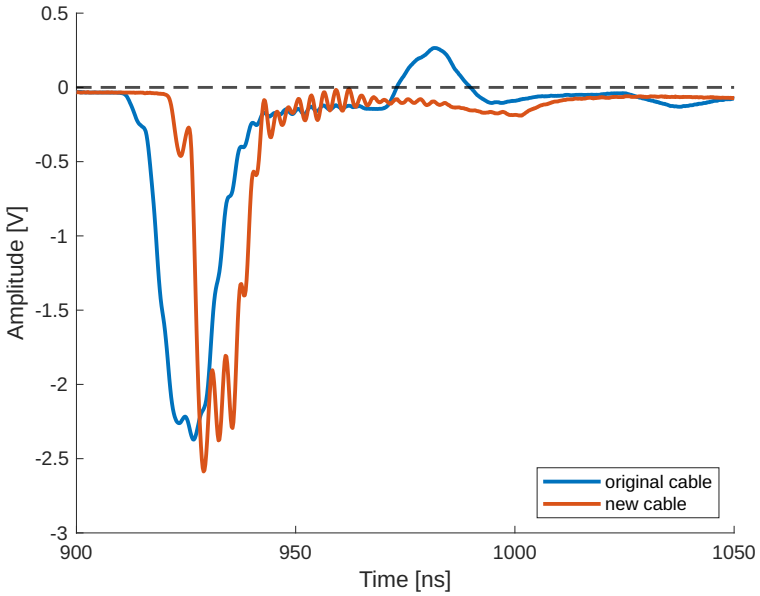



Figure 34: PMT response for two different BNC cables. Using the original cable, the response has a positive-voltage reflection; using a new cable, there is no reflection. The new cable does exhibit some ringing after 940 ns.

due to improper cable management. As will be discussed in the Digitizer Upgrade section, I replaced all BNC cables and connectors with SMA cables and connectors because the new digitizer used SMA connectors.

## Digital PMT Gain Controller

As alluded to in the previous section, the PMT's gain can be changed to control the output current's amplitude. For the particular PMTs used in the wingbeat-modulation lidar [2], the gain is controlled by an external voltage.

In general, we want to maximize the signal-to-noise ratio (SNR) of the PMT output. Setting the gain too low will result in a low SNR, whereas setting the gain too high can result in saturating the PMT output. Since the amount of light collected by the telescope depends on the object type and distance, we want to adjust the PMT gain for each experiment. The existing gain controller, pictured in Figure 35, was not easy to precisely adjust in the field. To remedy this, we created a digital gain controller, which is described in this chapter.

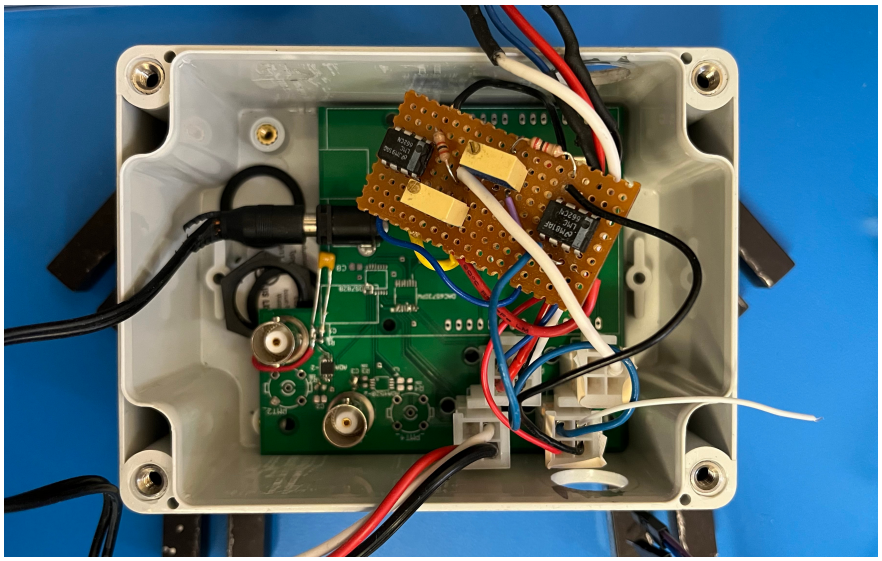


Figure 35: The original gain controller circuit. The gains for the trigger PMT and the output PMT were adjusted with the two potentiometers.

System Architecture

The wingbeat-modulation lidar has two PMTs: one for the lidar's output signal, and one that acts as the trigger input for the computer's digitizer. The digital PMT gain controller was designed to enable software-control of the both PMT gain voltages. The major system requirements were

1. Set gain values from a companion computer connected to the gain controller.

2. Set the gain values of both PMTs separately.

The gain controller has a microcontroller (MCU), a universal asynchronous receiver/transmitter (UART) interface that accepts commands from a computer, and a digital-to-analog converter (DAC) that sets the PMT gain voltages. Figure 36 shows the high-level system architecture.

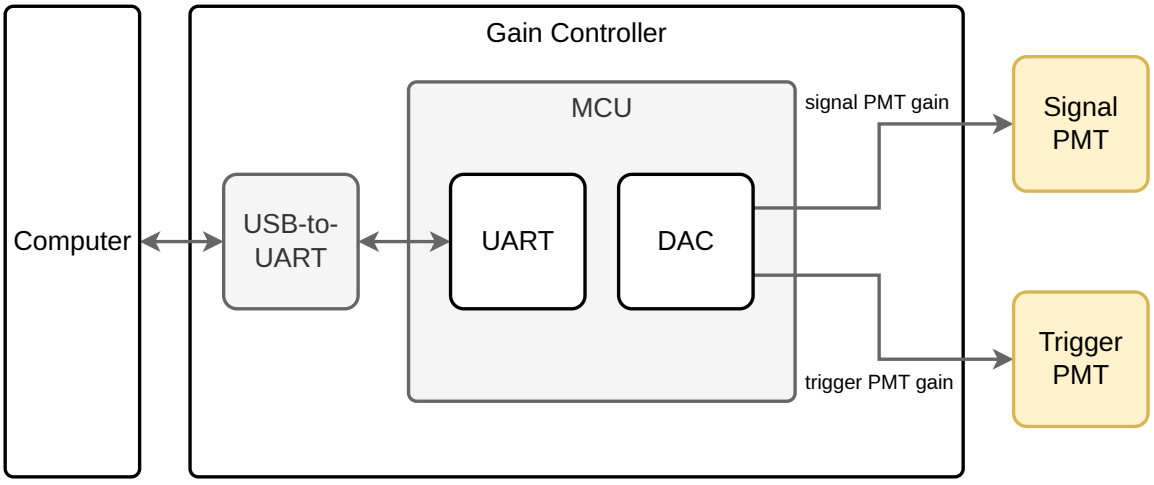



Figure 36: System architecture of the digital gain controller. A computer sends commands to the MCU, which sets the gain values for both PMTs independently.

Hardware Design

Figure 37 shows an overview of the hardware design. The primary components are

- An MSP430 MCU with a built-in DAC.
- A power system to power the PMTs and the MCU.
- A USB-to-UART converter.

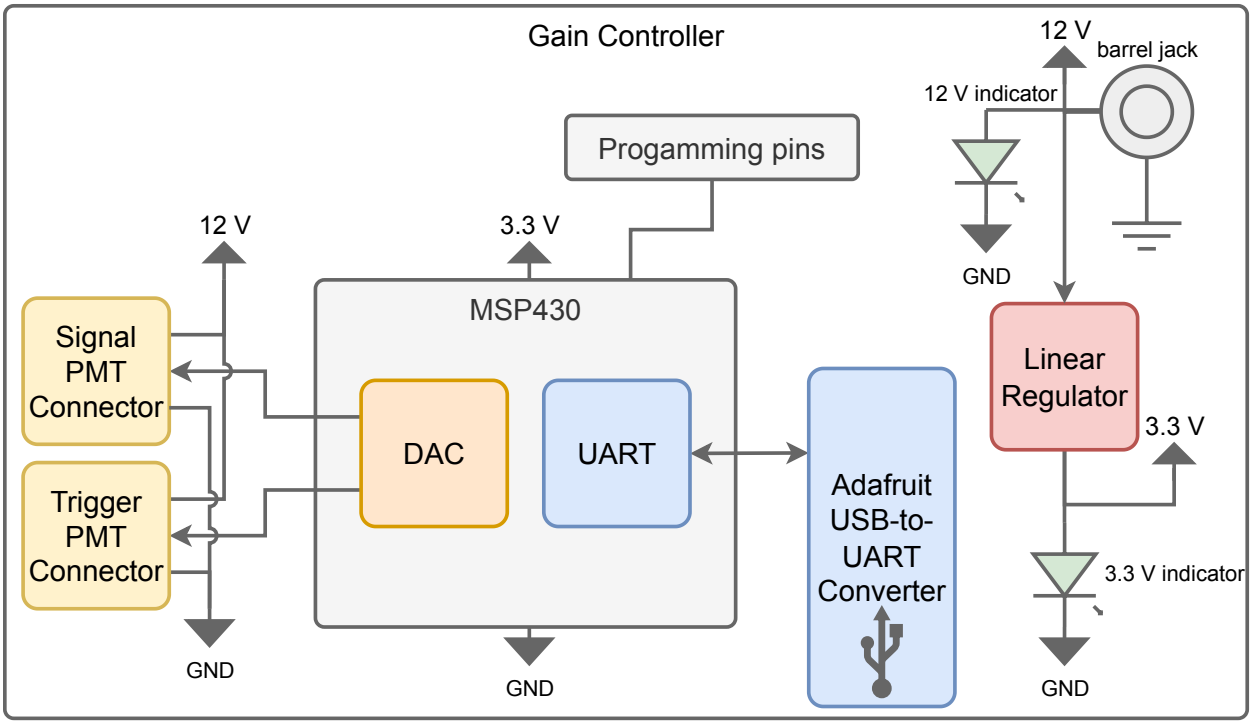



Figure 37: Hardware architecture of the digital gain controller.

The system requirements drove the selection of which MCU to use. Many MCUs have UART interfaces and a built-in DAC; choosing an MCU with a UART and built-in DAC significantly simplified the overall design. Since the system needed to control two PMT gains independently, we wanted an MCU with a built-in DAC with two or more channels.

Since the possible PMT gain values ranged between 100 mV to 900 mV, we needed the DAC to have a resolution of at least ~10 mV. This resolution would allow fine-grained control of the PMT gain.

Given these requirements and our familiarity with the TI MSP430 family of MCUs, we chose an MSP430FR2355, which has 2 UARTs and 4 12-bit DACs. By using the 1.5 V

internal DAC reference, we achieved a resolution of 0.366 mV, which was more than precise enough for our purpose.

For simplicity, we used a commercial-off-the-shelf USB-to-UART converter circuit board from Adafruit instead of designing our own. The Adafruit CP2102N Friend is primarily designed for plugging into a breadboard, so we added female headers to the gain controller circuit board for the CP2102N Friend to plug into. This was far simpler than designing our own circuit based around the CP2012N USB-to-UART converter that the Adafruit board uses.

The power system was the final piece we designed. In keeping with the previous design, we decided to power the PMTs from the gain controller; this simplified the wiring harnesses used to connect the PMTs to the gain controller. Since the PMTs can be powered with anywhere from 11.5 V to 15.5 V, we selected a 12 V AC-to-DC "wall wart" power supply to power the gain controller. The MSP430 uses a nominal supply voltage of 3.3 V. We chose an LD1117AV33 low-dropout linear regulator to create the 3.3 V voltage rail.

After the initial component selection was complete, we selected a Hammond 1591LFLBK enclosure to put the gain controller in. With the hardware architecture and component selection finalized, Nathaniel Sweeney designed the schematic and circuit board. Meanwhile, I created a 3D assembly in Autodesk Fusion 360 to verify that the circuit board dimensions and layout worked with our chosen enclosure. Figure 38 shows the assembled gain controller circuit board and the enclosure. Emma Wadsworth helped cut holes in the enclosure and assemble the system. The hardware design files and documentation can be found on our PMT gain controller GitHub repository.

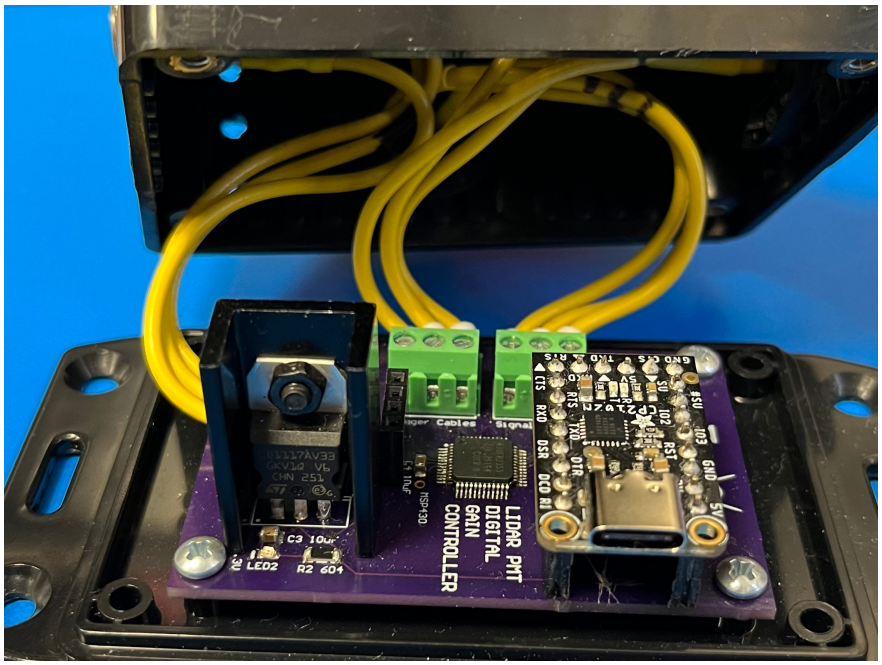



Figure 38: Picture of the assembled gain controller in its enclosure.

Software Design

After the hardware was designed, I developed the MCU software, which is responsible for setting the PMT gain voltages based upon user input. The primary software requirements are listed below. The first two requirements were hard requirements, whereas the third requirement would be helpful, but was not strictly necessary.

1. Accept user input which specifies the PMT voltage and which PMT gain to set (either the trigger PMT or the signal PMT).

2. Set the DAC voltage according to the user's input.

3. Save the DAC voltages in nonvolatile memory so the voltage settings would persist the next time the gain controller is powered on.

Figure 39 depicts the software's functionality as a flowchart. On power up, the program must initialize the MCU and the required peripherals. My initialization routine comprises

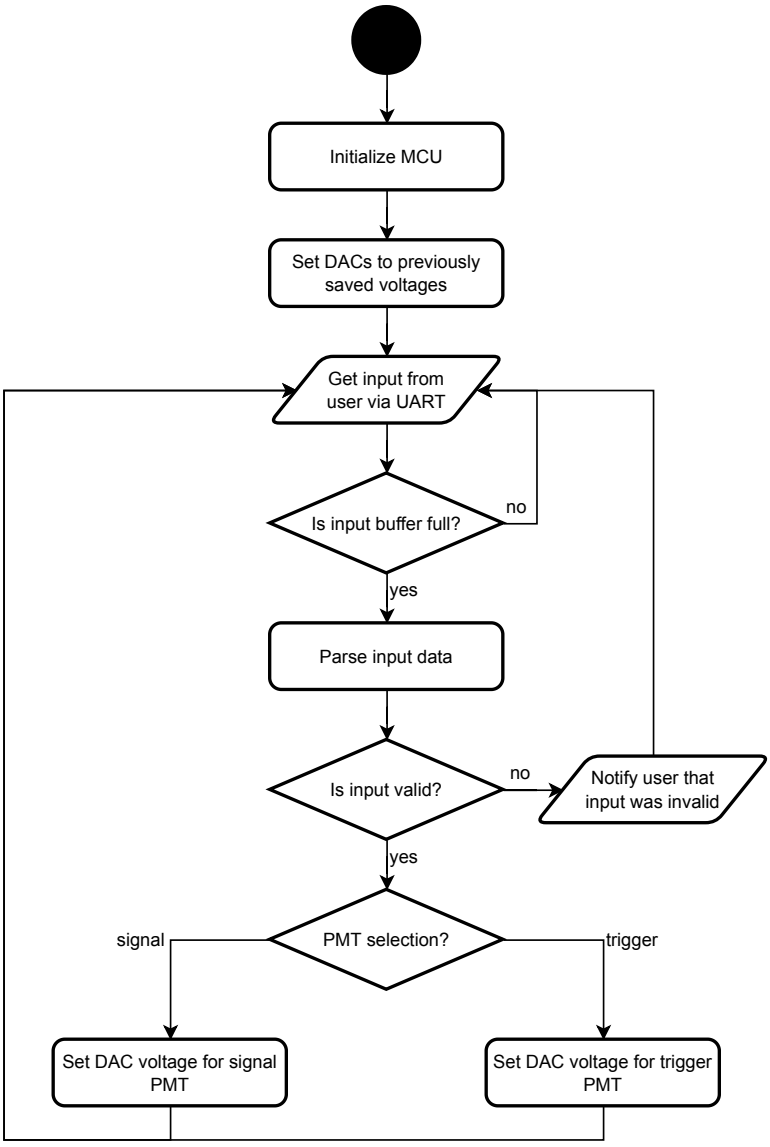



Figure 39: Flowchart of the program running on the MSP430 MCU. On power up, the DACs and UART are initialized, then the DAC voltages are set using values saved from last time the gain controller was used. The program then waits for user input and fills a buffer. If the input is valid, the relevant DAC voltage is set and saved.

configuring the DACs, configuring the UART, configuring unused pins, and disabling the watchdog timer. Once initialization is finished, the DACs are programmed with their values from last time. Then the program enters an infinite loop of receiving user input and setting DAC values. Once a full buffer of user input is received, the input is parsed and validated.

When the input is valid, the relevant DAC voltage is set and saved to nonvolatile memory. This loop repeats until the gain controller in powered off.

Architecture Figure 40 shows an overview of the software components and their responsibilities and relationships. I split the software design into several modules:

- A UART module, which configures the UART hardware and handles receiving user input.
- A Parser module, which parses and validates the user input.
- A DAC module, which is used for configuring the DAC and setting the DAC voltage.
- A PMT configuration module, which specifies PMT names and voltage limits.

These modules were designed to be independent and reusable. The documentation and code found on our PMT gain controller GitHub repository provides more detailed documentation and implementation details.

Communication Protocol To support setting gain values for multiple PMTs, I had to design a message format that could describe the desired PMT and voltage. To keep the format short and simple, a single character at the beginning of the message to indicates which PMT is being selected, then the next 4 characters indicate the voltage in mV. The resulting message looks like `pXXXX`, where `p` is the PMT type (either `s` for signal or `t` for trigger) and `XXXX` is the voltage in mV.

The accepted PMT type characters are defined in the Parser's source code (`parser.c`), and the accepted voltage values for each PMT are defined in `pmt_config.h`.

Usage

This section briefly describes how to use the gain controller. The primary steps are

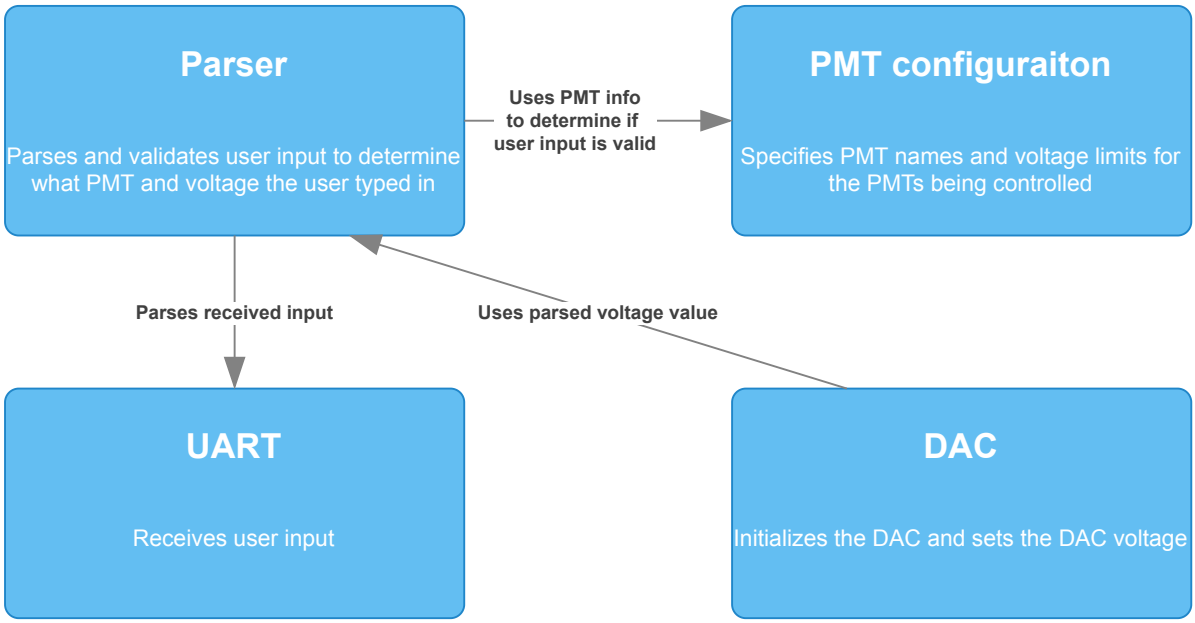



Figure 40: Overview of the software components, their responsibilities, and their relationships.

1. Power on the gain controller.
2. Connect a computer to the USB-C port of the gain controller.
3. Use a terminal emulator (e.g., PuTTY, picocom, etc.) to connect to the gain controller.
4. Set the PMT voltage by sending messages of format `pXXXX`, where `p` is the PMT type (either `s` for signal or `t` for trigger) and `XXXX` is the voltage in mV.

If one only wants to use the lidar and not adjust any gain settings, powering on the gain controller is the only necessary step.

Connecting to the Gain Controller One must use a terminal emulator to connect to the gain controller's UART interface. PuTTY is the most popular terminal emulator available on Windows. Various terminal emulators, such as minicom, picocom, or PuTTY are available on Linux and macOS.

The UART settings are

- 115200 baud rate
- eight data bits
- no parity bits
- one stop bit

Setting Gain Values Once connected to the UART interface, one can set gain values for both PMTs. The message format is `pXXXX`, where `p` is the PMT type (either `s` for signal or `t` for trigger) and `XXXX` is the voltage in mV. For example, to set the signal PMT's gain to 850 mV, one would type `s0850` in their terminal emulator and then hit enter—pressing enter tells the gain controller you are done entering your message. As another example, one could set the trigger PMT's gain to 222 mV by typing `t0222`.

The accepted range of gain values are defined in `pmt_config.h` in the source code. These values are taken from the relevant PMT datasheets. These values must be changed if new PMTs are used.

## Digitizer Upgrade

An analog-to-digital converter (ADC, or digitizer) is used to convert the lidar's analog signal into digital form that can be stored and analyzed on a computer. If the digitizer's sampling rate is greater than the laser's pulse width, the digitizer's sampling rate determines the lidar's range resolution. This can be seen by the lidar range resolution equation in Eq. (12):

$$\Delta d = c\frac{\Delta T}{2} \tag{12}$$

where $\Delta d$ is the range resolution, c is the speed of light, and $\Delta T$ is either the laser's pulse width or the digitizer's sampling period.

The original digitizer was a Gage CompuScope 14200 with a 200 MS/sec sampling frequency (5 ns sampling period). The sampling period of 5 ns is larger than the 0.6 ns laser pulse width, so the digitizer's sampling period was the limiting factor [135]. This sampling rate resulted in a range resolution of approximately 0.75 m. While this resolution was generally sufficient for detecting insects, we wanted finer range resolution for characterizing drone propellers. On the small drones we were interested in, the separation between propellers was much less than 0.75 m: one drone measured 0.081 m between adjacent motors, and the other measured 0.173 m. Consequently, we upgraded to a faster 1 GS/sec digitizer.

The new digitizer is a Gage CSE161G2. In order to use the new digitizer, we had to upgrade to new computer which supported the PCIe version the digitizer uses. With the digitizer's 1 GS/sec sampling frequency (1 ns sampling period), the new effective range resolution is 0.15 m. While this resolution is generally not capable of resolving different motors on small drones, it is the best resolution we could reasonably attain because higher-sampling-rate digitizers are uncommon and prohibitively expensive.

Digitizer Control Software

The software used to control the old digitizer was not compatible with the new digitizer, so I wrote software to control the new digitizer. The software is written in Python and uses Gage's Python SDK to interact with the digitizer. The software is available on GitHub[1].

I created a `wingbeat_lidar` Python package which contains a `digitizer` module and a `range_calibration` module. The `digitizer` module contains the `Digitizer` class, which is used to initialize the digitizer and collect data. The `range_calibration` module has a program user's can run to perform a range calibration, as well as functions that can be used to convert range bin numbers into distances in meters. While developing the modules, I wrote many unit tests to verify the software's functionality.

[1] `https://github.com/BMW-lab-MSU/wingbeat-modulation-lidar-data-acquisition`

The digitizer's configuration is specified in a TOML configuration file, an example of which is shown in Listing 3. The `Digitizer` class reads the configuration file and configures the hardware accordingly. Once the hardware has been configured, the `digitizer` object is used to capture data. Once captured, the data can be saved in an h5 file for long-term storage. Typical usage examples are shown in Listing 1 and Listing 2.

Typical Usage To use the `Digitizer` class, one needs to perform the following basic steps in their code:

1. Instantiate a digitizer object.
2. Initialize the digitizer hardware.
3. Configure the digitizer.
4. Capture data.
5. (optional) Save data.

Before running code, a configuration file needs to be created. Listing 1 shows an example of capturing a single image, and Listing 2 shows an example of capturing multiple images. Listing 3 shows an example TOML configuration file.

```
import numpy as np

from wingbeat_lidar.digitizer import Digitizer
import wingbeat_lidar.range_calibration as rangecal

# Create the digitizer object. The digitizer configuration is assumed to be in config.toml.
digitizer = Digitizer("config.toml")

# Perform basic digitizer hardware initialization.
digitizer.initialize()

# Configure the digitizer with our desired settings from config.toml
digitizer.configure()

# Capture one record of data.
(raw_data,timestamps,capture_time) = digitizer.capture()

# Convert raw ADC numbers to voltage values.
data = digitizer.convert_to_volts(raw_data)

# Compute the range bin distances so we can save that information in the h5 file.
rangecal.load_calibration("calibration.toml")
range_bins = np.arange(digitizer.acquisition_config.SegmentSize)
distance = rangecal.compute_range(range_bins)

# Save the data
data_is_volts = True
with h5py.File("data.hdf5", "w") as h5file:
    digitizer.save_data_in_h5(h5file, data, timestamps, capture_time, data_is_volts, distance)
```

Listing 1: An example of collecting a single record of lidar data. In this example, the raw data is converted into voltage values and the range bins are converted into distances in meters; these conversions are optional.

```
import numpy as np

from wingbeat_lidar.digitizer import Digitizer

# Number of images to collect.
N_IMAGES = 16

# Create the digitizer object. The digitizer configuration is assumed to be in config.toml.
digitizer = Digitizer("config.toml")

# Perform basic digitizer hardware initialization.
digitizer.initialize()

# Configure the digitizer with our desired settings from config.toml
digitizer.configure()

# Get the image size that the digitizer is collecting so we can preallocate
# matrices later for saving the data.
n_samples = digitizer.acquisition_config.SegmentSize
n_segments = digitizer.acquisition_config.SegmentCount

# Preallocate data matrices.
data = np.empty((n_images, n_samples, n_segments))
timestamps = np.empty(shape=(n_images, n_segments))
capture_time = np.empty(shape=n_images, dtype=np.bytes_)

# Collect the data.
for i in range(N_IMAGES):
    (data[i, :, :], timestamps[i, :], capture_time[i]) = digitizer.capture()

# Save the data. We did not convert data into volts or compute the range bin distances,
# so we don't need to use those arguments in save_data_in_h5.
with h5py.File("data.hdf5", "w") as h5file:
    digitizer.save_data_in_h5(h5file, data, timestamps, capture_time)
```

Listing 2: An example of collecting a multiple records of lidar data.

```
[trigger]
condition = "falling"
# Trigger level as a percent of the channel's input range.
level = -10
# Supported values: "external", 1 (for channel 1), 2 (for channel 2)
source = 1

[acquisition]
sample_rate = 1e9
# We use dual acquisition mode because one of the channels needs to be used
# as the trigger source. The trigger voltage is negative, and the external
# trigger input only supports postiive voltage.
mode = "dual"
# The number of samples to acquire per pulse / trigger event.
n_samples = 128
# Trigger delay can be used to delay data acquisition by a number of samples
# after the trigger event. This might be useful if the first handful of samples
# occur before the light has even left the instrument
trigger_delay = 0
# The number of sequential pulses to acquire. Values greater than 1 result in a
# "multiple record acquisition", as it's referred to in the Gage documentation.
segment_count = 1024

[channel]
channel = 2
# Voltage range of the input channel, in mV. The ADC's full input range is +/- 1 V,
range = 2000
# The DC offset is programmable and can span the full scale input range.
dc_offset = -1000
```

Listing 3: Example digitizer configuration file. The settings in this file correspond to settings used during the drone experiments in the Architecture and Implementation chapter.

## Pan-tilt Controller Software

The need for writing software to control pan-tilt mounts was two-fold: the lidar was mounted on a pan-tilt controller, which was used to scan and position the lidar; and the UAVs in the next chapter were mounted a pan-tilt controller. While there was existing code that controlled the lidar's pan-tilt mount, there were several issues. The previous pan-tilt controller code was tightly coupled with the data collection code; consequently, the code was not particularly modular or easy to reuse. Moreover, the code was written in MATLAB, but the new digitizer data collection code was written in Python, which necessitated a rewrite. Given that we now needed to control two separate pan-tilt mounts, writing an independent pan-tilt controller that could easily be used in new data collection and experiment automation software was necessary.

The lidar used a Quickset QPT-130 7-55510 pan-tilt mount, while the UAVs were mounted on a Quickset QPT-20 7-22HSB. These two pan-tilt mounts use similar but slightly different communication protocols. The QPT-130 7-55510 uses the PTHR-90 protocol, while the QPT-20 7-22HSB uses the PTCR-20 protocol. The similarities and differences of these two communication protocols informed my software design.

### Architecture and Implementation

The pan-tilt controller software was split into two modules: a `controller` module and a `protocol` module. The `controller` is responsible for all high-level control of the pan-tilt mount. The `protocol` module handles all the implementation details of the communication protocol itself, that way the `controller` can be used unmodified for pan-tilt mounts with different protocols. Figure 41 shows a class diagram for the `quickset_pan_tilt` package.

The abstract `QuicksetController` class that implements most of the high-level functionality for interacting with the pan-tilt mount. When sending or receiving commands,

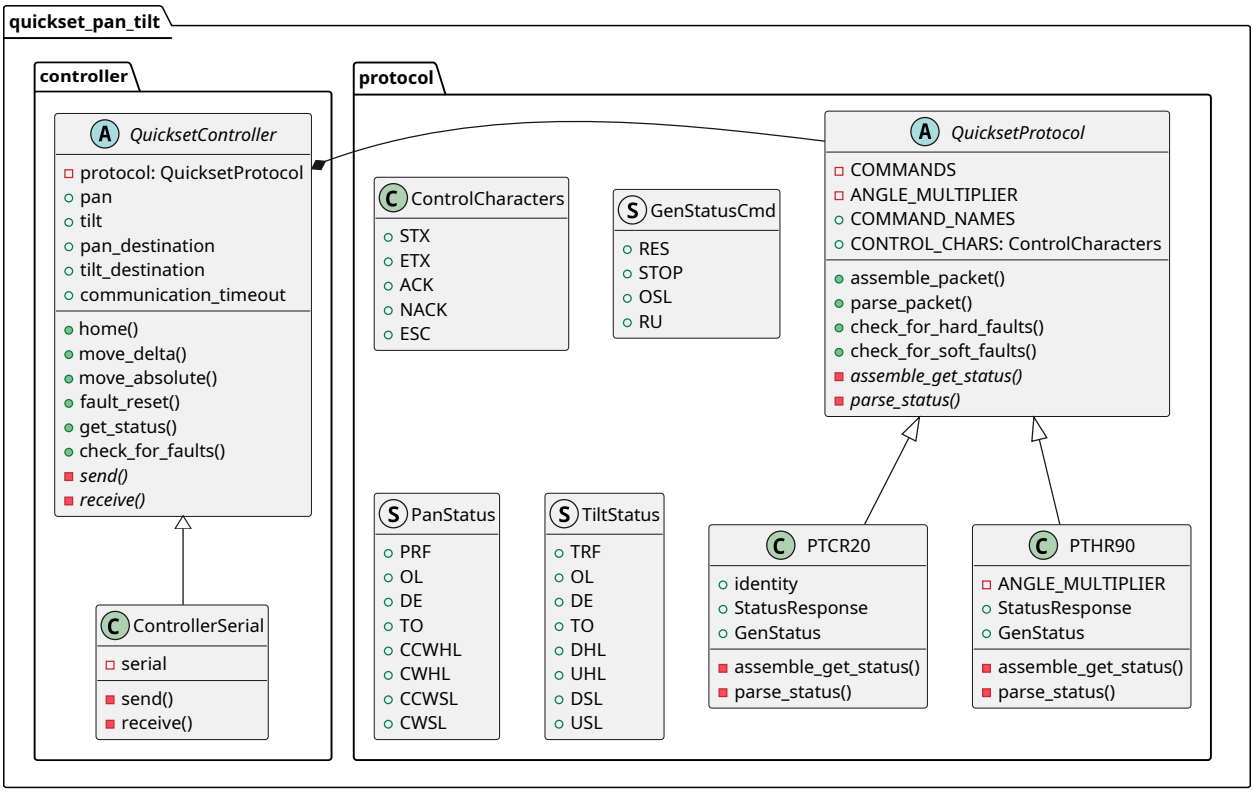



Figure 41: Class diagram for the `quickset_pan_tilt` package. "A" represents an abstract class, "C" represents a concrete class, and "S" represents a struct. Green circles mean the attributes or methods are public, while red squares mean the attributes or methods are private. For simplicity, most private methods are not shown; only private abstract methods and their concrete implementations are shown to illustrate how the subclasses implement parts of the abstract classes. Method arguments are also not shown for simplicity.

the `QuicksetController` class defers to the `QuicksetProtocol` class to create and parse the relevant communication packets. The concrete `ControllerSerial` class implements sending and receiving commands over an RS-232/RS-422 serial link. Some pan-tilt mounts support communicating over Ethernet, so a `ControllerEthernet` class could be implemented to handle sending and receiving commands over Ethernet in the future.

The `protocol` module has an abstract `QuicksetProtocol` class that handles creating and parsing packets. The abstract class handles all the functionality that is common between the PTCR-20 and PTHR-90 protocols, while the concrete `PTCR20` and `PTHR90` classes implement the protocol-specific behaviors. For example, the two protocols use slightly different

status bits, so the concrete classes implement assembling and parsing the status packets; however, both protocols share the same pan status, tilt status, general status command bits, and control characters, so those are implemented as separate data structures in the `protocol` module.

The `QuicksetProtocol` class implements a method dispatch dictionary[2]. This dictionary, shown in Listing 4, contains the command number, packet-assembly method, and packet-parsing method for each supported command. This design pattern allows additional commands to be added without needing to update the `assemble_packet()` and `parse_packet()` methods. Listing 5 shows an excerpt of what the method dispatch looks like, while Listing 6 shows an alternative implementation using case statements. While the method dispatch pattern adds complexity and is not as immediately understandable as a case statement, the pattern makes extending functionality easier.

Listing 7 shows a typical usage example of controlling a pan-tilt mount.

Overall, the architecture creates a loosely coupled system that is easy to extend. Although the controller and protocol are inherently coupled because they need to agree upon what functionality is supported, they generally do not need to know each other's implementation details. The primary exception is that modules need to agree on the command names for each command because the controller passes the desired command name to the protocol object.

[2] `https://blog.alicegoldfuss.com/function-dispatch-tables/`

```
        self._COMMANDS = {
            "get_status": {
                "assemble": self._assemble_get_status,
                "parse": self._parse_status,
                "number": 0x31,
            },
            "move_absolute": {
                "assemble": self._assemble_move_to_entered,
                "parse": self._parse_status,
                "number": 0x33,
            },
```

Listing 4: An excerpt of the protocol method dispatch dictionary.

```
        # Call the command-specific function to prepare the data bytes.
        data_bytes = self._COMMANDS[cmd_name]["assemble"](*data)
```

Listing 5: Implementation of assembling a command packet using method dispatch.

```
match cmd_name:
    case "move_absolute":
        data_bytes = self._assemble_move_to_entered(*data)
    case "move_delta":
        data_bytes = self._assemble_move_to_delta(*data)
    case "get_status":
        data_bytes = self._assemble_get_status(*data)
    case "home":
        data_bytes = None
    case "fault_reset":
        data_bytes = self._assemble_fault_reset(*data)
```

Listing 6: Implementation of assembling a command packet using a case statement.

```
from quickset_pan_tilt import controller, protocol

# Create the pan-tilt controller object
ptc = controller.ControllerSerial("/dev/ttyUSB0", protocol.PTCR20)

# Go to (0,0)
ptc.home()

# Move to -40 degrees pan and 30 degrees tilt
ptc.move_absolute(-40,30)
```

Listing 7: Typical usage of the `quickset_pan_tilt` package.

# UAV DETECTION AND PROPELLER CHARACTERIZATION

## Introduction

Consumer-grade unmanned aerial vehicles (UAVs), commonly known as drones, are becoming increasingly accessible and capable. Unfortunately, the increased prevalence of consumer UAVs has led to an increase in UAV incidents, such as aircraft collisions and near-misses [155, 156]. The rise in incidents has spurred research into UAV detection systems.

Common UAV detection methods include radar [157], visual [158], acoustic [159], passive radio-frequency (RF) [160], and lidar [161]. See the survey article by Marin et al. [162] for a good overview of the different detection modalities. Each modality has distinct advantages and disadvantages, such that a comprehensive UAV detection system will utilize multiple sensor modalities [163–165]. For example, passive RF systems detect drones by analyzing communication channels between the drone and the controller [160, 162]. This method has the advantages of long range, low sensitivity to environmental factors, and being able to detect the UAV's user; however, it cannot detect fully autonomous drones. As another example, radar offers long range and robustness to adverse environmental conditions; however, it's high detection sensitivity often leads to false detections of birds and other small moving objects [166], though significant progress has been made in distinguishing between birds and drones [157, 167, 168].

In addition to simply detecting drones, research is being done to characterize properties like payload size [169, 170], rotor size [171], propeller frequency [172], and number of propellers [173]. The work in this chapter uses MSU's wingbeat-modulation lidar [2] to detect drones and measure their propeller frequencies.

Ciattaglia et al. have measured propeller frequency using FMCW radar and micro-doppler processing techniques at short distances in a lab environment [172, 174, 175]. In their initial study, they needed to wrap the propellers in aluminum tape to increase the

propeller's radar cross-section [172, 174] because the radar used was not designed for objects with small radar cross-sections. In their 2024 study [175], they were able to measure propeller frequency without aluminum tape, but the measurements were taken at 2.61 m, which is a very close range. Rodrigo et al. detected propeller speed on a stationary drone at 500 m using a continuous-wave coherent lidar and micro-doppler processing techniques [176]. In many real-world scenarios, such as detecting drones at airports, detecting the drone long before it reaches the no-fly or hazard zone is necessary. Measuring propeller speed at 500 m proved the real-world feasibility of such measurements. The primary drawback of using FMCW and micro-doppler is that the ability to measure propeller speed is dependent on the angle between the drone and the lidar. As an overhead drone approaches a ground-based lidar, the angle between them approaches 90°, which shifts the speed measurements towards 0 [176]. This results in incorrect speed measurements as the drone approaches the lidar. When the drone is directly overhead, the propeller's distance away from the lidar would be not changing, resulting in an incorrect speed measurement of 0.

In previous work [177], we determined that the wingbeat-modulation lidar could detect propeller frequency when the beam overlaps part of a propeller blade. The work in this chapter builds upon the previous work by covering multiple propellers with the laser beam and taking measurements over many drone tilt angles and throttle values.

## Experiment Prerequisites

### Beam Diverger

From our previous experiments [177], we determined that we can detect propeller frequency when the beam overlaps part of a propeller blade. However, a more realistic drone detection and characterization system would have the lidar beam cover an entire drone. In order to test if our methods can detect the frequency of multiple propellers, we had to design a beam diverger that allowed us to vary the beam's fill factor. With the beam diverger, we

were able to cover anywhere from one propeller to an entire drone with the lidar beam.

The primary constraints when designing the beam diverger were the telescope's small field of view (FOV), the length of available space in the lab, and the size of the drones. Ultimately, we needed to ensure that the entire drone fit into the telescope's FOV at the desired distance. For this, we needed to calculate the telescope's FOV diameter as a function of distance. Figure 42 shows the geometry of the telescope's FOV, and Eq. (13) shows the equation; $\theta_{telescope}$ is the telescope's half-angle divergence, $d_{telescope}$ is the telescope's diameter, $x$ is the horizontal distance away from the telescope, and $d_{telescopeFOV,x}$ is the diameter of the telescope's FOV at distance $x$.

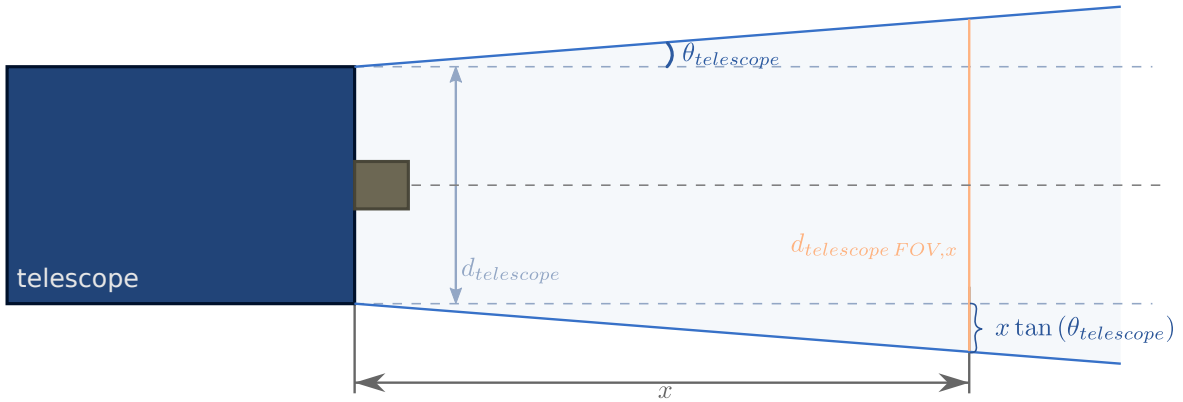



Figure 42: Geometry used to derive the telescope's FOV diameter as a function of distance.

$$d_{telescope\,FOV,x} = d_{telescope} + 2 \times (x \tan(\theta_{telescope})) \tag{13}$$

The telescope has the following half-angle divergences [135]: 0.911 mrad (horizontal), 3.20 mrad (vertical), and 3.33 mrad (diagonal). These were calculated with the following equation:

$$\theta_{telescope} = \arctan\left(\frac{h}{EP_{location}}\right), \tag{14}$$

where $h$ is 1/2 of the relevant PMT dimension, and $EP_{location}$ is how far away the entrance

pupil is from the system's focus. The PMT sensor dimensions are 3.7 mm by 13.0 mm (referred to as horizontal and vertical, respectively). Which dimension, and thus FOV divergence angle, is actually horizontal or vertical depends on the orientation of the PMT itself, which is adjustable. In practice, we are limited by the telescope's smallest FOV (0.911 mrad): when a drone is oriented perpendicular, the horizontal and vertical extent of the drone will be the same. Thus, we use 0.911 mrad for $\theta_{telescope}$.

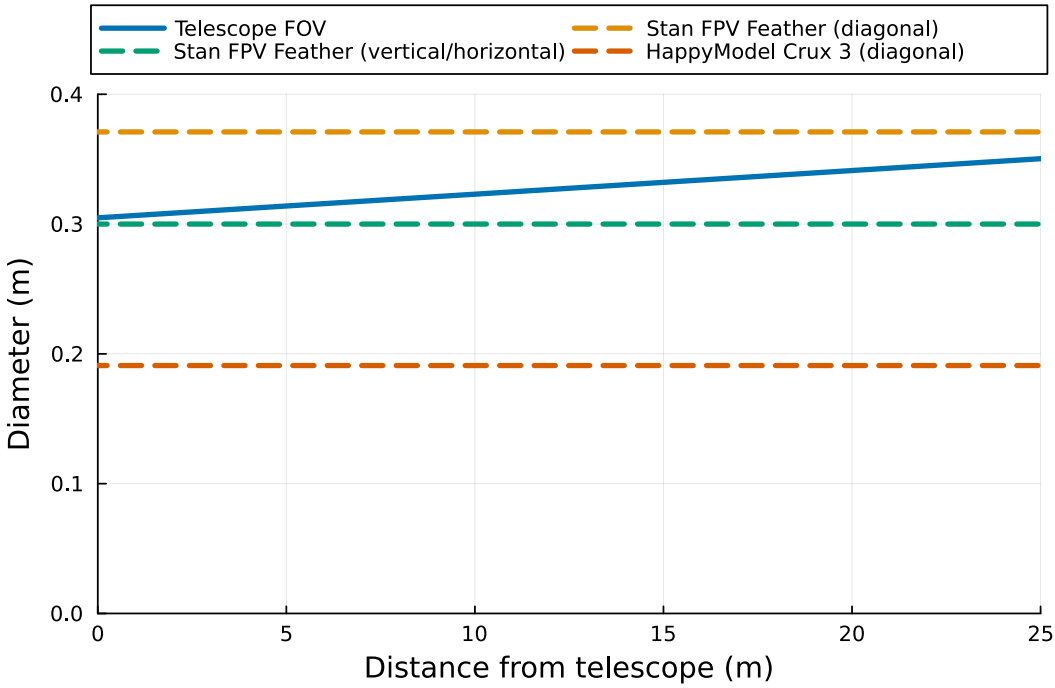



Figure 43: Plot of the telescope's FOV diameter and the UAV dimensions. The diagonal dimensions are measured across the diagonal of the drone body, between the tips of the propellers; this is the largest possible dimension.

Figure 43 shows the telescope's FOV diameter as a function of distance, along with the dimensions of the two drones we used. The small HappyModel Crux 3 drone fit entirely within the telescope's diameter. The larger Stan FPV Feather partially fit within the telescope's FOV; the square horizontal/vertical dimension of the drone fit within the telescope's FOV at all distances, but the diagonal dimension did not quite fit within the telescope's FOV at the distances we had available in the lab. This means most of the drone would fit in the

telescope's FOV, save for a small portion of the propellers.

Knowing the telescope's FOV diameter and the drone dimensions, I moved onto designing the optics that would expand the beam. We wanted the beam width large enough to encompass the drone while not being larger than the telescope's FOV diameter. Figure 44 shows the geometry used to derive the distance where the telescope FOV and beam diameters intersect.

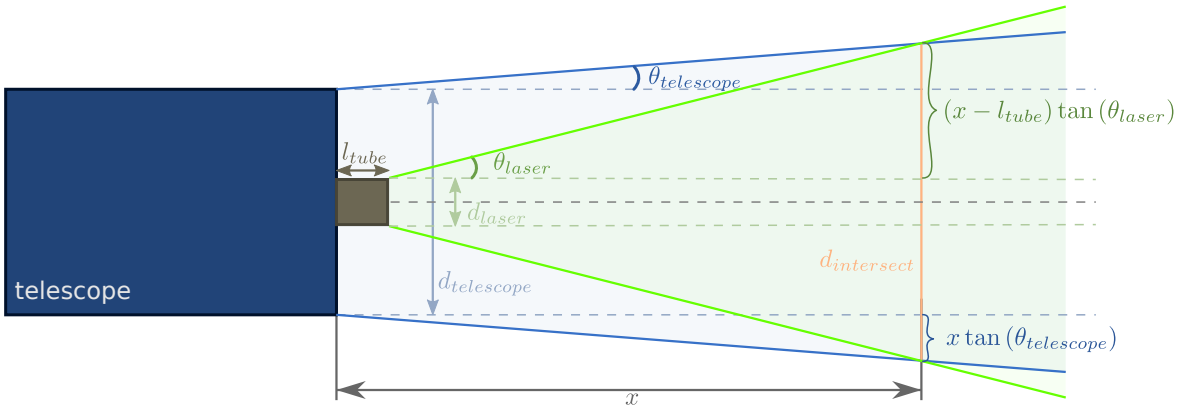



Figure 44: Geometry used to derive the distance where the telescope FOV and beam diameters intersect.

The beam width as a function of distance away from the telescope, $x$, is given by

$$W_{beam} = d_{laser} + 2(x - l_{tube}) \tan(\theta_{laser}), \tag{15}$$

where $d_{laser}$ is the laser's output diameter, $l_{tube}$ is the length of the lens tube (approximated as the distance between the lenses in the lens tube), and $\theta_{laser}$ is the laser's half-angle output divergence. Rather than deriving an exact distance where the beam and telescope FOV fully overlap, I iteratively designed the beam diverger optics by varying design parameters and plotting the corresponding beam width and telescope FOV diameter.

Figure 45 shows the optical design of the beam diverger. The input beam radius and divergence, $y_{in}$ and $\theta_{in}$, are 25.4 mm and 0.2 mrad [135]. The ultimate goal is to have an output radius and divergence, $y_{out}$ and $\theta_{out}$, that result in the desired beam width at the

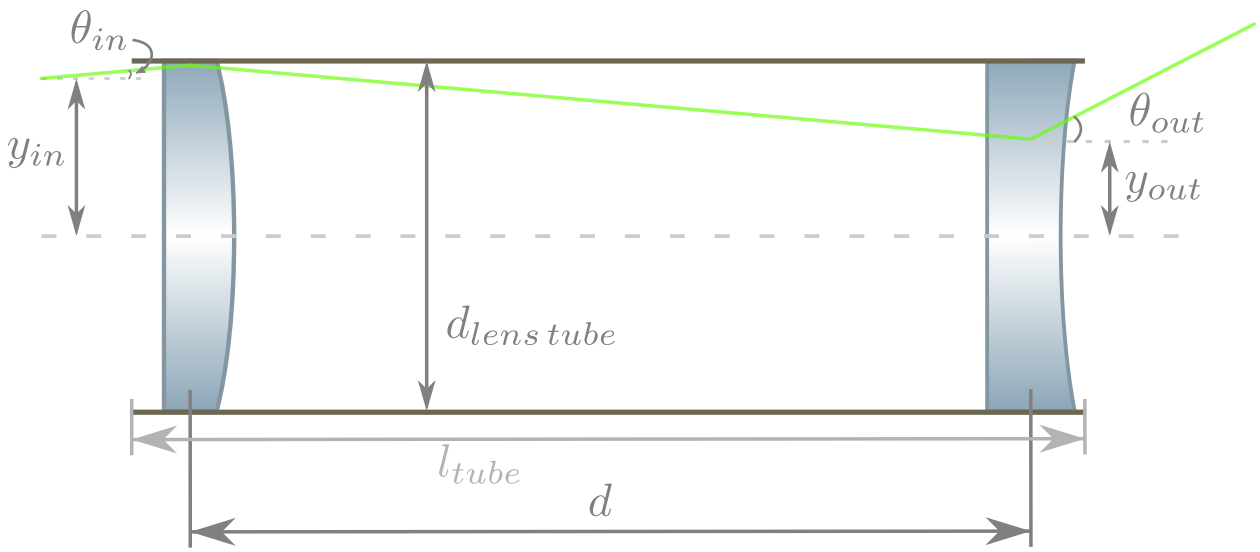



Figure 45: Optical schematic of the beam diverger system.

desired distance. The design parameters are the distance between lenses, $d$, and the focal lengths of the converging and diverging lenses, $f_c$ and $f_d$. The ray transfer matrices for the system above are

$$M_{converging} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{f_c} & 1 \end{bmatrix} \qquad M_{tube} = \begin{bmatrix} 1 & d \\ 0 & 1 \end{bmatrix} \qquad M_{diverging} = \begin{bmatrix} 1 & 0 \\ \frac{-1}{f_d} & 1 \end{bmatrix}.$$

The ray transfer matrix for the whole system then becomes

$$M = M_{diverging} \cdot M_{tube} \cdot M_{converging} = \begin{bmatrix} \left(1 - \frac{d}{f_c}\right) & d \\ \left(\frac{-1}{f_d} - \frac{1}{f_c}\left(1 - \frac{d}{f_d}\right)\right) & \left(1 - \frac{d}{f_d}\right) \end{bmatrix}. \tag{16}$$

Finally, the relations between the input radius/divergence and output radius/divergence are

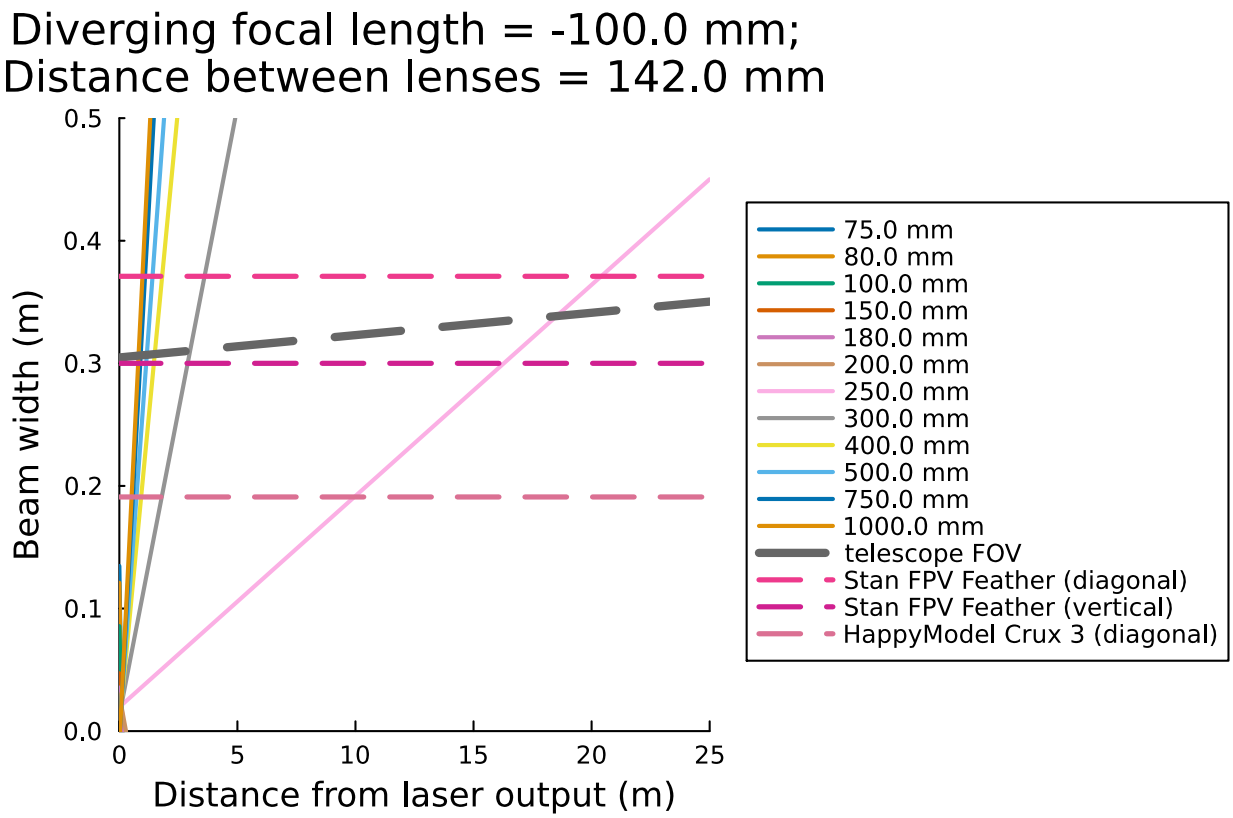



Figure 46: Beam width as a function of distance for different converging lens focal lengths.

given by

$$\begin{bmatrix} r_{out} \\ \\ \theta_{out} \end{bmatrix} = M \cdot \begin{bmatrix} r_{in} \\ \\ \theta_{in} \end{bmatrix} \tag{17}$$

$$r_{out} = \left(1 - \frac{d}{f_c}\right) r_{in} + d\theta_{in} \tag{18}$$

$$\theta_{out} = \left(\frac{-1}{f_d} - \frac{1}{f_c}\left(1 - \frac{d}{f_d}\right)\right) r_{in} + \left(1 - \frac{d}{f_d}\right) \theta_{in}. \tag{19}$$

By substituting Eq. (18) and Eq. (19) into Eq. (15), I was able to calculate beam width as a function of the beam diverger design parameters and the distance away from the lidar.

Using an interactive version of the plot in Fig. 46, I performed a parameter sweep

to determine which lenses and lens tube lengths to use. The final design uses a 100 mm focal-length plano-concave lens (Thorlabs LC1093-A), a 250 mm focal-length plano-convex lens (Newport KPX202-C), and a lens tube system comprising a 76.2 mm lens tube, followed by a 50.4 mm tube, followed by a 38.1 mm adjustable tube. This design allowed us to achieve all of our desired beam widths by varying the lens tub length.

Pan-tilt Controller

In our previous work [177], we measured propeller frequency at two different tilt angles: 0° and 90°. To more fully characterize the wingbeat-modulation lidar's ability to detect propeller frequency as a function of angle, we needed a way to easily control the tilt angle of the UAV relative to the lidar. To control the UAV's tilt angle, we mounted the UAV on a small software-controlled pan-tilt mount.

We chose a Quickset QPT-20 7-22HSB pan-tilt positioner. This pan-tilt mount has a payload capacity of approximately 9 kg, which is more than enough to hold a small UAV; this payload capacity is also sufficient for other small payloads that MSU's Optical Remote Sensor Laboratory may use in the future. It has a pan range of 360° and a tilt range from −90° to 90°. The unit can be controlled over RS232, RS422, or Ethernet using Quickset's custom PTCR-20 communication protocol. We made a long custom USB-to-RS232 cable that connects to the pan-tilt controller's circular DIN connector; the USB cable has a weatherproof inline connector that allows the USB cable to be disconnected from the pan-tilt controller's wiring harness during transport and storage.

To keep the lidar aligned with the UAV across all tilt angles, we mounted the UAVs such that their front motors were in line with the pan-tilt mount's rotation axis. Wyatt Weller designed the mounting brackets, which we 3D-printed. Fig. 47 shows a small UAV mounted to the pan-tilt mount, and Fig. 48 shows a custom-built Stan FPV Feather UAV mounted to the pan-tilt mount. The pan-tilt mount had to be attached to a wood board and weighed

down to prevent the Stan FPV Feather from moving the pan-tilt mount.

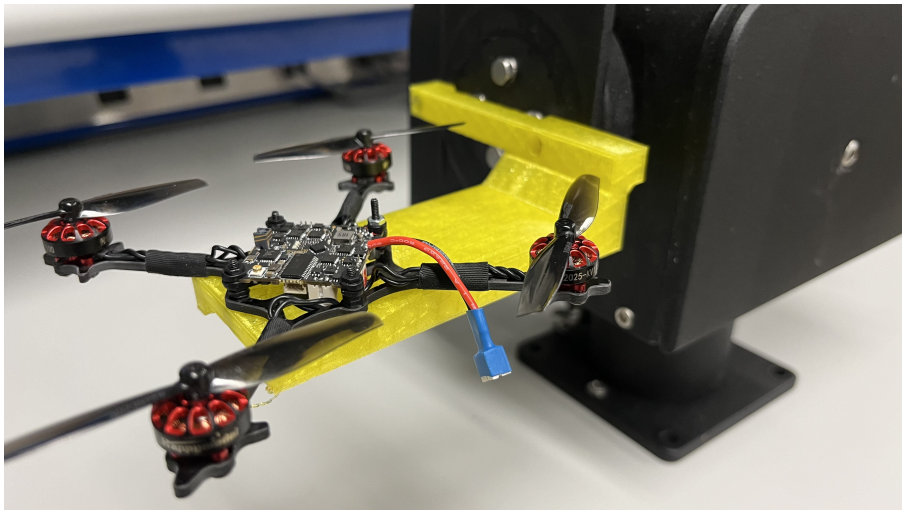


Figure 47: A Happymodel Crux3 UAV mounted to the pan-tilt controller.

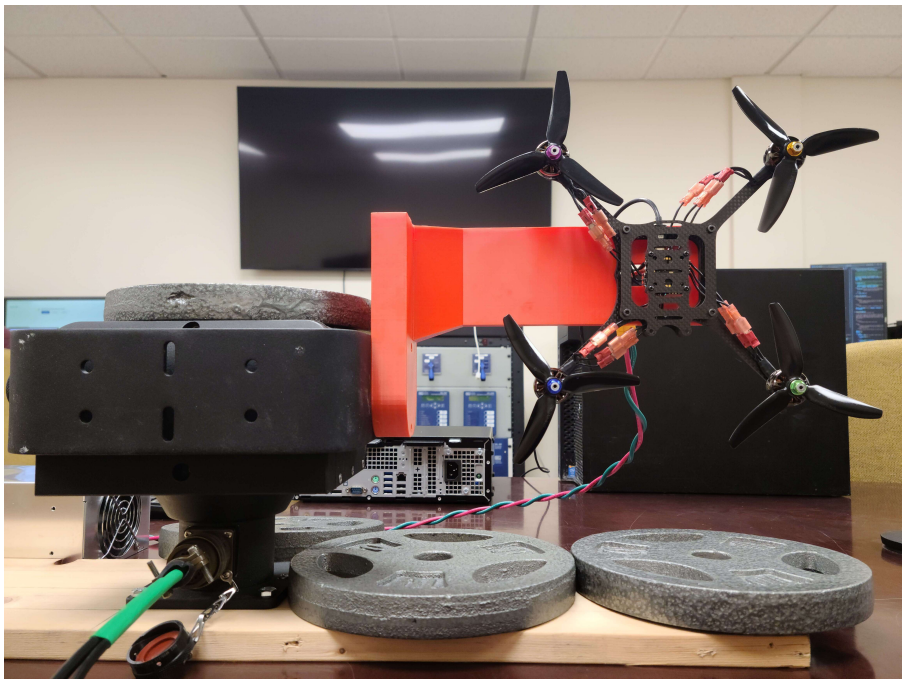


Figure 48: A Stan FPV Feather UAV mounted to the pan-tilt controller. The pan-tilt controller is set to a 90° tilt angle. Notice that the front propellers are in line with the center of the pan-tilt mount.

Drone Control Software

To enable precise motor control, collect ground truth data, and facilitate experiment automation, we needed to control each UAV motor manually via software instead of using a typical UAV controller. The UAVs we used run the Betaflight open-source firmware[1], which allows controlling the UAV over a serial port. This allowed us to manually control the drone via custom software.

Betaflight supports the MultiWii Serial Protocol (MSP), which supports both controlling the UAV and receiving telemetry data from the UAV's flight controller. We used the YAMSPy [178] python library to send/receive MSP messages to/from the UAV. YAMSPy allowed us to individually control each motor's throttle. Listing 8 shows an example of using YAMSPy to set all four motors to 50% throttle.

```
from yamspy import MSPy

# Connect to the drone's flight controller
board = MSPy("/dev/ttyACM0")
board.connect()

# Set all four motor throttles to 50\%
board.send_RAW_MOTORS([1500,1500,1500,1500,0,0,0,0])
```

Listing 8: A minimal example of setting motor throttle values using the YAMSPy library.

The version of MSP used by Betaflight supports collecting motor RPM telemetry data from the UAV's flight controller. However, YAMSPy did not support collecting RPM telemetry data. Based upon Betaflight Configurator's implementation of RPM telemetry collection[2], I added support RPM telemetry support to YAMSPy (as of this writing, RPM telemetry support has not been merged into YAMSPy, so it is only available in our fork[3]). Listing 9 shows an example of getting RPM telemetry data using YAMSPy.

[1] https://betaflight.com/

[2] https://github.com/betaflight/betaflight-configurator/blob/dd16cb767989ecfefe76d6779969466a53d5bd63/src/js/msp/MSPHelper.js#L273

[3] https://github.com/BMW-lab-MSU/YAMSPy

```
from yamspy import MSPy
from yamspy.msp_codes import MSPCodes

# Connect to the drone's flight controller
board = MSPy("/dev/ttyACM0")
board.connect()

# Tell the flight controller to send us motor telemetry data
board.send_RAW_msg(MSPCodes["MSP_MOTOR_TELEMETRY"])

# Receive the raw communication packet from the flight controller
recv = board.receive_msg(delete_buffer=True)

# Process the raw communication packet and extract the RPM data
if recv['dataView']:
    board.process_recv_data(recv)
    rpm = np.array(board.MOTOR_TELEMETRY_DATA["rpm"][0:4])
```

Listing 9: A minimal example of getting motor RPM values using the YAMSPy library.

To make controlling the UAV motors and collecting motor RPM data easier, I wrote a `motor_control` library[4] that wraps YAMSPy and provides convenience functions for controlling the UAV motors and collecting rpm data. Listing 10 shows a minimal example of connecting to the UAV, setting motor throttles, and collecting motor RPM data.

```
import motor_control

# Connect to the flight controller
motor_control.connect("/dev/ttyACM0")

# Set the back motors to 50\% throttle and the front motors to 80\%
motor_control.set_throttle([50, 80, 50, 80])

rpm = get_rpm_telemetry()
```

Listing 10: A minimal example of getting motor RPM values using the YAMSPy library.

Setting the motor throttle values using `motor_control` has a few advantages over using YAMSPy directly. `motor_control.set_throttle` uses throttle values between 0 and 100 instead of 1000 and 2000. Additionally, `motor_control.set_throttle` ramps the motor throttles to their desired value instead of setting the throttle immediately. Ramping the motors reduces inrush current. Large inrush currents should be avoided, and such large

[4] `https://github.com/BMW-lab-MSU/drone-lidar-experiments/blob/main/motor-control/src/motor_control/motor_control.py`

current spikes often tripped our power supply's protection circuitry during development.

The `motor_control` library supports collecting RPM data over a period of time and returning an average RPM. The `collect_rpm_data` function is designed to run in a separate process using the `multiprocessing` library. For our experiments, using multiple processes allowed the RPM telemetry collection to happen while the digitizer was collecting data. This allowed us to calculate average RPM values for motor during the time period when data was being collected. We were then able to use these RPM values as our ground truth data to validate the propeller frequencies detected by the lidar.

## Data Collection

### Data Collection Procedure

The general experiment setup is shown in Fig. 49, where the UAV was mounted as discussed in the Pan-tilt Controller section. The data were collected during the summer of 2024 in the Optical Remote Sensor Laboratory's lab. For this experiment, the lidar and UAV were 15.7 m apart; this distance was limited by our available indoor space.

Data were collected using the five different configurations shown in Fig. 50. These configurations allowed us to test the effect of beam size and our ability to detect multiple

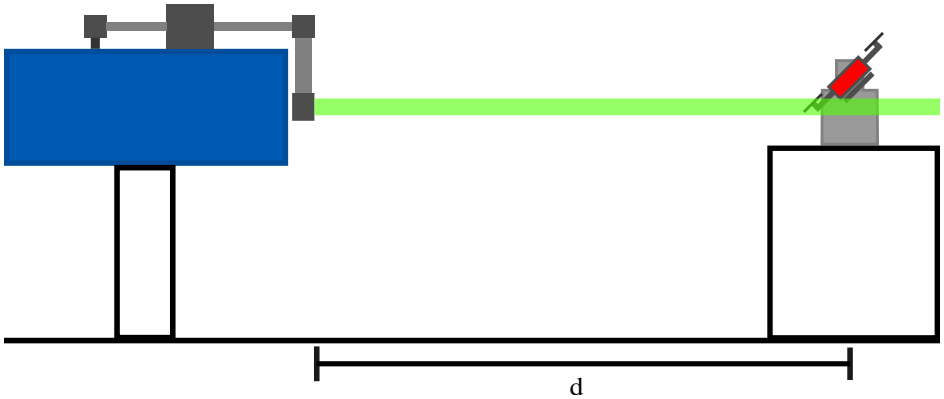



Figure 49: Diagram of the experiment setup. The lidar was a distance, $d$, away from the drone, which was mounted on a pan-tilt mount.

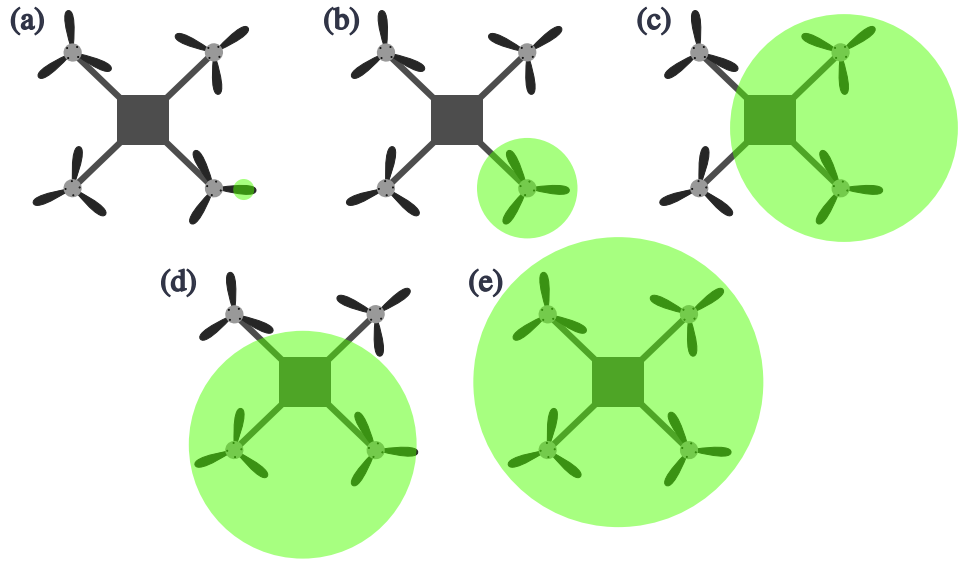



Figure 50: The five different beam configurations: **(a)** intersecting one propeller blade; **(b)** encompassing an entire propeller; **(c)** encompassing the front-right and back-right propellers; **(d)** encompassing the front-left and front-right propellers; **(e)** encompassing the entire UAV.

propeller frequencies. For each configuration, data were collected at angles between 0° and 90° in 10° increments. At each angle, motor throttles were set in 10% increments between 10% and 100%; combinations of throttles (e.g., 40% for the rear motors and 70% for the front motors) were also tested. Thirty-two images were collected for each parameter combination. Ground truth RPM data were collected from the UAV's flight controller to verify our method's accuracy; Motor RPM was converted into an equivalent frequency as seen by the lidar with the following equation:

$$f\,[\text{Hz}] = \frac{RPM_{motor}\,[\text{RPM}]}{60\,[\text{s/min}]} \cdot N_{blades}, \tag{20}$$

where $N_{blades}$ is the number of blades on the propeller.

Initially, data were collected using both the Happymodel Crux 3 and Stan FPV Feather UAVs mentioned in the Pan-tilt Controller section, but hardware issues caused us to not be able to complete data collection with the Happymodel Crux 3; consequently, only the Stan FPV Feather data was analyzed.

Data Collection Software

To efficiently collect data, we needed software that could control the pan-tilt mount, control the UAV's motors, collect ground truth RPM data, and collect the lidar data. Utilizing the digitizer, pan-tilt controller, and motor control libraries, we were able to effectively automate the majority of the data collection. The only thing we could not automate was adjusting the beam expander to change the beam size. Figure 51 shows a high-level flowchart of how the data collection software works.

The data collection software needed to configure the drone for the desired parameters (i.e., tilt angle and throttle values), collect and save the lidar data and ground truth RPM data, and save associated metadata for each set of parameters. We created a spreadsheet that contained all sets of experiment parameters we wanted to run, along with empty columns to save ground truth data and metadata such as the filename where the data were saved. The software read this spreadsheet and collected data for each set of experiment parameters (i.e., each row in the spreadsheet).

While developing the data collection and UAV control software, the UAVs often shut down at seemingly random intervals. As of this writing, we are still unsure about why this happened; we suspect it had to do with how we were manually controlling the UAV motors, which may have triggered a safety feature in the UAV's firmware. After multiple weeks of debugging, we decided to make the data collection code robust against the UAVs dying during the middle of data collection. The software was written to skip any rows in the spreadsheet that already had ground truth data in them.

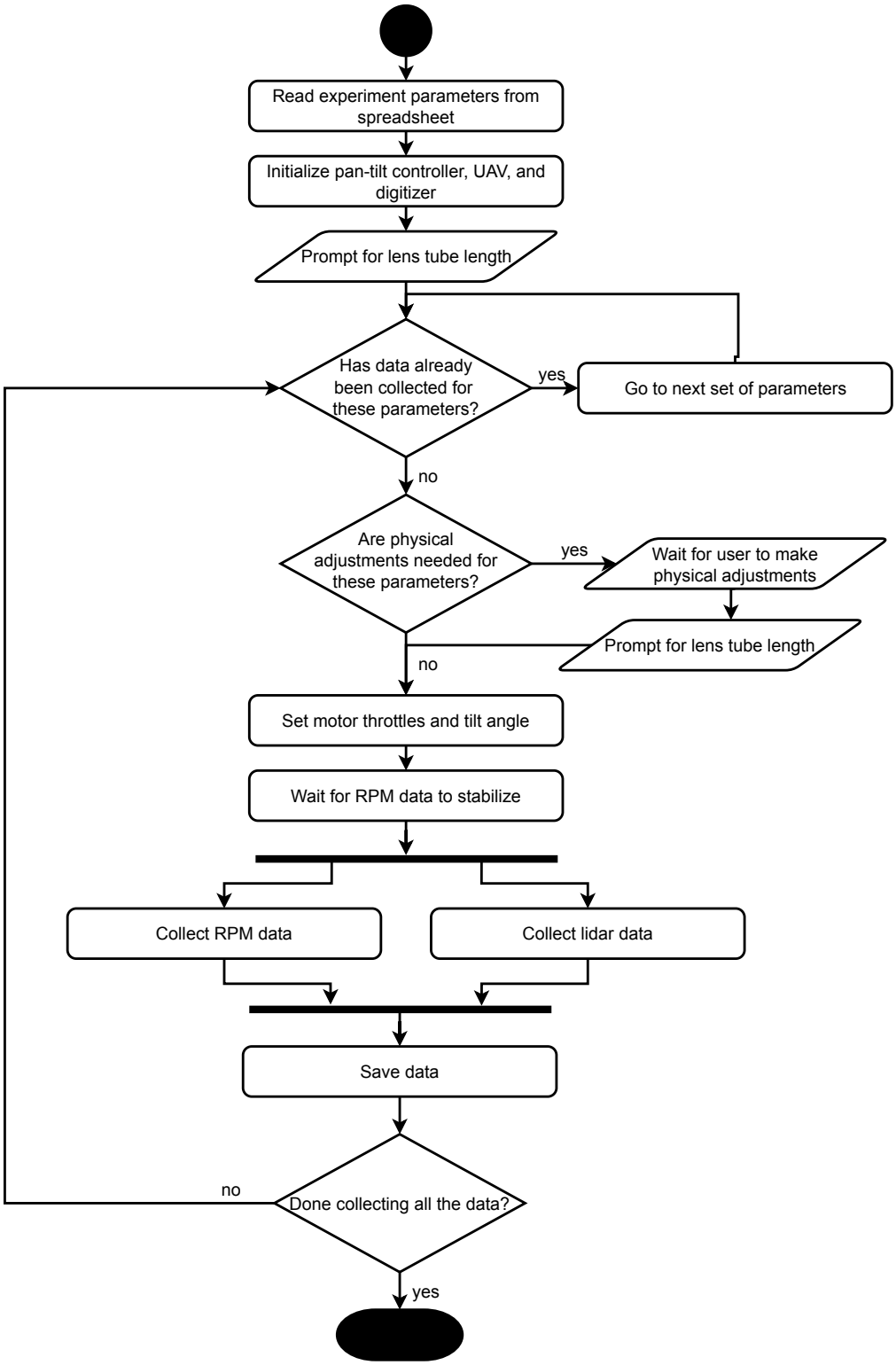



Figure 51: Data collection flowchart. Parallelograms represent user input or intervention. The black bars represent a fork and a join of parallel tasks.

To ensure the ground truth RPM data corresponded to the lidar data, the RPM data and lidar data were collected concurrently using multiple processes. Python's multithreading library does not allow two threads to run concurrently due to Python's global interpreter lock, which only allows one thread to control the Python interpreter. Consequently, multithreading code in Python is actually just context-switching between threads. To actually use multiple CPU cores concurrently, one must use multiple processes. The RPM data and lidar data collection routines are likely I/O-bound: the RPM data collection process sends commands and collects RPM data over a serial port in a loop; the lidar data collection process polls the digitizer status until the digitizer is done collecting data. Given that these processes are I/O-bound, context-switching may have been sufficient. However, using multiple processes ensures that both routines are truly running concurrently (unless the operating system's scheduler decides it needs to context-switch on one of those cores, which is unlikely unless the computer is being used for many other things during data collection).

Overall, the software I designed allowed us to easily configure the experiment parameters and automate the majority of the data collection process. The UAVs dying during the middle of data collection slowed down the data collection process, but the process was still much faster than it would have been without automation. We collected data for approximately 1800 sets of experiment parameters. Even with the automation, the data collection process still took between one and two weeks.

Dataset Description

As mentioned previously, only data from the Stan FPV Feather UAV was analyzed due to hardware issues that prevented all parameter combinations from being collected for the Happymodel Crux 3. The dataset comprises 892 h5 files, each of which has a unique parameter combination (i.e., tilt angle, motor throttles, and beam configuration). Thirty-two images were taken at each configuration, totaling 28544 images. The UAV was present in

each image, though was not always visible. Each image was originally 256 by 1024 pixels, but the number of rows was reduced to 150 because some reflections were present in the data after the wall that was located around the $146^{\text{th}}$ range bin. The total number of rows was 4 281 600; rows were used as data points in $\mathbb{R}^{1024}$.

The data were split into training, validation, and testing sets with 60% for training, 20% for validation, and 20% for testing. The data were split randomly by file name. Since each file contained 32 images of the same configuration, each image could not be considered independent. Splitting by parameter combination helped make the training, validation, and testing sets disjoint. The rows in each image were manually labeled via inspection as "UAV", "nothing", or "wall".

Figure 52 shows a lidar image with a clearly visible propeller signal around 15.7 m. Figure 53 shows example time-domain and frequency-domain plots of a propeller signal.

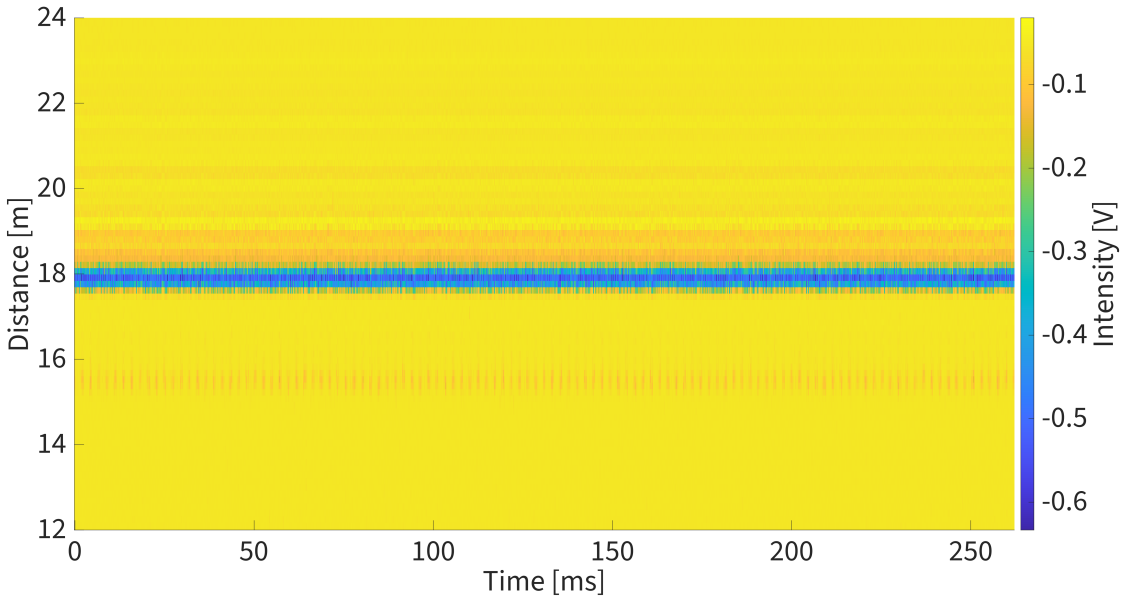



Figure 52: Lidar image with a visible UAV propeller near 15.7 m. The bright blue signal around 18 m is the return from the wall behind the UAV.

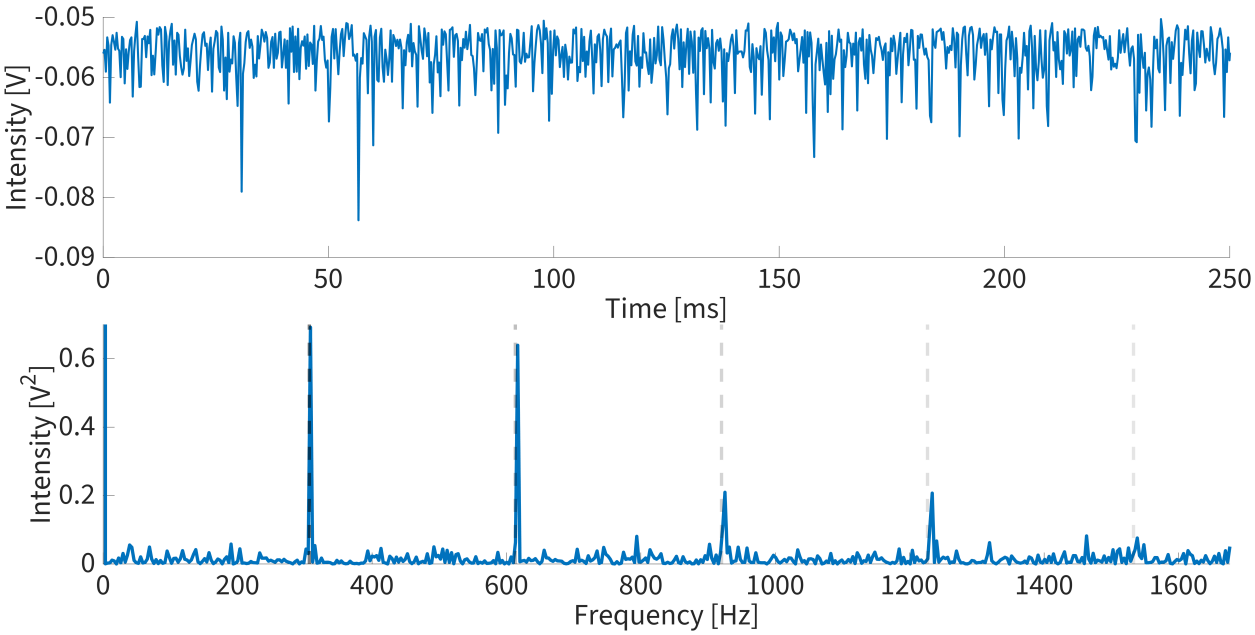



Figure 53: Time-domain and frequency-domain plots of a UAV propeller signal. The ground truth propeller frequency and the associated harmonics are indicated with vertical dashed lines in the spectrum.

## Data Analysis Methodology

### Machine Learning

Similar to Chapter 3, supervised classification was used to predict which rows in the lidar images had a UAV in them. Additionally, the importance of each feature was ranked by computing the feature's mutual information with the class labels.

Overall, the methodology was very similar to the Supervised Methods section of Chapter 3. The exceptions are

- Image-based results were not computed because each lidar image had a UAV in it; consequently, two-dimensional CNNs were not used.
- The oversampling ratios were reduced to $\{0, 1, 10\}$ instead of $\{0, 1, 10, 100\}$ because the class imbalance was less pronounced (due to there being a UAV in each image).

For full details of the methodology, please refer to the Supervised Methods section of Chapter 3.

Visual Inspection

We visually inspected images and spectra from each parameter combination to determine whether propeller frequencies were identifiable in the data. We plotted spectra of multiple rows around the UAV's location in the image; the ground-truth frequency information was overlaid on the spectra. Based upon visual inspection of agreement between the ground truth and measured data, we recorded a "yes", "maybe", or "no". Examples of "yes", "maybe", and "no" spectra are shown in Fig. 54.

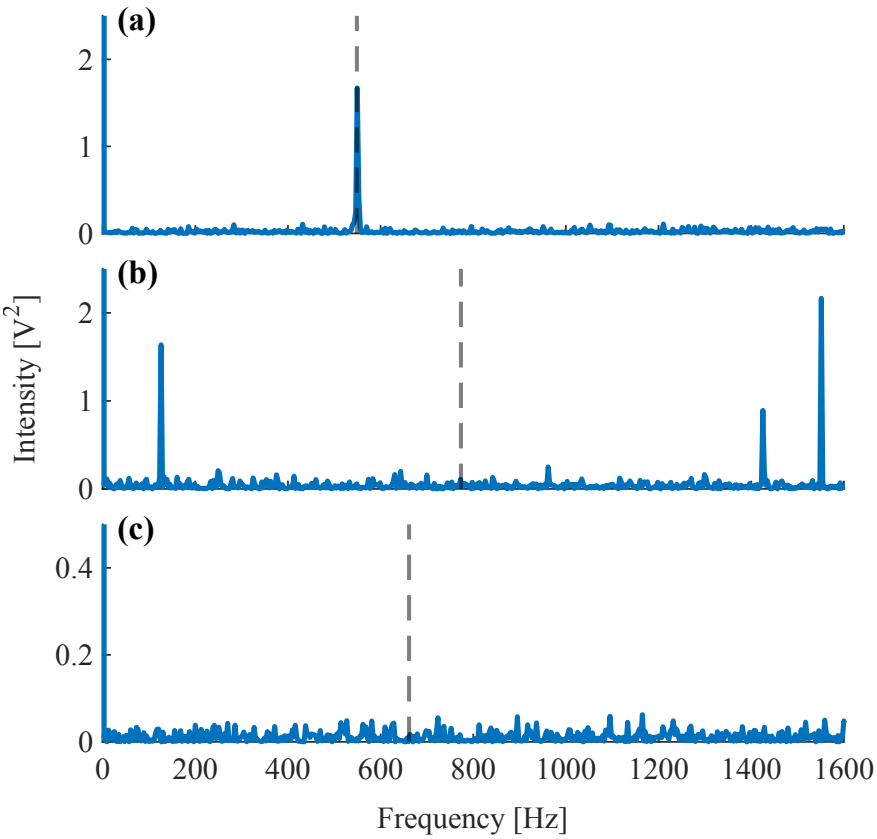



Figure 54: Examples of spectra and their corresponding label. **(a)** Example of a "yes" spectrum where the ground truth frequency aligns with the measured data. **(b)** Example of a "maybe" spectrum where the spectrum is distinct but the ground truth frequencies do not match the measured data. **(c)** Example of a "no" spectrum with no clear peaks above the noise floor.

If the spectrum was distinct (e.g., clearly had a fundamental frequency and harmonics)

and aligned with the ground-truth data, then a "yes" was recorded. If a peak aligned with the ground truth but the signal was near the noise floor, a "maybe" was recorded. If distinct peaks seemed to come from a UAV but didn't align with the ground-truth data, then a "maybe" was recorded. If there were no distinct peaks above the noise floor and none near the expected frequencies, then a "no" was recorded.

These visual inspection results were used to determine how tilt angle and beam configuration affected the lidar's ability to detect propeller frequencies.

## Results and Discussion

### Visible Inspection

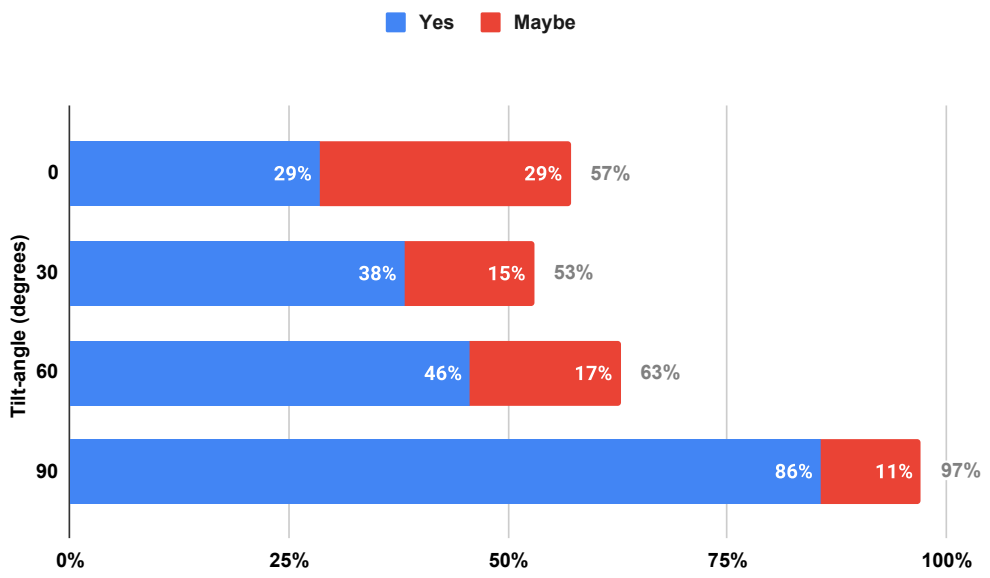



Figure 55: Propeller frequency detectability as a function of angle. The gray text to the right of the bars shows the sum of the "yes" and "maybe" percentages.

Fig. 55 shows how propeller frequency detection (via manual visual inspection) varied as a function of angle in the lab. When the UAV was parallel to the ground at 0°, the propeller frequency was visible in 9% of the images and possibly visible in 27%, making a total of 36% of the images that had some frequency information of the UAV. For 10° to 30°, the propeller

frequency was visible less than 5% of the time. After 30°, the propeller frequency was more visible as the tilt angle increased. The best results were for tilt angles of 80° and 90°, where the detectability was around 75%.

The detectability also depended on the propellers' orientation and the number of propellers in the beam, as shown in Fig. 56. When the laser intersected only one propeller or the right propellers, the detectability was approximately 45% when including "maybe" labels. The main outlier in this data set was the images containing the front propellers: the detectability was nearly 0%. When the entire UAV and all four propellers were in the beam, detectability was approximately 60%.

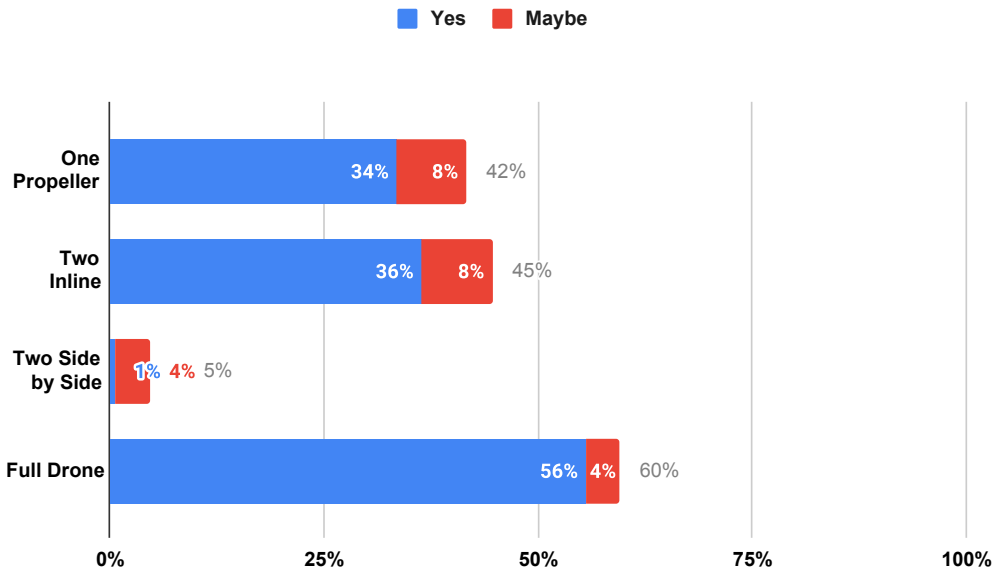



Figure 56: Propeller frequency detectability based on beam configuration. "One Propeller" corresponds to the configurations shown in Fig. 50**(a)** and **(b)**. "Two Inline" corresponds to Fig .50**(c)**, "Two Side by Side" corresponds to Fig. 50**(d)**, and "Full Drone" corresponds to Fig. 50**(e)**.

Supervised Learning

Data Sampling Table 18 shows the undersampling and oversampling ratios that achieved the best MCC scores on the validation set.

Table 18: Undersampling and augmentation ratios that resulted in the best MCC scores on the validation set. The best MCC is highlighted in bold. "NaN" is "not a number", indicating that the algorithm did not predict any drones, resulting in a divide by 0 in the MCC equation.

| **Method** | **Undersampling Ratio** | **Augmentation Ratio** | **MCC** |
|---|---|---|---|
| AdaBoost | 0 | 1 | **0.605** |
| RUSBoost | 0 | 1 | 0.504 |
| SVM | 0 | 1 | 0.130 |
| MLP 1 | 0.25 | 0 | NaN |
| MLP 3 | 0.25 | 0 | NaN |
| MLP 5 | 0.25 | 0 | NaN |
| MLP 7 | 0.25 | 0 | NaN |
| 1D CNN 1 | 0.50 | 0 | 0.280 |
| 1D CNN 3 | 0.50 | 0 | 0.477 |
| 1D CNN 5 | 0 | 1 | 0.472 |
| 1D CNN 7 | 0.75 | 0 | 0.458 |

Most algorithms did not benefit from oversampling; only small oversampling ratios of 1 synthetic data point per UAV data point were seen. This makes sense given that each image has multiple rows containing a UAV signal, leading to a dataset that is not as imbalanced as the insect dataset in Chapter 3. Each MLP predicted every data point as "nothing" (i.e., not a UAV), which resulted in MCC values of NaN (not a number). The decision tree algorithms performed best.

Validation During hyperparameter tuning, the training set was sampled using the undersampling and augmentation ratios in Table 18. Table 19 shows the MCC scores that the tuned models obtained on the validation set after hyperparameter tuning. If the default hyperparameters performed better than all hyperparameter combinations encountered during

Table 19: Classification results on the validation set. The best results for each metric are shown in bold. Methods marked with a * indicate that the default hyperparameters performed best. The "Per UAV" results indicate the percentage of UAVs that were identified.

| **Method** | **Rows** | | | **Per UAV** |
|---|---|---|---|---|
| | **Recall** | **Precision** | **MCC** | **Recall** |
| AdaBoost | **0.580** | 0.809 | **0.672** | **0.975** |
| RUSBoost | 0.492 | 0.565 | 0.506 | 0.947 |
| SVM | 0.053 | **0.882** | 0.209 | 0.087 |
| MLP 1 | 0 | NaN | NaN | 0.000 |
| MLP 3 | 0 | NaN | NaN | 0.000 |
| MLP 5 | 0 | NaN | NaN | 0.000 |
| MLP 7 | 0 | NaN | NaN | 0.000 |
| 1D CNN 1 | 0.407 | 0.437 | 0.394 | 0.845 |
| 1D CNN 3 | 0.477 | 0.564 | 0.496 | 0.948 |
| *1D CNN 5 | 0.308 | 0.769 | 0.472 | 0.763 |
| *1D CNN 7 | 0.372 | 0.613 | 0.458 | 0.899 |

hyperparameter tuning, the default parameters were used for the final training.

For all evaluated hyperparameter settings, the MLPs predicted everything as "nothing". The SVM had a very low recall rate, indicating it did not correctly predict most UAV rows. AdaBoost preformed the best on the validation set, yet only correctly predicted 58% of the UAV rows. However, due to the UAV propeller(s) showing up in multiple adjacent range bins, it's possible that the models correctly predict some of the UAV rows while missing others. The "Per UAV" column in Table 19 aggregates the row predictions to determine what percent of UAVs the models detected. If any of the adjacent rows containing a UAV were predicted as a UAV, that is considered a correct detection. The aggregated results show that the UAV detection performance is much better than the row-based results suggest. AdaBoost

Table 20: Classification results on the testing set. The best results for each metric are shown in bold.

| **Method** | **Rows** | | | **Per UAV** |
|---|---|---|---|---|
| | **Recall** | **Precision** | **MCC** | **Recall** |
| AdaBoost | **0.613** | 0.808 | **0.691** | 0.969 |
| RUSBoost | 0.521 | 0.566 | 0.522 | 0.939 |
| SVM | 0.079 | 0.927 | 0.264 | 0.113 |
| MLP 1 | 0.000 | NaN | NaN | 0.000 |
| MLP 3 | 0.000 | NaN | NaN | 0.000 |
| MLP 5 | 0.000 | NaN | NaN | 0.000 |
| MLP 7 | 0.000 | NaN | NaN | 0.000 |
| 1D CNN 1 | 0.577 | 0.288 | 0.366 | **0.970** |
| 1D CNN 3 | 0.504 | 0.505 | 0.480 | 0.967 |
| 1D CNN 5 | 0.025 | **1.000** | 0.154 | 0.059 |
| 1D CNN 7 | 0.276 | 0.754 | 0.442 | 0.719 |

correctly identified 97.5% of UAVs.

Testing Using the best parameters found during the data sampling and hyperparameter tuning phases, each model was trained one final time on the full training and validation datasets. These trained models were then tested on the testing set, which was collected approximately one month after the training data was collected. Table 20 shows the recall, precision, and MCC scores that the models obtained on the testing set.

The MLPs again failed to predict any UAV rows. The 1D CNN achieved the highest per-UAV recall, though it's low row-based precision means the model predicted many false positive UAV rows, which is ultimately not helpful for UAV detection. AdaBoost performed the best overall with reasonable row-based precision of 80.8% and a 96.9% recall of UAVs.

Feature Importance Figure 57 shows the features ranked by their mutual information with the class labels. The standard deviation had the highest correlation to the class labels, while some wavelet features also had high mutual information. The harmonic locations and heights did correlate to the class labels, but they had significantly lower mutual information than most of the time-domain, wavelet, and ESD statistical features. The harmonic locations and heights had higher mutual information with the UAV data than they did with the insect data in Chapter 3; I made some improvements to the harmonic feature extraction code after the insect experiments, so it's possible that those improvements lead to increased performance of some harmonic features.

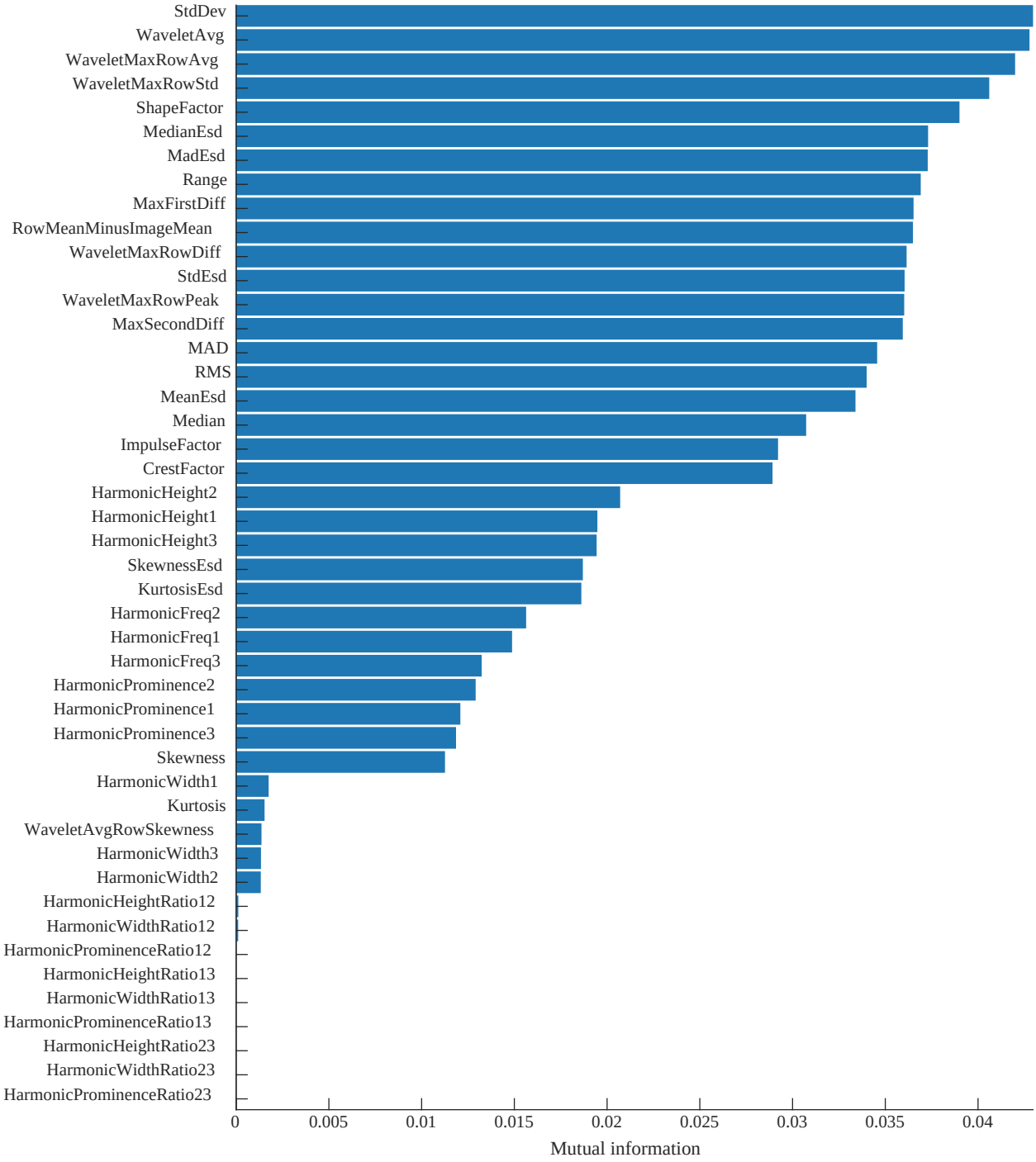



Figure 57: A list of all features ranked by their mutual information with the class labels.

Comparison to Visual Inspection Results The machine learning methods detected more UAVs than we did via visual inspection of spectra. By visual inspection of spectra, we saw a UAV-caused spectrum in 42.36% of the files. In contrast, the machine learning methods detected approximately 97% of UAVs in the validation and testing sets. We didn't look at each of the 32 images in a file during visual inspection, whereas the machine learning models did see every image in the validation and testing sets. Consequently, the percentages do not refer to the same exact set of data or number of images. However, all 32 images within a file were collected consecutively and look similar. In spite of the differences in the data being considered in each case, we can extrapolate that the machine learning methods performed better than we did at detecting UAVs. Considering that many of the most relevant features in Fig. 57 were from the time-domain or wavelet transform, the machine learning methods likely picked up on features and patterns we did not look at or identify.

Aliasing

The maximum RPM of a UAV motor depends on many factors, including the motor's $K_v$ rating, the battery voltage, and the load. Our custom Stan FPV Feather used 1900 $K_v$ Emax ECO II motors and a 15 V power supply (which is similar to a 4S lithium-polymer battery), which resulted in RPM readings around 30 000 RPM at full throttle. Since we used a 3-blade propeller, 30 000 RPM corresponds to 1500 Hz.

The PRF of our laser varied between approximately 3.8 kHz to 4 kHz, which resulted in a typical Nyquist frequency between 1.9 kHz to 2 kHz. While the maximum propeller frequency of approximately 1500 Hz is below the Nyquist frequency, the corresponding harmonics are above the Nyquist frequency. Consequently, we observed significant aliasing in some of the data, as the example in Fig. 58 shows. Although aliasing does not affect our ability to manually determine the propeller's fundamental frequency, it would affect algorithmic estimation of the fundamental frequency. Using a laser with a higher PRF would mitigate

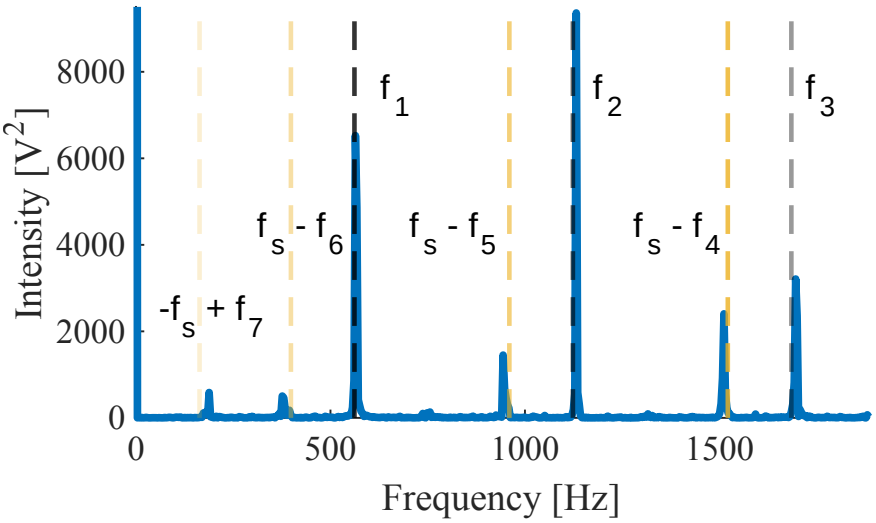



Figure 58: Example of aliasing. The fundamental frequency, $f_1$, was approximately 561 Hz. The non-aliased harmonic frequencies are gray lines, and the aliased harmonic frequencies are yellow lines, decreasing opacity corresponding to the distance from the fundamental. The aliases are annotated as $\pm f_s \pm f_n$, where $f_s$ is the sampling frequency and $f_n$ is the harmonic number. Aliases of the 8$^{\text{th}}$ and 9$^{\text{th}}$ harmonics are also visible, but are not labeled.

this issue.

Signal-to-Noise Ratio

The indoor data had low signal-to-noise ratio (SNR) compared to the outdoor data. This was caused by the telescope's nominally long-range focus and the expanded beam sizes. In order to cover the entire UAV, the beam size was expanded to approximately 300 mm. Since the 305-mm telescope had a small field of view (0.2 mrad half-angle divergence), much of the light reflected off the UAV was likely not received by the telescope. We believe these two issues resulted in much of the data having a low signal-to-noise ratio, especially at low tilt angles. These issues may have contributed to the propeller signals not showing up in the data collected with the propeller configuration shown in Fig. 50**(d)**. Overall, the low signal-to-noise ratio and poor data quality prevent us from drawing definitive conclusions about angle dependence and multiple-frequency performance.

## Visibility of Multiple Propeller Frequencies

In spite of the low SNR, we were able to see multiple propeller frequencies in a few scenarios. Figure 59 shows an example of a spectrum with multiple propeller frequencies present. Better quality data and more analysis will be required to characterize the system's ability to detect multiple propeller frequencies, but these initial experiments show promise.

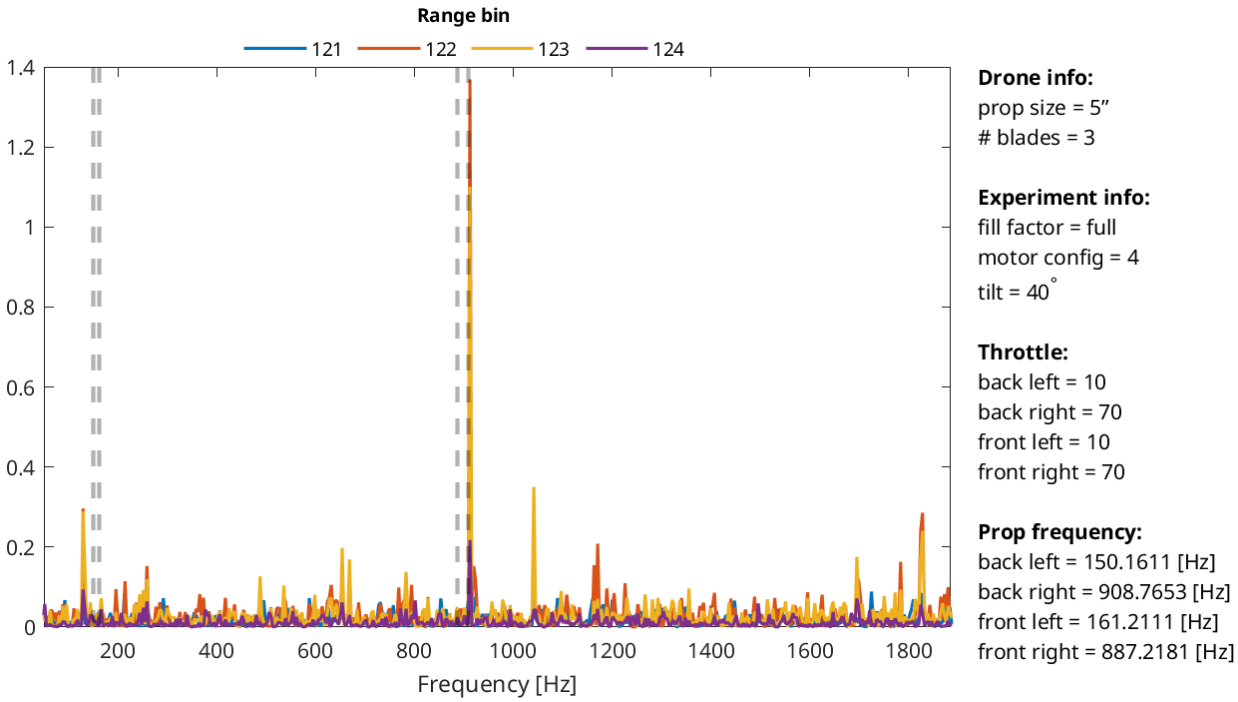



Figure 59: Example spectrum showing multiple propeller frequencies. The strong peak near 900 Hz corresponds to the back-right and/or front-right propeller signal. Although the peak near the front-left and back-left propeller frequencies does not line up perfectly, it is likely caused by one or both of those propellers.

## Conclusion

UAV detect-and-avoid systems rely on accurately detecting and characterizing UAVs. This chapter detailed the development and execution of experiments designed to test the wingbeat-modulation lidar's ability in UAV detection and characterization.

We demonstrated that the wingbeat-modulation lidar method can detect UAV propeller frequencies across different throttle speeds, tilt angles, and beam sizes. We were able to visually identify UAV signals when the full UAV was covered by the lidar beam, which is an improvement upon our previous single-propeller work [177]. Low SNR caused by our experiment setup reduced our ability to visually identify UAV signals. However, some machine learning models I used were able to detect approximately 97% of the UAVs in the dataset, which is more than double what we could see via visual inspection.

In data where the propeller was clearly visible in the spectrum, visual inspection showed that the ground truth propeller frequency matched what was visible in the lidar data. Preliminary exploration of the algorithmically estimated fundamental frequencies showed that the estimated frequency often did not match the ground truth. Due to this, a detailed analysis was not performed. Low SNR and aliasing contributed to the difficulty of algorithmically estimating propeller frequency. Future work should look into using a laser with a higher PRF to reduce aliasing. With better data, the accuracy of frequency estimation algorithms should be thoroughly evaluated.

The work in this chapter lays the groundwork for future UAV detection and characterization experiments with the wingbeat-modulation lidar. All the engineering effort that went into building and automating the experiment setup can be reused in future experiments. Overall, this chapter shows that wingbeat-modulation lidar can consistently detect UAVs in lidar data and is capable of measuring propeller frequency under various circumstances. The wingbeat-modulation lidar method has potential to aid and supplement existing UAV detect-and-avoid methods.

## Acknowledgments

I would like to acknowledge John Fike and Wyatt Weller for all their help with the UAV experiments. John was very helpful with the preliminary work, teaching me about UAVs, and

helping upgrade the lidar. Wyatt made many contributions to what ended up in this chapter. He helped with a lot of the engineering that was required to run the UAV experiments; he also contributed to the visual data analysis. These experiments required a lot of engineering and two years to make happen!

Computational efforts were performed on the Tempest High Performance Computing System, operated and supported by University Information Technology Research Cyberinfrastructure (RRID:SCR_026229) at Montana State University.

# CONCLUSION

Lidar is commonly used for remotely detecting and characterizing objects in a large variety of applications. Many lidar applications produce large amounts of imbalanced data where the majority of measurements do not contain the object of interest. This imbalanced data makes detecting objects time-consuming when humans have to manually analyze the data. In addition to making manual analysis time-consuming, imbalanced data presents numerous challenges for machine-learning based object detection. When trained on imbalanced data, most traditional machine learning algorithms will fail to identify the minority class because the majority class data points dominate the value of the objective function being minimized.

The work in this dissertation applied signal processing and machine learning techniques to address the time-consuming and challenging nature of detecting objects in imbalanced lidar data. By using my expertise in signal processing and class-imbalanced machine learning, I was able to make important contributions in three application areas: fish detection, insect detection, and UAV detection and characterization. Each of these applications required specific domain knowledge to help automate data analysis and object detection. The domain knowledge I learned during each project was used to identify and extract important features that the machine learning algorithms could use to detect the objects of interest.

Detecting fish in lidar data has traditionally been done manually [34–36]. By applying machine learning to existing datasets, I was able to develop methods that would significantly reduce the time required to detect fish. My methods were able to reliably identify regions of interest that contained fish. By only having to analyze a small number of regions, my methods would have reduced the total analysis time from 14 hours to 9.4 hours. Reducing data analysis time will allow fisheries personnel to accomplish their faster—whether that is estimating biomass or removing invasive species. My methods help reduce a primary bottleneck that limits the efficiency of lidar-based fisheries applications.

Similar to fish detection, reliably identifying insects in entomological lidar data is one of the primary challenges and bottlenecks in the field. Entomological lidar data is highly imbalanced and contains vast amounts of data. If insect signals can be reliably identified, entomological lidar can be used for long-term non-invasive monitoring of insects. My machine learning methods reliably detected insect signals, and I contributed an in-depth analysis of which features were most salient for insect detection. These contributions will help future researchers develop more methods and algorithms that enable entomological lidar to help scientists understand insect behavior, abundance, and biodiversity loss.

UAV detection and characterization concluded this dissertation. Building upon the work I did detecting insects, I detected UAVs and measured UAV propeller frequencies. My machine learning techniques detected approximately 97% of the UAVs in the dataset, outperforming our ability to manually detect UAVs in approximately 43% of the data via visual inspection. Visual inspection showed that we were able to correctly identify UAV propeller frequencies in the data, which is an important way to distinguish between small UAVs and birds [41]. With further work, our lidar and data analysis method can be utilized as part of a comprehensive UAV detection and characterization system.

The primary limitation of my work is that supervised machine learning requires labeled data. Such labeling is usually done manually, which is time-consuming. Future work should be directed towards using unsupervised machine learning techniques which do not require labels. If unsupervised techniques, which generally perform worse than supervised methods, can successfully detect objects in imbalanced lidar data, human involvement in the data analysis pipeline will be significantly reduced.

In summary, my research has advanced the state-of-the-art in fish detection, insect detection, and UAV detection and characterization. My contributions in each field will enable researchers that are not signal processing and machine learning experts to utilize automated object detection techniques in their applications.

# CUMULATIVE REFERENCES CITED


[1] Behnam Behroozpour et al. "Lidar System Architectures and Circuits". In: *IEEE Communications Magazine* 55.10 (Oct. 2017), pp. 135–142. DOI: 10.1109/mcom.2017.1700030.

[2] Martin J. Tauc, Kurt M. Fristrup, and Joseph A. Shaw. "Development of a wing-beat-modulation scanning lidar system for insect studies". In: *Lidar Remote Sensing for Environmental Monitoring 2017*. Ed. by Upendra N. Singh. SPIE, Aug. 2017. DOI: 10.1117/12.2274656.

[3] Vladimir Vapnik. *Estimation of dependences based on empirical data*. Springer Science & Business Media, 2006.

[4] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition". In: SPRINGER NATURE, Feb. 2009. Chap. 14.1. ISBN: 0387848576.

[5] R. A. FISHER. "THE USE OF MULTIPLE MEASUREMENTS IN TAXONOMIC PROBLEMS". In: *Annals of Eugenics* 7.2 (Sept. 1936), pp. 179–188. DOI: 10.1111/j.1469-1809.1936.tb02137.x.

[6] Leo Breiman et al. *Classification and regression trees*. CRC press, 1984.

[7] Kishan Mehrotra, Chilukuri K Mohan, and Sanjay Ranka. *Elements of artificial neural networks*. MIT press, 1997.

[8] Evelyn Fix and J. L. Hodges. "Discriminatory Analysis. Nonparametric Discrimination: Consistency Properties". In: *International Statistical Review / Revue Internationale de Statistique* 57.3 (Dec. 1989), p. 238. DOI: 10.2307/1403797.

[9] S. Lloyd. "Least squares quantization in PCM". In: *IEEE Transactions on Information Theory* 28.2 (Mar. 1982), pp. 129–137. DOI: 10.1109/tit.1982.1056489.

[10] A. P. Dempster, N. M. Laird, and D. B. Rubin. "Maximum Likelihood from Incomplete Data Via the EM Algorithm". In: *Journal of the Royal Statistical Society: Series B (Methodological)* 39.1 (Sept. 1977), pp. 1–22. DOI: 10.1111/j.2517-6161.1977.tb01600.x.

[11] Martin Ester et al. "A density-based algorithm for discovering clusters in large spatial databases with noise." In: *kdd*. Vol. 96. 34. 1996, pp. 226–231.

[12] Fionn Murtagh and Pedro Contreras. "Algorithms for hierarchical clustering: an overview". In: *WIREs Data Mining and Knowledge Discovery* 2.1 (Dec. 2011), pp. 86–97. DOI: 10.1002/widm.53.

[13] Andrew Y Ng, Michael I Jordan, and Yair Weiss. "On spectral clustering: Analysis and an algorithm". In: *Advances in neural information processing systems*. 2002, pp. 849–856.

[14] Brian W. Matthews. "Comparison of the predicted and observed secondary structure of T4 phage lysozyme". In: *Biochimica et Biophysica Acta (BBA) - Protein Structure* 405.2 (Oct. 1975), pp. 442–451. DOI: 10.1016/0005-2795(75)90109-9.

[15] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (Jan. 2020). DOI: 10.1186/s12864-019-6413-7.

[16] Nitesh V. Chawla, Nathalie Japkowicz, and Aleksander Kotcz. "Editorial". In: *ACM SIGKDD Explorations Newsletter* 6.1 (June 2004), pp. 1–6. DOI: 10.1145/1007730.1007733.

[17] Haibo He and E.A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284. DOI: 10.1109/tkde.2008.239.

[18] Gang Wu and Edward Y. Chang. "Class-boundary alignment for imbalanced dataset learning". In: *ICML 2003 workshop on learning from imbalanced data sets II, Washington, DC*. 2003, pp. 49–56.

[19] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. "The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition". In: SPRINGER NATURE, Feb. 1, 2009. Chap. 12.2. ISBN: 0387848576.

[20] Foster Provost. "Machine learning from imbalanced data sets 101". In: *Proceedings of the AAAI'2000 workshop on imbalanced data sets*. Vol. 68. 2000. AAAI Press. 2000, pp. 1–3.

[21] Gary M. Weiss. "Mining with rarity". In: *ACM SIGKDD Explorations Newsletter* 6.1 (June 2004), pp. 7–19. DOI: 10.1145/1007730.1007734.

[22] Chris Seiffert et al. "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.1 (Jan. 2010), pp. 185–197. DOI: 10.1109/tsmca.2009.2029559.

[23] Gustavo E. A. P. A. Batista, Ronaldo C. Prati, and Maria Carolina Monard. "A study of the behavior of several methods for balancing machine learning training data". In: *ACM SIGKDD Explorations Newsletter* 6.1 (June 2004), pp. 20–29. DOI: 10.1145/1007730.1007735.

[24] Jason Van Hulse, Taghi M. Khoshgoftaar, and Amri Napolitano. "Experimental perspectives on learning from imbalanced data". In: *Proceedings of the 24th international conference on Machine learning - ICML '07*. ACM Press, 2007. DOI: 10.1145/1273496.1273614.

[25] I Tomek. "Two Modifications of CNN". In: *IEEE Trans. Systems, Man and Cybernetics* 6 (1976), pp. 769–772.

[26] PE Hart. "The condensed nearest neighbor rule". In: *IEEE Trans. Inf. Theory* 14.3 (1968), pp. 515–516.

[27] Salvador García and Francisco Herrera. "Evolutionary Undersampling for Classification with Imbalanced Datasets: Proposals and Taxonomy". In: *Evolutionary Computation* 17.3 (Sept. 2009), pp. 275–306. DOI: 10.1162/evco.2009.17.3.275.

[28] Alberto Fernández. *Learning from imbalanced data sets*. Cham, Switzerland: Springer, 2018. ISBN: 3319980742.

[29] Sebastien C. Wong et al. "Understanding Data Augmentation for Classification: When to Warp?" In: *2016 International Conference on Digital Image Computing: Techniques and Applications (DICTA)*. IEEE, Nov. 2016. DOI: 10.1109/dicta.2016.7797091.

[30] N. V. Chawla et al. "SMOTE: Synthetic Minority Over-sampling Technique". In: *Journal of Artificial Intelligence Research* 16 (June 2002), pp. 321–357. DOI: 10.1613/jair.953.

[31] Haibo He et al. "ADASYN: Adaptive synthetic sampling approach for imbalanced learning". In: *2008 IEEE International Joint Conference on Neural Networks (IEEE World Congress on Computational Intelligence)*. IEEE, June 2008. DOI: 10.1109/ijcnn.2008.4633969.

[32] Yoav Freund, Robert E. Schapire, et al. "Experiments with a new boosting algorithm". In: *icml*. Vol. 96. Citeseer. 1996, pp. 148–156.

[33] R. Elwell and R. Polikar. "Incremental Learning of Concept Drift in Nonstationary Environments". In: *IEEE Transactions on Neural Networks* 22.10 (Oct. 2011), pp. 1517–1531. DOI: 10.1109/tnn.2011.2160459.

[34] James H. Churnside and Richard D. Marchbanks. "Subsurface plankton layers in the Arctic Ocean". In: *Geophysical Research Letters* 42.12 (June 2015), pp. 4896–4902. DOI: 10.1002/2015gl064503.

[35] JamesH Churnside et al. "Surveying the distribution and abundance of flying fishes and other epipelagics in the northern Gulf of Mexico using airborne lidar". In: *Bulletin of Marine Science* 93.2 (Apr. 2017), pp. 591–609. DOI: 10.5343/bms.2016.1039.

[36] Michael R. Roddewig, James H. Churnside, and Joseph A. Shaw. *Lidar measurements of the diffuse attenuation coefficient in Yellowstone Lake*. Mar. 2020. DOI: 10.1364/AO.389169.

[37] James H. Churnside, Eirik Tenningen, and James J. Wilson. "Comparison of data-processing algorithms for the lidar detection of mackerel in the Norwegian Sea". In: *ICES Journal of Marine Science* 66.6 (Feb. 2009), pp. 1023–1028. DOI: 10.1093/icesjms/fsp026.

[38] James H. Churnside and Richard E. Thorne. "Comparison of airborne lidar measurements with 420 kHz echo-sounder measurements of zooplankton". In: *Applied Optics* 44.26 (Sept. 2005), p. 5504. DOI: 10.1364/ao.44.005504.

[39] Roel van Klink et al. "Emerging technologies revolutionise insect ecology and monitoring". In: *Trends in Ecology & Evolution* 37.10 (Oct. 2022), pp. 872–885. DOI: 10.1016/j.tree.2022.06.001.

[40] Joseph A. Shaw et al. "Optical detection of oscillating targets using modulation of scattered laser light". English. U.S. pat. 7,511,624. Mar. 2009. URL: https://patents.google.com/patent/US20060254522A1/en.

[41] Ram M. Narayanan, Bryan Tsang, and Ramesh Bharadwaj. "Classification and Discrimination of Birds and Small Drones Using Radar Micro-Doppler Spectrogram Images". In: *Signals* 4.2 (May 2023), pp. 337–358. DOI: 10.3390/signals4020018.

[42] Ransom A. Myers and Boris Worm. "Rapid worldwide depletion of predatory fish communities". In: *Nature* 423.6937 (May 2003), pp. 280–283. DOI: 10.1038/nature01610.

[43] John Hampton et al. "Decline of Pacific tuna populations exaggerated?" In: *Nature* 434.7037 (2005), E1–E2. DOI: 10.1038/nature03581.

[44] Tom Polacheck. "Tuna longline catch rates in the Indian Ocean: Did industrial fishing result in a 90% rapid decline in the abundance of large predatory species?" In: *Marine Policy* 30.5 (Sept. 2006), pp. 470–482. DOI: 10.1016/j.marpol.2005.06.016.

[45] RK Bauer et al. "Aerial surveys to monitor bluefin tuna abundance and track efficiency of management measures". In: *Marine Ecology Progress Series* 534 (Aug. 2015), pp. 221–234. DOI: 10.3354/meps11392.

[46] M Lutcavage, S Kraus, and W Hoggard. "Aerial survey of giant bluefin tuna, Thunnus thynnus, in the great Bahama Bank, Straits of Florida, 1995". English. In: *FISHERY BULLETIN* 95.2 (APR 1997), 300–310. ISSN: 0090-0656.

[47] James H. Churnside, Alexei F. Sharov, and Ronald A. Richter. "Aerial surveys of fish in estuaries: a case study in Chesapeake Bay". In: *ICES Journal of Marine Science* 68.1 (Sept. 2010), pp. 239–244. ISSN: 1054-3139. DOI: 10.1093/icesjms/fsq138. eprint: https://academic.oup.com/icesjms/article-pdf/68/1/239/29137540/fsq138.pdf. URL: https://doi.org/10.1093/icesjms/fsq138.

[48] James H. Churnside, James J. Wilson, and Viatcheslav V. Tatarskii. "Lidar profiles of fish schools". In: *Applied Optics* 36.24 (Aug. 1997), p. 6011. DOI: 10.1364/ao.36.006011.

[49] James H. Churnside, James J. Wilson, and Viatcheslav V. Tatarskii. "Airborne lidar for fisheries applications". In: *Optical Engineering* 40.3 (Mar. 2001), p. 406. DOI: 10.1117/1.1348000.

[50] J Churnside. "A comparison of lidar and echosounder measurements of fish schools in the Gulf of Mexico". In: *ICES Journal of Marine Science* 60.1 (Feb. 2003), pp. 147–154. DOI: 10.1006/jmsc.2002.1327.

[51] Michael R. Roddewig et al. "Dual-polarization airborne lidar for freshwater fisheries management and research". In: *Optical Engineering* 56.3 (Mar. 2017), p. 031221. DOI: 10.1117/1.oe.56.3.031221.

[52] Michael R. Roddewig et al. "Airborne lidar detection and mapping of invasive lake trout in Yellowstone Lake". In: *Applied Optics* 57.15 (May 2018), p. 4111. DOI: 10.1364/ao.57.004111.

[53] William G. Pichel et al. "Marine debris collects within the North Pacific Subtropical Convergence Zone". In: *Marine Pollution Bulletin* 54.8 (Aug. 2007), pp. 1207–1211. DOI: 10.1016/j.marpolbul.2007.04.010.

[54] Timothy S. Veenstra and James H. Churnside. "Airborne sensors for detecting large marine debris at sea". In: *Marine Pollution Bulletin* 65.1-3 (Jan. 2012), pp. 63–68. DOI: 10.1016/j.marpolbul.2010.11.018.

[55] James H. Churnside and Percy L. Donaghay. "Thin scattering layers observed by airborne lidar". In: *ICES Journal of Marine Science* 66.4 (Feb. 2009), pp. 778–789. DOI: 10.1093/icesjms/fsp029.

[56] Kaipeng Li et al. “A Dual-Wavelength Ocean Lidar for Vertical Profiling of Oceanic Backscatter and Attenuation”. In: *Remote Sensing* 12.17 (Sept. 2020), p. 2844. DOI: 10.3390/rs12172844.

[57] Wei Li et al. “Influence of characteristics of micro-bubble clouds on backscatter lidar signal”. In: IEEE, 2009. DOI: 10.1364/oe.17.017772.

[58] James H. Churnside. “Lidar signature from bubbles in the sea”. In: *Optics Express* 18.8 (Apr. 2010), p. 8294. DOI: 10.1364/oe.18.008294.

[59] Michael R. Roddewig, James H. Churnside, and Joseph A. Shaw. “Airborne lidar detection of an underwater thermal vent”. In: *Journal of Applied Remote Sensing* 11.03 (Aug. 2017), p. 1. DOI: 10.1117/1.jrs.11.036014.

[60] S Rodier et al. “CALIPSO lidar measurements for ocean sub-surface studies”. In: *34th International Symposium on Remote Sensing of the Environment*. 2011.

[61] James Churnside, Brandi McCarty, and Xiaomei Lu. “Subsurface Ocean Signals from an Orbiting Polarization Lidar”. In: *Remote Sensing* 5.7 (July 2013), pp. 3457–3475. DOI: 10.3390/rs5073457.

[62] Michael J. Behrenfeld et al. “Space-based lidar measurements of global ocean carbon stocks”. In: *Geophysical Research Letters* 40.16 (Aug. 2013), pp. 4355–4360. DOI: 10.1002/grl.50816.

[63] Xiaomei Lu et al. “Ocean subsurface studies with the CALIPSO spaceborne lidar”. In: *Journal of Geophysical Research: Oceans* 119.7 (July 2014), pp. 4305–4317. DOI: 10.1002/2014jc009970.

[64] Michael J. Behrenfeld et al. “Annual boom–bust cycles of polar phytoplankton biomass revealed by space-based lidar”. In: *Nature Geoscience* 10.2 (Dec. 2016), pp. 118–122. DOI: 10.1038/ngeo2861.

[65] Yongxiang Hu and Pengwang Zhai. “Development and validation of the CALIPSO ocean subsurface data”. In: *2016 IEEE International Geoscience and Remote Sensing Symposium (IGARSS)*. IEEE, July 2016. DOI: 10.1109/igarss.2016.7729981.

[66] Yongxiang Hu et al. “Ocean Lidar Measurements of Beam Attenuation and a Roadmap to Accurate Phytoplankton Biomass Estimates”. In: *EPJ Web of Conferences* 119 (2016). Ed. by B. Gross, F. Moshary, and M. Arend, p. 22003. DOI: 10.1051/epjconf/201611922003.

[67] Chris A. Hostetler et al. “Spaceborne Lidar in the Study of Marine Systems”. In: *Annual Review of Marine Science* 10.1 (Jan. 2018), pp. 121–147. DOI: 10.1146/annurev-marine-121916-063335.

[68] Davide Dionisi et al. “Seasonal distributions of ocean particulate optical properties from spaceborne lidar measurements in Mediterranean and Black sea”. In: *Remote Sensing of Environment* 247 (Sept. 2020), p. 111889. DOI: 10.1016/j.rse.2020.111889.

[69] H.-G. Maas et al. “IMPROVEMENTS IN LIDAR BATHYMETRY DATA ANALYSIS”. In: *ISPRS - International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences* XLII-2/W10 (Apr. 2019), pp. 113–117. DOI: 10.5194/isprs-archives-xlii-2-w10-113-2019.

[70] Nan Xu et al. “A Method to Derive Bathymetry for Dynamic Water Bodies Using ICESat-2 and GSWD Data Sets”. In: *IEEE Geoscience and Remote Sensing Letters* (2020), pp. 1–5. DOI: 10.1109/lgrs.2020.3019396.

[71] Xinglei Zhao et al. “Improved waveform decomposition with bound constraints for green waveforms of airborne LiDAR bathymetry”. In: *Journal of Applied Remote Sensing* 14.02 (May 2020), p. 1. DOI: 10.1117/1.jrs.14.027502.

[72] James H. Churnside and Joseph A. Shaw. *Lidar remote sensing of the aquatic environment: invited*. Mar. 2020. DOI: 10.1364/AO.59.000C92.

[73] Trevor Hastie, Robert Tibshirani, and Jerome Friedman. “The Elements of Statistical Learning: Data Mining, Inference, and Prediction, Second Edition”. In: SPRINGER NATURE, Feb. 1, 2009. Chap. 14.1. ISBN: 0387848576.

[74] Samuel J. Purkis and John C. Brock. “LiDAR Overview”. In: *Coral Reef Remote Sensing*. Springer Netherlands, 2013, pp. 115–143. DOI: 10.1007/978-90-481-9292-2_5.

[75] Frank E. Hoge et al. “Airborne lidar detection of subsurface oceanic scattering layers”. In: *Applied Optics* 27.19 (Oct. 1988), p. 3969. DOI: 10.1364/ao.27.003969.

[76] Yury A. Goldin et al. “Results of Barents Sea airborne lidar survey”. In: ed. by Iosif M. Levin et al. SPIE, Apr. 2007. DOI: 10.1117/12.740456.

[77] V. S. Shamanaev. “Laser Sensing of the Atmosphere and Upper Layer of the Ocean by an Airborn and Shipborne Lidars”. In: *Russian Physics Journal* 56.7 (Nov. 2013), pp. 813–821. DOI: 10.1007/s11182-013-0104-7.

[78] Alexander P. Vasilkov et al. “Airborne polarized lidar detection of scattering layers in the ocean”. In: *Applied Optics* 40.24 (Aug. 2001), p. 4353. DOI: 10.1364/ao.40.004353.

[79] Hang Liu et al. “Subsurface plankton layers observed from airborne lidar in Sanya Bay, South China Sea”. In: *Optics Express* 26.22 (Oct. 2018), p. 29134. DOI: 10.1364/oe.26.029134.

[80] Jian Ma et al. “Compact dual-wavelength blue-green laser for airborne ocean detection lidar”. In: *Applied Optics* 59.10 (Mar. 2020), p. C87. DOI: 10.1364/ao.382174.

[81] Heather M Zorn, James H Churnside, and Charles W Oliver. “Laser safety thresholds for cetaceans and pinnipeds”. In: *Marine mammal science* 16.1 (2000), pp. 186–200.

[82] Lynn R. Kaeding, Glenn D. Boltz, and Daniel G. Carty. “Lake Trout Discovered in Yellowstone Lake Threaten Native Cutthroat Trout”. In: *Fisheries* 21.3 (Mar. 1996), pp. 16–20. DOI: 10.1577/1548-8446(1996)021<0016:ltdiyl>2.0.co;2.

[83] James R. Ruzycki, David A. Beauchamp, and Daniel L. Yule. “EFFECTS OF INTRODUCED LAKE TROUT ON NATIVE CUTTHROAT TROUT IN YELLOWSTONE LAKE”. In: *Ecological Applications* 13.1 (Feb. 2003), pp. 23–37. DOI: 10.1890/1051-0761(2003)013[0023:eoilto]2.0.co;2.

[84] Todd M. Koel et al. “Nonnative Lake Trout Result in Yellowstone Cutthroat Trout Decline and Impacts to Bears and Anglers”. In: *Fisheries* 30.11 (Nov. 2005), pp. 10–19. DOI: 10.1577/1548-8446(2005)30[10:nltriy]2.0.co;2.

[85] James H. Churnside et al. “Hollow aggregations of moon jellyfish (Aureliaspp.)” In: *Journal of Plankton Research* 38.1 (Nov. 2015), pp. 122–130. DOI: 10.1093/plankt/fbv092.

[86] Gary C. Guenther. “Wind And Nadir Angle Effects On Airborne Lidar Water ”Surface” Returns”. In: *Ocean Optics VIII*. Ed. by Marvin A. Blizard. SPIE, Aug. 1986. DOI: 10.1117/12.964243.

[87] Georgii M. Krekov, Margarita M. Krekova, and Vitalii S. Shamanaev. “Laser sensing of a subsurface oceanic layer II Polarization characteristics of signals”. In: *Applied Optics* 37.9 (Mar. 1998), p. 1596. DOI: 10.1364/ao.37.001596.

[88] James H. Churnside. “Polarization effects on oceanographic lidar”. In: *Optics Express* 16.2 (2008), p. 1196. DOI: 10.1364/oe.16.001196.

[89] George C. Mcgavin et al. *Essential entomology: an order-by-order introduction.* Oxford University Press, 2001.

[90] Nigel E. Stork. “How Many Species of Insects and Other Terrestrial Arthropods Are There on Earth?” In: *Annual Review of Entomology* 63.1 (Jan. 2018), pp. 31–45. DOI: 10.1146/annurev-ento-020117-043348.

[91] World Health Organization et al. “Malaria entomology and vector control”. In: (2013).

[92] Ulrich G. Mueller et al. “The Evolution of Agriculture in Insects”. In: *Annu. Rev. Ecol. Evol. Syst.* 36.1 (Dec. 2005), pp. 563–595. DOI: 10.1146/annurev.ecolsys.36.102003.152626.

[93] David L. Wagner. “Insect Declines in the Anthropocene”. In: *Annual Review of Entomology* 65.1 (Jan. 2020), pp. 457–480. DOI: 10.1146/annurev-ento-011019-025151.

[94] Charlotte L. Outhwaite, Peter McCann, and Tim Newbold. “Agriculture and climate change are reshaping insect biodiversity worldwide”. In: *Nature* 605.7908 (Apr. 2022), pp. 97–102. DOI: 10.1038/s41586-022-04644-x.

[95] Jeannine Cavender-Bares et al. “Integrating remote sensing with ecology and evolution to advance biodiversity conservation”. In: *Nature Ecology &Evolution* 6.5 (Mar. 2022), pp. 506–519. DOI: 10.1038/s41559-022-01702-5.

[96] Kenneth McCravy. “A Review of Sampling and Monitoring Methods for Beneficial Arthropods in Agroecosystems”. In: *Insects* 9.4 (Nov. 2018), p. 170. DOI: 10.3390/insects9040170.

[97] Graham A. Montgomery et al. “Is the insect apocalypse upon us? How to find out”. In: *Biological Conservation* 241 (Jan. 2020), p. 108327. DOI: 10.1016/j.biocon.2019.108327.

[98] Lars Eisen and Rebecca J. Eisen. “Need for Improved Methods to Collect and Present Spatial Epidemiologic Data for Vectorborne Diseases”. In: *Emerging Infectious Diseases* 13.12 (Dec. 2007), pp. 1816–1820. DOI: 10.3201/eid1312.070211.

[99] Toke T. Høye et al. “Deep learning and computer vision will transform entomology”. In: *Proceedings of the National Academy of Sciences* 118.2 (Jan. 2021), e2002545117. DOI: 10.1073/pnas.2002545117.

[100] Rory Gibb et al. “Emerging opportunities and challenges for passive acoustics in ecological assessment and monitoring”. In: *Methods in Ecology and Evolution* 10.2 (Nov. 2018). Ed. by Luca Börger, pp. 169–185. DOI: 10.1111/2041-210x.13101.

[101] Drake V. Alistair and Reynolds Donald R., eds. *Radar entomology: observing insect flight and migration*. CABI, 2012. DOI: 10.1079/9781845935566.0000.

[102] Mahaveer Dwivedi, Malik Hashmat Shadab, and V. R. Santosh. “Insect Pest Detection, Migration and Monitoring Using Radar and LiDAR Systems”. In: *Innovative Pest*

*Management Approaches for the 21st Century*. Springer Singapore, 2020, pp. 61–76. DOI: 10.1007/978-981-15-0794-6_4.

[103] Alexey Noskov, Joerg Bendix, and Nicolas Friess. "A Review of Insect Monitoring Approaches with Special Reference to Radar Techniques". In: *Sensors* 21.4 (Feb. 2021), p. 1474. DOI: 10.3390/s21041474.

[104] Mikkel Brydegaard and Samuel Jansson. "Advances in entomological laser radar". In: *The Journal of Engineering* 2019.21 (Oct. 2019), pp. 7542–7545. DOI: 10.1049/joe.2019.0598.

[105] Martin J. Tauc et al. "Field demonstration of a wing-beat modulation lidar for the 3D mapping of flying insects". In: *OSA Continuum* 2.2 (Jan. 2019), p. 332. DOI: 10.1364/osac.2.000332.

[106] Benoit K. Kouakou et al. "Entomological Scheimpflug lidar for estimating unique insect classes in-situ field test from Ivory Coast". In: *OSA Continuum* 3.9 (Aug. 2020), p. 2362. DOI: 10.1364/osac.387727.

[107] Klas Rydhmer et al. "Scheimpflug lidar range profiling of bee activity patterns and spatial distributions". In: *Animal Biotelemetry* 10.1 (Apr. 2022). DOI: 10.1186/s40317-022-00285-z.

[108] Yihao Wang et al. "Real-time monitoring of insects based on laser remote sensing". In: *Ecological Indicators* 151 (July 2023), p. 110302. DOI: 10.1016/j.ecolind.2023.110302.

[109] Robin Steenweg et al. "Scaling-up camera traps: monitoring the planet's biodiversity with networks of remote sensors". In: *Frontiers in Ecology and the Environment* 15.1 (Dec. 2016), pp. 26–34. DOI: 10.1002/fee.1448.

[110] Jaeson Clayborn and Tyson Clayborn. "What happens in forests when nobody's present? A sustainable method to document insect behaviors and interactions using video surveillance". In: *International Journal of Tropical Insect Science* 39.4 (July 2019), pp. 341–345. DOI: 10.1007/s42690-019-00034-5.

[111] Michele Preti, François Verheggen, and Sergio Angeli. "Insect pest monitoring with camera-equipped traps: strengths and limitations". In: *Journal of Pest Science* 94.2 (Dec. 2020), pp. 203–217. DOI: 10.1007/s10340-020-01309-4.

[112] Maxime Martineau et al. "A survey on image-based insect classification". In: *Pattern Recognition* 65 (May 2017), pp. 273–284. DOI: 10.1016/j.patcog.2016.12.020.

[113] Mohammad Sadegh Norouzzadeh et al. "Automatically identifying, counting, and describing wild animals in camera-trap images with deep learning". In: *Proceedings of the National Academy of Sciences* 115.25 (June 2018), E5716–E5725. DOI: 10.1073/pnas.1719367115.

[114] Jana Wäldchen and Patrick Mäder. "Machine learning for image based species identification". In: *Methods in Ecology and Evolution* 9.11 (Sept. 2018). Ed. by Natalie Cooper, pp. 2216–2225. DOI: 10.1111/2041-210x.13075.

[115] Hayder Yousif et al. "Animal Scanner: Software for classifying humans, animals, and empty frames in camera trap images". In: *Ecology and Evolution* 9.4 (Feb. 2019), pp. 1578–1589. DOI: 10.1002/ece3.4747.

[116] Kim Bjerge et al. "An automated light trap to monitor moths (Lepidoptera) using computer vision-based tracking and deep learning". In: (Mar. 2020). DOI: 10.1101/2020.03.18.996447.

[117] Adrien P. Genoud et al. "A comparison of supervised machine learning algorithms for mosquito identification from backscattered optical signals". In: *Ecological Informatics* 58 (July 2020), p. 101090. DOI: 10.1016/j.ecoinf.2020.101090.

[118] Carsten Kirkeby et al. "Advances in automatic identification of flying insects using optical sensors and machine learning". In: *Scientific Reports* 11.1 (Jan. 2021). DOI: 10.1038/s41598-021-81005-0.

[119] Susan Bender et al. *Tracking Honey Bees Using LIDAR (Light Detection and Ranging) Technology*. Tech. rep. Jan. 2003. DOI: 10.2172/808625.

[120] Joseph A. Shaw et al. "Polarization lidar measurements of honey bees in flight for locating land mines". In: *Optics Express* 13.15 (2005), p. 5853. DOI: 10.1364/opex.13.005853.

[121] David S. Hoffman et al. "Range-resolved optical detection of honeybees by use of wing-beat modulation of scattered light for locating land mines". In: *Applied Optics* 46.15 (May 2007), p. 3007. DOI: 10.1364/ao.46.003007.

[122] Mikkel Brydegaard et al. "The Scheimpflug lidar method". In: *Lidar Remote Sensing for Environmental Monitoring 2017*. Ed. by Upendra N. Singh. SPIE, Aug. 2017. DOI: 10.1117/12.2272939.

[123] Zuguang Guan et al. "Insect monitoring with fluorescence lidar techniques: field experiments". In: *Applied Optics* 49.27 (Sept. 2010), p. 5133. DOI: 10.1364/ao.49.005133.

[124] Hampus Manefjord et al. “3D-Printed Fluorescence Hyperspectral Lidar for Monitoring Tagged Insects”. In: *IEEE Journal of Selected Topics in Quantum Electronics* 28.5: Lidars and Photonic Radars (Sept. 2022), pp. 1–9. DOI: 10.1109/jstqe.2022.3162417.

[125] Mikkel Brydegaard et al. “Lidar reveals activity anomaly of malaria vectors during pan-African eclipse”. In: *Science Advances* 6.20 (May 2020). DOI: 10.1126/sciadv.aay5487.

[126] Samuel Jansson et al. “Real-time dispersal of malaria vectors in rural Africa monitored with lidar”. In: *PLOS ONE* 16.3 (Mar. 2021). Ed. by Daniel Becker, e0247803. DOI: 10.1371/journal.pone.0247803.

[127] Elin Malmqvist et al. “Effective Parameterization of Laser Radar Observations of Atmospheric Fauna”. In: *IEEE Journal of Selected Topics in Quantum Electronics* 22.3 (May 2016), pp. 327–334. DOI: 10.1109/jstqe.2015.2506616.

[128] Klas Rydhmer et al. “Automating insect monitoring using unsupervised near-infrared sensors”. In: *Scientific Reports* 12.1 (Feb. 2022). DOI: 10.1038/s41598-022-06439-6.

[129] Klas Rydhmer and Raghavendra Selvan. “Dynamic β-VAEs for quantifying biodiversity by clustering optically recorded insect signals”. In: *Ecological Informatics* 66 (Dec. 2021), p. 101456. DOI: 10.1016/j.ecoinf.2021.101456.

[130] Alem Gebru et al. “Multiband modulation spectroscopy for the determination of sex and species of mosquitoes in flight”. In: *Journal of Biophotonics* 11.8 (Apr. 2018), e201800014. DOI: 10.1002/jbio.201800014.

[131] Trevor C. Vannoy et al. “Detection of Insects in Class-Imbalanced Lidar Field Measurements”. In: *2021 IEEE 31st International Workshop on Machine Learning for Signal Processing (MLSP)*. IEEE, Oct. 2021. DOI: 10.1109/mlsp52302.2021.9596143.

[132] Nathaniel Sweeney et al. “Insect Identification in Pulsed Lidar Images Using Changepoint Detection Algorithms”. In: *2023 Intermountain Engineering, Technology and Computing (IETC)*. IEEE, May 2023. DOI: 10.1109/ietc57902.2023.10152205.

[133] Yann LeCun, Yoshua Bengio, and Geoffrey Hinton. “Deep learning”. In: *Nature* 521.7553 (May 2015), pp. 436–444. DOI: 10.1038/nature14539.

[134] Trevor Vannoy and Whitaker Bradley. *Dataset for Insect Detection Remote Sensing*. 2023. DOI: 10.5281/ZENODO.10055762.

[135] Martin Tauc. "Scanning wing-beat-modulation LIDAR for insect studies". MA thesis. Montana State University, Apr. 2017. URL: https://scholarworks.montana.edu/xmlui/handle/1/12810.

[136] Haibo He and E. A. Garcia. "Learning from Imbalanced Data". In: *IEEE Transactions on Knowledge and Data Engineering* 21.9 (Sept. 2009), pp. 1263–1284. DOI: 10.1109/tkde.2008.239.

[137] Antonio R. S. Parmezan et al. "Changes in the wing-beat frequency of bees and wasps depending on environmental conditions: a study with optical sensors". In: *Apidologie* 52.4 (June 2021), pp. 731–748. DOI: 10.1007/s13592-021-00860-y.

[138] Douglas L. Altshuler et al. "Short-amplitude high-frequency wing strokes determine the aerodynamics of honeybee flight". In: *Proceedings of the National Academy of Sciences* 102.50 (Dec. 2005), pp. 18213–18218. DOI: 10.1073/pnas.0506590102.

[139] Jason T. Vance et al. "Hovering Flight in the Honeybee *Apis mellifera*: Kinematic Mechanisms for Varying Aerodynamic Forces". In: *Physiological and Biochemical Zoology* 87.6 (Nov. 2014), pp. 870–881. DOI: 10.1086/678955.

[140] Manfred R. Schroeder. "Period Histogram and Product Spectrum: New Methods for Fundamental-Frequency Measurement". In: *The Journal of the Acoustical Society of America* 43.4 (Apr. 1968), pp. 829–834. DOI: 10.1121/1.1910902.

[141] Chris Seiffert et al. "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance". In: *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans* 40.1 (Jan. 2010), pp. 185–197. DOI: 10.1109/tsmca.2009.2029559.

[142] S.C. Olhede and A.T. Walden. "Generalized Morse wavelets". In: *IEEE Transactions on Signal Processing* 50.11 (Nov. 2002), pp. 2661–2670. DOI: 10.1109/tsp.2002.804066.

[143] Trevor Vannoy, Nathaniel Sweeney, and Bradley Whitaker. *BMW-lab-MSU/insect-detection-remote-sensing-mdpi: v1.0.0*. 2023. DOI: 10.5281/ZENODO.10055809.

[144] Abhinav Maurya. "Bayesian optimization for predicting rare internal failures in manufacturing processes". In: *2016 IEEE International Conference on Big Data (Big Data)*. IEEE, Dec. 2016. DOI: 10.1109/bigdata.2016.7840827.

[145] Diederik P. Kingma and Jimmy Ba. *Adam: A Method for Stochastic Optimization*. 2014. DOI: 10.48550/ARXIV.1412.6980.

[146] Vincent Runge et al. "gfpop: an R Package for Univariate Graph-Constrained Change-Point Detection". In: (2020). DOI: 10.48550/ARXIV.2002.03646. URL: https://arxiv.org/abs/2002.03646.

[147] Toby Dylan Hocking et al. "Constrained Dynamic Programming and Supervised Penalty Learning Algorithms for Peak Detection in Genomic Data". In: *J. Mach. Learn. Res.* 21.1 (Jan. 2020). ISSN: 1532-4435.

[148] Gary M. Weiss. "Mining with rarity". In: *ACM SIGKDD Explorations Newsletter* 6.1 (June 2004), pp. 7–19. DOI: 10.1145/1007730.1007734.

[149] Davide Chicco and Giuseppe Jurman. "The advantages of the Matthews correlation coefficient (MCC) over F1 score and accuracy in binary classification evaluation". In: *BMC Genomics* 21.1 (Jan. 2020). DOI: 10.1186/s12864-019-6413-7.

[150] Thomas M Cover. *Elements of information theory*. John Wiley & Sons, 1999.

[151] Brian C. Ross. "Mutual Information between Discrete and Continuous Data Sets". In: *PLoS ONE* 9.2 (Feb. 2014). Ed. by Daniele Marinazzo, e87357. DOI: 10.1371/journal.pone.0087357.

[152] Peter H. Westfall. "Kurtosis as Peakedness, 1905–2014. *R.I.P.*" In: *The American Statistician* 68.3 (July 2014), pp. 191–195. DOI: 10.1080/00031305.2014.917055.

[153] Aharon Azulay and Yair Weiss. "Why do deep convolutional networks generalize so poorly to small image transformations?" In: *JMLR 20(184) 1-25 2019* (May 2018). DOI: 10.48550/ARXIV.1805.12177. arXiv: 1805.12177 [cs.CV].

[154] Fawwaz T. Ulaby and Umberto Ravaioli. *Fundamentals of Applied Electromagnetics*. Seventh. Pearson, 2015. ISBN: 978-0-13-335681-6.

[155] Milad Ghasri and Mojtaba Maghrebi. "Factors affecting unmanned aerial vehicles' safety: A post-occurrence exploratory data analysis of drones' accidents and incidents in Australia". In: *Safety Science* 139 (July 2021), p. 105273. DOI: 10.1016/j.ssci.2021.105273.

[156] Florin-Lucian Chiper et al. "Drone Detection and Defense Systems: Survey and a Software-Defined Radio-Based Solution". In: *Sensors* 22.4 (Feb. 2022), p. 1453. DOI: 10.3390/s22041453.

[157] Divy Raval et al. "Convolutional Neural Networks for Classification of Drones Using Radars". In: *Drones* 5.4 (Dec. 2021), p. 149. ISSN: 2504-446X. DOI: 10.3390/drones5040149.

[158] Subroto Singha and Burchan Aydin. "Automated Drone Detection Using YOLOv4". In: *Drones* 5.3 (Sept. 2021), p. 95. DOI: 10.3390/drones5030095.

[159] Yoojeong Seo, Beomhui Jang, and Sungbin Im. "Drone Detection Using Convolutional Neural Networks with Acoustic STFT Features". In: *2018 15th IEEE International*

*Conference on Advanced Video and Signal Based Surveillance (AVSS)*. IEEE, Nov. 2018. DOI: 10.1109/avss.2018.8639425.

[160] David Shorten et al. "Localisation of Drone Controllers from RF Signals using a Deep Learning Approach". In: *Proceedings of the International Conference on Pattern Recognition and Artificial Intelligence*. ACM, Aug. 2018. DOI: 10.1145/3243250.3243272.

[161] Marcus Hammer et al. "Lidar-based detection and tracking of small UAVs". In: *Emerging Imaging and Sensing Technologies for Security and Defence III; and Unmanned Sensors, Systems, and Countermeasures*. Ed. by Markus Mueller. SPIE, Oct. 2018. DOI: 10.1117/12.2325702.

[162] Jaakko Marin et al. "Neural networks in the pursuit of invincible counterdrone systems". In: *IEEE Potentials* 41.1 (Jan. 2022), pp. 14–21. DOI: 10.1109/mpot.2021.3113639.

[163] Hao Liu et al. "Drone Detection Based on an Audio-Assisted Camera Array". In: *2017 IEEE Third International Conference on Multimedia Big Data (BigMM)*. IEEE, Apr. 2017, pp. 402–406. DOI: 10.1109/bigmm.2017.57.

[164] Xiufang Shi et al. "Anti-Drone System with Multiple Surveillance Technologies: Architecture, Implementation, and Challenges". In: *IEEE Communications Magazine* 56.4 (Apr. 2018), pp. 68–74. ISSN: 1558-1896. DOI: 10.1109/mcom.2018.1700430.

[165] Stamatios Samaras et al. "Deep Learning on Multi Sensor Data for Counter UAV Applications—A Systematic Review". In: *Sensors* 19.22 (Nov. 2019), p. 4837. ISSN: 1424-8220. DOI: 10.3390/s19224837.

[166] Mohammed Jahangir and Chris Baker. "Robust Detection of Micro-UAS Drones with L-Band 3-D Holographic Radar". In: *2016 Sensor Signal Processing for Defence (SSPD)*. IEEE, Sept. 2016. DOI: 10.1109/sspd.2016.7590610.

[167] Samiur Rahman and Duncan A. Robertson. "Radar micro-Doppler signatures of drones and birds at K-band and W-band". In: *Scientific Reports* 8.1 (Nov. 2018). ISSN: 2045-2322. DOI: 10.1038/s41598-018-35880-9.

[168] L. Fuhrmann et al. "Micro-Doppler analysis and classification of UAVs at Ka band". In: *2017 18th International Radar Symposium (IRS)*. IEEE, June 2017. DOI: 10.23919/irs.2017.8008142.

[169] F. Fioranelli et al. "Classification of loaded/unloaded micro-drones using multistatic radar". In: *Electronics Letters* 51.22 (Oct. 2015), pp. 1813–1815. ISSN: 1350-911X. DOI: 10.1049/el.2015.3038.

[170] Ulzhalgas Seidaliyeva et al. "Detection of loaded and unloaded UAV using deep neural network". In: *2020 Fourth IEEE International Conference on Robotic Computing (IRC)*. IEEE, Nov. 2020. DOI: 10.1109/irc.2020.00093.

[171] Yong Zhang et al. "A New Estimation Method for Rotor Size of UAV Based on Peak Time-Shift Effect in Micro-Doppler Lidar". In: *Frontiers in Physics* 10 (Apr. 2022). DOI: 10.3389/fphy.2022.865240.

[172] Gianluca Ciattaglia et al. "UAV Propeller Rotational Speed Measurement through FMCW Radars". In: *Remote Sensing* 15.1 (Jan. 2023), p. 270. DOI: 10.3390/rs15010270.

[173] Peter Klaer et al. "An Investigation of Rotary Drone HERM Line Spectrum under Manoeuvering Conditions". In: *Sensors* 20.20 (Oct. 2020), p. 5940. ISSN: 1424-8220. DOI: 10.3390/s20205940.

[174] Gianluca Ciattaglia et al. "Determination of UAV propellers speed with FMCW Radar: comparison with accelerometer data". In: *2023 IEEE International Workshop on Technologies for Defense and Security (TechDefense)*. IEEE, Nov. 2023. DOI: 10.1109/techdefense59795.2023.10380909.

[175] Gianluca Ciattaglia et al. "Measuring UAV Propeller RPM with FMCW Radar: Validation with Calibrated Accelerometers". In: *2024 IEEE Sensors Applications Symposium (SAS)*. IEEE, July 2024, pp. 1–6. DOI: 10.1109/sas60918.2024.10636506.

[176] Peter John Rodrigo, Henning Engelbrecht Larsen, and Christian Pedersen. "CW coherent detection lidar for micro-Doppler sensing and raster-scan imaging of drones". In: *Optics Express* 31.5 (Feb. 2023), p. 7398. DOI: 10.1364/oe.483561.

[177] John J. Fike et al. "Preliminary analysis of drone propeller signals using wingbeat-modulation lidar". In: *Applications of Machine Learning 2023*. Ed. by Barath Narayanan Narayanan et al. SPIE, Oct. 2023, p. 29. DOI: 10.1117/12.2676175.

[178] Nahcmot, Ricardo de Azambuja, and Yann Bouteiller. *thecognifly/YAMSPy: Release 0.3.3*. Version v0.3.3. Oct. 2021. DOI: 10.5281/zenodo.5553212. URL: https://doi.org/10.5281/zenodo.5553212.