Analyzing Three-Dimensional Radar Voxel Data Using the Discrete Fourier Transform for SAEH Detection

P. Plodpradista*, J. M. Keller*, D. K. C. Ho*, M. Popescu**, A. Buck*

*University of Missouri, Electrical and Computer Engineering Department, Columbia, MO

**University of Missouri, Health Management and Informatics Department, Columbia, MO

The detection of side-attack explosive hazards (SAEHs) is a challenging task especially if the SAEHs are camouflaged. Three-Dimensional Radar is one of the most prominent sensors that has shown a great capacity for detecting concealed SAEHs. This system produces high-resolution volumetric images where each voxel's intensity represents the radar signal return at a specific point in three-dimensional space. It has the capability to enhance the signal response from a SAEH nested in camouflage materials and suppress the interference from the surroundings. Nevertheless, processing the radar data in the spatial domain has some limitations in differentiating SAEH from clutter objects. In this paper, we propose the use of the discrete Fourier transform (DFT) to analyze the voxel data and capture the spatial frequency characteristics of the radar signal from SAEHs. Through a machine learning approach, our proposed algorithm is able to identify the frequency signatures of SAEHs and to differentiate them from anomalies caused by the background or clutter. This approach yields a confidence value indicating the likelihood of a SAEH at a particular location. The detection ability of the proposed algorithm is demonstrated by receiver operating characteristic (ROC) curves generated using a dataset collected from a U.S. Army test site.

Keywords: Fourier transform, side-attack explosive, explosive hazard detection, 3-dimensional radar, voxel, class imbalance, Boosting

1. INTRODUCTION

In an application of side-attack explosive hazard (SAEH) detection, one challenge is detecting camouflaged targets. SAEH can be concealed with a multitude of material which hinders the detection of a traditional radar system. With the introduction of the Stalker system, a high-resolution three-dimensional radar system developed by the Pacific Northwest National Laboratory (PNNL)[1], we have the capability to reconstruct (beamform) the SAEH's energy located behind any concealment. However, the reconstruction (beamforming image) of SAEH still can be overcast by a camouflage's noise, thus a basic energy detection algorithm will not be able to produce a sufficient result. In our previous work[2], we show that prescreener in three-dimensional space can achieve results comparable to human-level performance. We continue our work by investigating the possibility of using the three-dimensional frequency information to improve our detection system. The main focus of this paper is the second-stage of the SAEH detection system, which is a process of determining the potential threats (alarms) collected in first-stage (prescreener) that are truly SAEHs. The second-stage is composed of two components: a feature extraction and a classifier model.

Our previous work with forward-looking ground penetrating (FLGPR) radar has shown that the Fourier transform enables the classifier to identify explosive hazards based on their frequency information[3]. By transforming a SAEH's response energy from the spatial domain to the frequency domain, a classifier is able to learn the signature frequency of SAEHs and improve the detection. Using this concept, we proposed the use of 3-Dimensional Fast Fourier Transform (3D FFT) for a feature extraction method. 3D FFT allows us to produce a numeric (feature) vector that represents the frequency information of volumetric image around each alarm's location.

As for the classifier model, we need to look at the nature of our application. In essence, detecting SAEHs is an anomaly detection problem, where the detection system attempts to identify outlier observations/samples. Therefore, most samples in a working SAEH dataset consist of non-target samples and only a small number of samples represent the target class. This can lead to an issue of the class imbalance problem where the classifier would bias toward the majority class (non-target class) and degrade the classification rate. We propose the use of Random Under-Sampling

Boost (RUSBoost) classifier as a countermeasure to the class imbalance issue. RUSBoost has two advantages over other technique designed for class imbalance problem: it has lower computation cost and it doesn't overfit[4].

In the next section, we give a background for the Stalker data and methodology of our proposed second-stage system, which composed of a feature extraction method and the classifier model.

2. METHODOLOGY

The working model for SAEH detection is composed of two stages. The first stage is the prescreening stage where the system examines the data and generates a list of locations for all potential threats (alarms). The second stage is where the system reassesses each alarm by gathering extensive information about the alarms (feature extraction) and then let an intelligence system (classifier) determine the confidence level of each alarm being a SAEH. Next, we give a brief overview of the method used in the first stage to process the Stalker data.

Prescreening Stalker Data

In the first stage, we start by processing the three-dimensional beamformed images captured by the Stalker radar system (Stalker data). As shown in Figure 1, the Stalker data is processed in a manner such that the voxel resolution is equal to 1 cm^3 and the frame orientation is paralleled to the earth. This process aligns the voxel coordinates in the Stalker frame to the UTM coordinates, which streamlines the global registration of alarm locations.

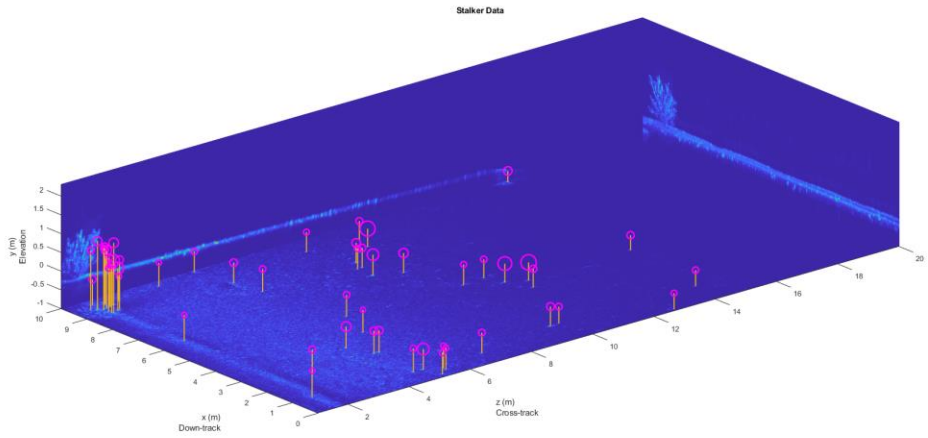

Figure 1. Shows an example frame from Stalker data. Magenta circles represent the alarms detected by the prescreener and the circle's size correspond to prescreener confidence value.

The algorithm (prescreener) used in the prescreening process is developed by CoVar[5]. The prescreener is based on a traditional constant false alarm rate (CFAR) algorithm combined with a synthetic occlusion array. The synthetic occlusion array measures the amount of occlusion between each voxel location and the Stalker's antenna which ranges from 0 to 1 (where 1 is extremely occluded). The prescreener picks the alarm locations (magenta circle in figure 1) using CFAR algorithm and produces a confidence value for each alarm by rescaling the CFAR energy with the occlusion value using equation 1.

$$conf_n = \frac{E_n}{\max\left((1 - occ_n), (0.2)\right)} \tag{1}$$

where $conf_n$ represents prescreener confidence value of n^{th} alarm

E_n represents CFAR energy at n^{th} alarm location

occ_n represents occlusion value at n^{th} alarm location

Finally, along with the prescreener confidence value, we extract a Stalker cube size of 30x30x30 cm^3 around each alarm location and proceed to the feature extraction step.

Feature Extraction: 3D Fast Fourier Transform Feature

The motivation for using the 3D Fast Fourier Transform (3D FFT) feature comes from our previous work[3] with FLGPR. In our previous work, we have concluded that the feature extracted from the two-dimensional Fast Fourier transform (2D FFT) performs better than those of one-dimensional or multi-layer Fourier transform. The reason is because the 2D FFT feature was able to capture the frequency information around the alarm location in both cross-track and down-track directions. We hypothesize that for a volumetric data (like those generated by the Stalker radar system), it is necessary to use 3D FFT for capturing additional frequency information on the elevation as well. Our hypothesis is partially confirmed by the visual comparison in Figure 2. In Figure 2, we show six examples of alarm cubes (three true targets and three false alarms) and their corresponding 3D FFT. The first column of images depicts three visually different true targets while their corresponding 3D FFT in the third column look very similar to each other. On the other hand, the fourth column of images shows that the corresponding 3D FFT of the false alarms are more dissimilar.

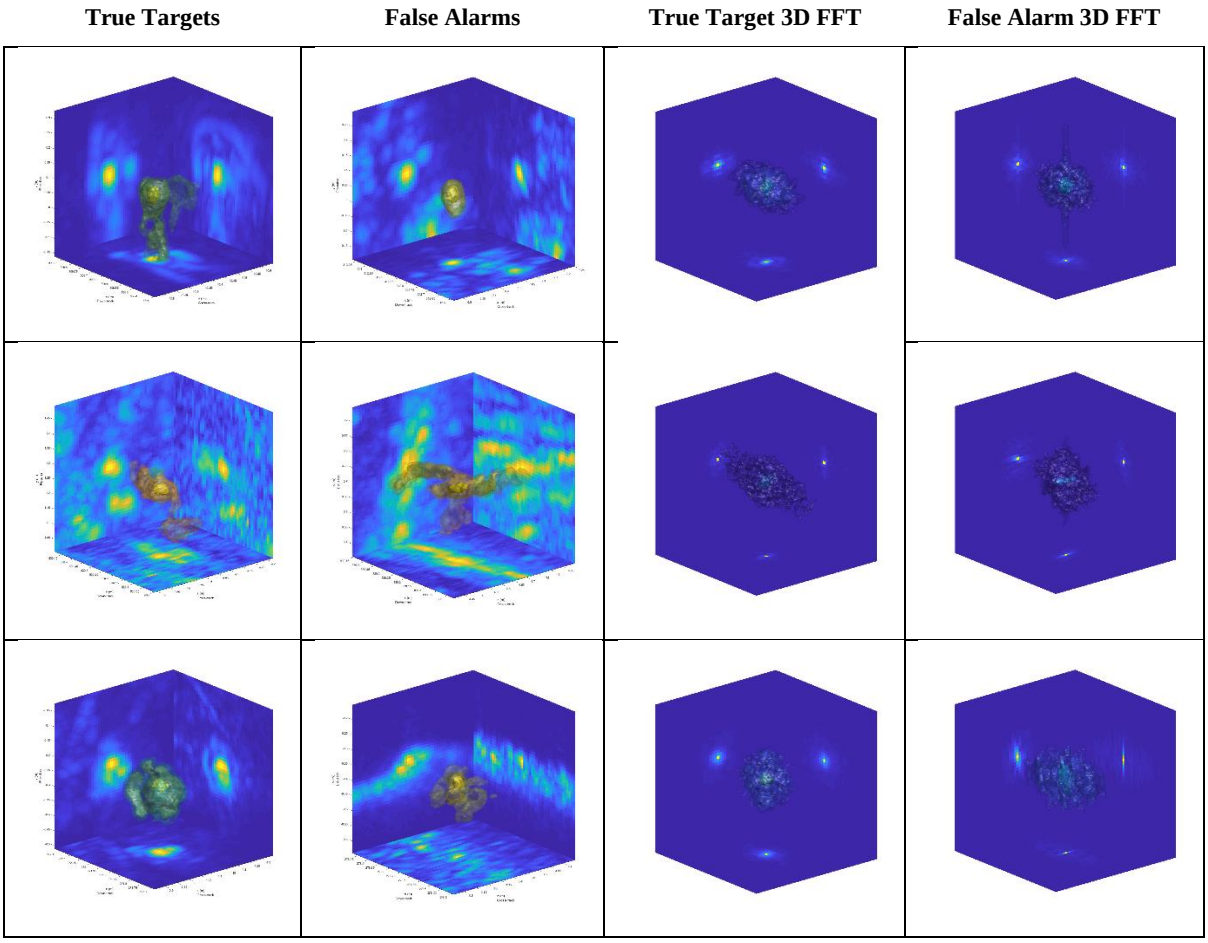

Figure 2. A visual comparison between three pairs of true target cubes and false alarm cubes. The first two columns illustrate the energy received by the Stalker system where the first column represents the magnitude cubes of true targets and the second column represents magnitude cubes of false alarms. The third and fourth columns represent the corresponding 3D FFT of the first and second columns, respectively.

We proposed the 3D FFT feature to be a representation of the input data's frequency information, where each element in the feature vector represents a frequency spectrum. The process of extracting the 3D FFT feature starts with transforming the Stalker data from the spatial domain to frequency domain by using Equation 2.

$$F_{k_1,k_2,k_3} = \sum_{n_1=0}^{N_1-1} \omega^{k_1 \frac{n_1}{N_1}} \sum_{n_2=0}^{N_2-1} \omega^{k_2 \frac{n_2}{N_2}} \sum_{n_3=0}^{N_3-1} \omega^{k_3 \frac{n_3}{N_3}} X_{n_1,n_2,n_3} \ , \qquad \omega = e^{-2\pi i} \tag{2}$$

where F represents spatial frequency

X represents voxel intensity

$\{N_1, N_2, N_3\}$ represent the dimensions of input data

Due to the symmetry property of Fourier transform, half of the transformed data are duplicate and can be removed for efficiency. However, it would inefficient to directly use the transformed data as the feature vector. In an attempt to create a more efficient feature vector, we define each feature element as a representation of a frequency bandwidth as shown in Equation 3.

$$\begin{gathered} FeatureVector = \left[Feat_1 \cdots Feat_v \cdots Feat_{\left\lfloor \frac{N_1}{B} \right\rfloor \cdot \left\lfloor \frac{N_2}{B} \right\rfloor \cdot \left\lfloor \frac{\frac{N_3}{2}}{B} \right\rfloor} \right] \\ Feat_v = S_{i,j,k} = \sum_{\hat{\imath}=1}^{B} \sum_{\hat{\jmath}=1}^{B} \sum_{\hat{k}=1}^{B} \left| F_{(i-1)B+\hat{\imath},(j-1)B+\hat{\jmath},(k-1)B+\hat{k}} \right|, \\ i \in \left\{1,2,\dots,\left\lfloor \frac{N_1}{B} \right\rfloor\right\}, j \in \left\{1,2,\dots,\left\lfloor \frac{N_2}{B} \right\rfloor\right\}, k \in \left\{1,2,\dots,\left\lfloor \frac{N_3/2}{B} \right\rfloor\right\}, \\ v = i + \left(\left\lfloor \frac{N_1}{B} \right\rfloor \cdot (j-1)\right) + \left(\left\lfloor \frac{N_1}{B} \right\rfloor \cdot \left\lfloor \frac{N_2}{B} \right\rfloor \cdot (k-1)\right) \end{gathered} \tag{3}$$

where B represent width of frequency band and v represent index of feature element.

Classifier: Random Under-Sampling Boost

Random Under-Sampling Boost (RUSBoost) was proposed by Seiffert et al.[6] as a classifier that mitigates the issue of class imbalance in the dataset. The class imbalance issue occurs when a training dataset contains samples (data points) of one class (majority class) that greatly outnumber another class (minority class). This issue causes a classifier trained from such a dataset to bias toward the majority class, which leads to a poor classification rate of the minority class and ultimately renders the classifier ineffective. Two common approaches used in the literature to handle the class imbalance issue are over-sampling the minority class and under-sampling the majority class. Over-sampling is a technique used to create new data points for the minority class so that it contains the same number of data points as the majority class. While the oversampling approach does not lose any information in the dataset, it increases the training cost of a classifier since it increases the size of the training data. Additionally, at least one study has shown that oversampling through duplicating the data points in the minority class could lead to overfitting[4]. On the other hand, undersampling is a technique used to remove data points from the majority class until all classes are balanced. Undersampling has the opposite effect of oversampling. It reduces the training cost and it does not cause overfitting. However, there is information lost during undersampling since it removes a large portion of data from the majority class which could diminish the classifier's accuracy. RUSBoost, which balances the dataset through the (random) undersampling technique, is able to eliminate this drawback by incorporating a boosting algorithm called AdaBoost.

Adaptive Boosting or AdaBoost is a meta-learning algorithm that creates an ensemble of weak learners by iteratively adapting the weight of training data and generates a new learner in each iteration. In the first iteration of AdaBoost, a new weak learner is trained using training data with uniform weights. Afterward, the pseudo-loss of the weak learner is calculated and the weights of the data points in training data are adapted accordingly. By increasing the weight of mislabeled data points and decreasing the weight of correctly labeled data points, the weak learners trained in following iterations are more likely to correctly classify the data points that were previously mislabeled. Once the algorithm reaches the specified maximum iterations, the training process is completed and the ensemble is composed of all weak learners that have been generated. During testing, the ensemble classifies the testing sample through the weighted votes of all generated weak learners. For complete details of the AdaBoost algorithm, please refer to Freund's work[7]. Using the AdaBoost approach, RUSBoost is able to minimize the loss of information in the majority class by creating an ensemble of weak learners from multiple instances of random undersampling. During each iteration in the

training process, a uniquely balanced dataset is generated by randomly undersampling the data points in the majority class and is used to train a weak learner. Therefore, a RUSBoost model that consists of multiple weak learners is learned from a more diverse sampling of data from the majority class.

In Table 1, we describe the process of training the RUSBoost model. Please note that the trained model $H(x)$ does not predict a class label but instead gives the likelihood of data point x belong to class y.

Table 1. Shows a training algorithm for RUSBoost.

RUSBoost Training Algorithm

Given:

-Training dataset *D* which consists of *N* pair of data points $\boldsymbol{X}$ and its labels $\boldsymbol{Y}$

where $\boldsymbol{X} = \{\boldsymbol{x_1}, \boldsymbol{x_2}, \dots, \boldsymbol{x_n}, \dots, \boldsymbol{x_N}\}$ and $\boldsymbol{Y} = \{\boldsymbol{y_1}, \boldsymbol{y_2}, \dots, \boldsymbol{y_n}, \dots, \boldsymbol{y_N}\}$

-Number of weak learner *T*

-Initialize the weight of each data point $\boldsymbol{W_1} = \{\frac{\boldsymbol{1}}{\boldsymbol{N}}, \frac{\boldsymbol{1}}{\boldsymbol{N}}, \dots, \frac{\boldsymbol{1}}{\boldsymbol{N}}\}$

- For t =1, 2, …, T
  1. Generate a balanced dataset $\widehat{\boldsymbol{D}}_{\boldsymbol{t}}$ by random subsampling of the majority class
  2. Train a weak learner $\boldsymbol{h_t}: \widehat{\boldsymbol{X}} \times \widehat{\boldsymbol{Y}} \rightarrow [\boldsymbol{0}, \boldsymbol{1}]$ with dataset $\widehat{\boldsymbol{D}}_{\boldsymbol{t}}$ and weights $\boldsymbol{W_t}$
  3. Calculate the weight parameter

     $$\boldsymbol{\alpha_t} = \frac{\boldsymbol{\varepsilon_t}}{\boldsymbol{1} - \boldsymbol{\varepsilon_t}}$$

     where $\boldsymbol{\varepsilon_t} = \frac{\boldsymbol{1}}{\boldsymbol{2}} \sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \sum_{\boldsymbol{y \neq y_n}} \boldsymbol{W_{t,n}} (\boldsymbol{1} - \boldsymbol{h_t}(\boldsymbol{x_n}, \boldsymbol{y_n}) + \boldsymbol{h_t}(\boldsymbol{x_n}, \boldsymbol{y}))$
  4. Update the weight of each data point $\boldsymbol{x_n}$,

     $$\boldsymbol{W_{t+1,n}} = \frac{\boldsymbol{W_{new,n}}}{\sum_{\boldsymbol{n=1}}^{\boldsymbol{N}} \boldsymbol{W_{new,n}}}$$

     where $\boldsymbol{W_{new,n}} = \boldsymbol{W_{t,n}} \boldsymbol{\alpha_t}^{\frac{\boldsymbol{1}}{\boldsymbol{2}}(\boldsymbol{1} - \boldsymbol{h_t}(\boldsymbol{x_n}, \boldsymbol{y_n}) + \boldsymbol{h_t}(\boldsymbol{x_n}, \boldsymbol{y \neq y_n}))}$

Generate a RUSBoost model as the ensemble of weak learners $\boldsymbol{H}(\boldsymbol{x}) = \sum_{\boldsymbol{t=1}}^{\boldsymbol{T}} \boldsymbol{h_t}(\boldsymbol{x}, \boldsymbol{y}) \boldsymbol{log} \frac{\boldsymbol{1}}{\boldsymbol{\alpha_t}}$

Weak Learner: Classification and Regression Trees

In this paper, we used the Matlab implementation of Breiman's Classification And Regression Trees (CART)[8, 9] as the weak learners for the RUSBoost classifier. CART is a type of binary decision tree that uses the Gini Diversity Index (GDI) to grow (split) the decision tree. GDI is a measure used to evaluate the purity of a tree's node. The purity of a tree's node indicates the degree of the class mixture within that node and the node is considered pure (GDI=0) when it contains data points of a single class. For a problem with C classes, the GDI of node n is calculated using Equation 4.

$$GDI_n = 1 - \sum_{c=1}^{C} {p_c}^2$$
$$p_c = \frac{|D_c|}{|D|} \tag{4}$$

where D represents the data points in node n and D_c represents the data points in node n that belong to class $\boldsymbol{c}$. To grow a decision tree using the CART algorithm, the data is optimally split in the top node which results in two child nodes. These child nodes will be used as the parent nodes for the next split. By recursively splitting the node a decision tree is grown and the process continues until all child nodes are pure. The optimal split is obtained by searching for threshold t of feature f that maximizes the impurity gain (ΔI). The impurity gain of a split at node n can be calculated using Equation 5.

$$\Delta I_n = GDI_n - \left(\frac{|D_L|}{|D|} \times GDI_L\right) - \left(\frac{|D_R|}{|D|} \times GDI_R\right)$$
$$D_L = \{x_i : x_i^f \leq t\},$$
$$D_R = \{x_i : x_i^f > t\}, \quad (5)$$
$$D = \{x_1, x_2, x_3, \dots\}$$

where x_i^f represents the value of the f^{th} feature on data point x_i. However, the data points are weighted during the RUSBoost training process and the calculation of impurity gain needs to take that into account. By including the weight of data points (w_i) as a prior probability(Pr), Equation 5 is modified to Equation 6.

$$\Delta I_n = Pr \times GDI_n - (Pr_L \times GDI_L) - (Pr_R \times GDI_R)$$
$$Pr = \sum_{i \in D} w_i \,,$$
$$Pr_L = \sum_{i \in D_L} w_i \,, \quad (6)$$
$$Pr_R = \sum_{i \in D_R} w_i$$

3. EXPERIMENTS

Experimental Data

The data and the ground truth are provided to us by the U.S. Army. The data was collected from 7 unique lanes in an arid U.S. Army test site. The total area cover by the data is 1,532,200 m^2 and there are 593 targets emplaced within the said area. CoVar provides us with the list of alarm locations detected by the prescreener and there are 783,784 alarms for this data. Finally, in Table 2, we present the composition of this data and give the statistical characteristics of each lane.

Table 2. List the data composition used in the experiments.

Lane	Coverage Area (m^2)	Number of Targets	Number of Alarms	Alarms per m^2	Alarms to Targets Ratio
A	217,200	72	119,182	0.55	1655.31:1
B	218,600	63	101,611	0.46	1612.87:1
C	220,800	24	45,119	0.20	1879.96:1
D	269,800	62	124,776	0.46	2012.52:1
E	55,000	37	33,117	0.60	895.05:1
F	334,800	179	263,921	0.79	1474.42:1
G	216,000	156	96,058	0.44	615.76:1
Total	**1,532,200**	**593**	**783,784**		

Experimental Setup

The target emplacement on each lane represents a different scenario encountered in the real world. Therefore, for testing our proposed method, we use a lane based cross-validation scheme which yields an overall performance. As

shown in Table 3, the classifier models used for testing each fold/lane are trained by the data from the other six folds/lanes and the final result is the concatenation of all seven folds.

Table 3. Shows an algorithm for lane-based cross-validation.

Lane-based Cross-validation

Given:
- $\{D_A, D_B, D_C, D_D, D_E, D_F, D_G\}$ represent data extracted from each lane
- *Results* = ∅

- For *TestLane* = A to G
  1. *TestData* = $D_{TestLane}$
  2. *TrainData* = ∅
  3. For *TrainLane* = A to G
     - If *TrainLane* ≠ *TestLane*
       TrainData = *TrainData* ∪ $D_{TrainLane}$ *%add data to the training set*
  4. For trial = 1 to 10 *%training and testing are repeat 10 times*
     - $\boldsymbol{H_{trial}(x)}$ = Train(*TrainData*)
     - $\boldsymbol{Conf_{trial}}$ = Test($\boldsymbol{H_{trial}(x)}$, *TestData*) *%output of RUSBoost is the target class confidence level*
  5. $Result_{TestLane}$ = $\overline{\boldsymbol{Conf}}$ *%result is the average confidence level of 10 trials*
  6. *Results* = *Results* ∪ $Result_{TestLane}$
- Generates ROC curve from *Results*

In our experiments, we set the RUSBoost model to be an ensemble of 100 CART decision trees where each decision tree is grown until all remaining nodes are pure. Additionally, in order to obtain a more consistent result from a RUSBoost model, we repeat the process of training and testing ten times and the result in each fold is the average confidence value.

In this paper, we use the Receiver Operating Characteristic (ROC) curve to measure the result from our experiments. We use the ground truth provided by the U.S. Army to identify whether each alarm hit is a positive detection (PD) or a false alarm (FA) at a certain detection threshold. Next, we arrange the results and generate the ROC curve based on the PD and FA.

Experimental Results

In the first set of experiments, we investigate the effect of both the size of the cube from which the 3D FFT feature is extracted and the frequency bandwidth of the 3D FFT. Since the size of the SAEHs in our dataset are roughly 30cm in diameter, we experiment with the cube size equal to 30x30x30 cm^3 and 40x40x40 cm^3, and a frequency bandwidth range from 2 to 6. The results in Figure 3 show that for any given FAR, the extracted 3D FFT feature from a cube size equal to 30x30x30 cm^3 yields 6-10% higher PD than 40x40x40 cm^3. The bigger data cube yields a less discriminative feature vector due to the fact it contains more background data, thus the 3D FFT feature is more saturated with the background frequencies. Next, we analyze the impact of the frequency bandwidth parameter. The results show that the 40x40x40 cm^3 cube size is more sensitive to the bandwidth parameter, as smaller bandwidths have better ROC curves. These results are to be expected since increasing the frequency bandwidth means less resolution for the 3D FFT feature. However, the 30x30x30 cm^3 cube size shows dissimilar results. With the bandwidth ranging from 2 to 5, the ROC curves of the 30x30x30 cm^3 cubes are almost identical to each other. Only with the bandwidth set to 6 does the ROC curve show a noticeable drop in PD at the lower FAR.

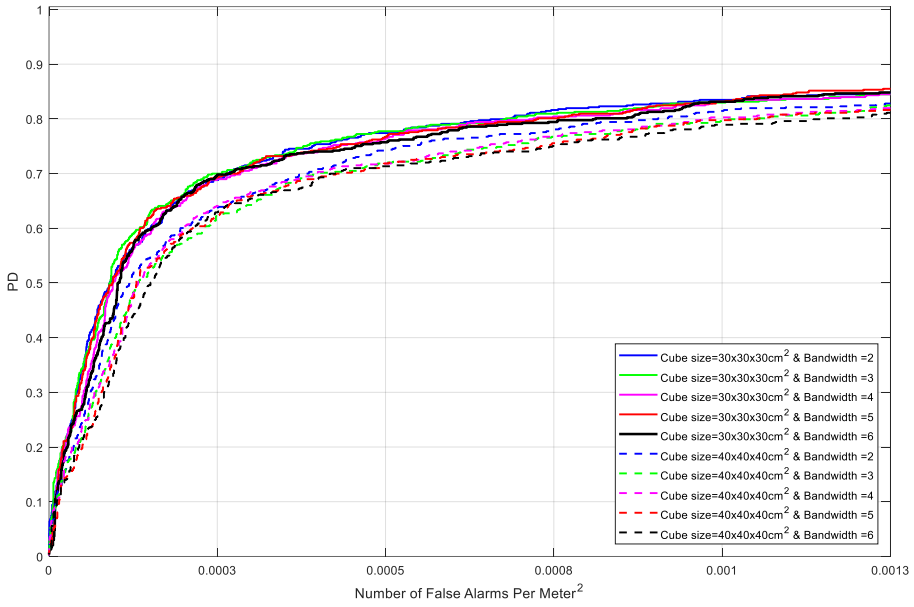

Figure 3. Shows the comparison of results between the 2 cube sizes and 5 bandwidth variations of the 3D FFT feature.

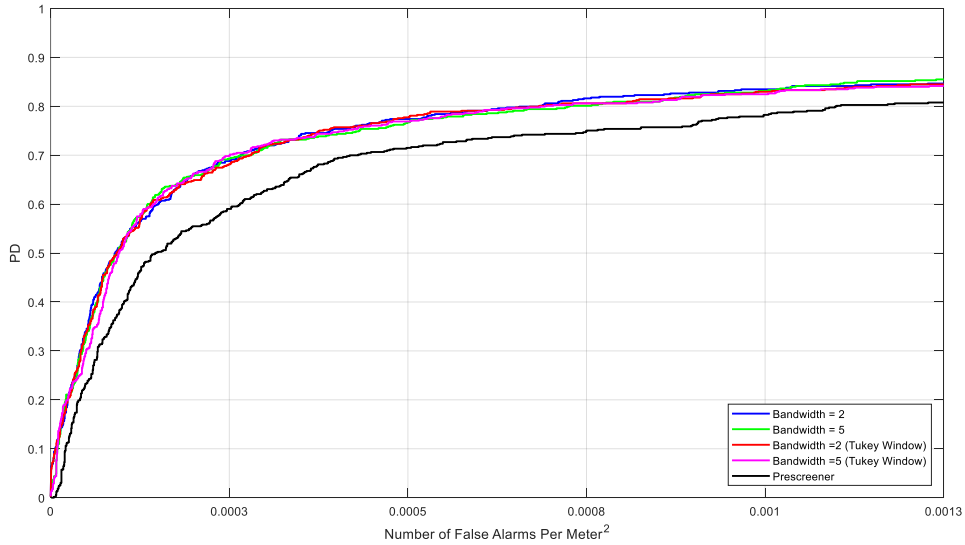

Figure 4. Shows the impact of Tukey window and the improvement over prescreener.

For the final set of experiments, we examine the effect of applying a Tukey window to the data cube and compare the results of the 3D FFT feature to the prescreener. The motivation for this experiment is an attempt to improve the performance of the 3D FFT feature. In the field of signal processing, it is a common practice to apply a window function to an input signal in order to mitigate the artificial high frequency that occurs from the Fourier transform. In this experiment, we apply a 3D tapered cosine (Tukey) window to the data cube and extract the 3D FFT feature with bandwidth parameter equal to 2 and 5 since both configurations yield similar results in previous experiments despite the difference in frequency resolution. The results in Figure 4 show that applying Tukey windows has no effect when the bandwidth is set to 2, while it has a slightly diminishing result when the bandwidth is set to 5. We suspect the

reason that the Tukey window has no effect is due to the size of data cube that we use. Since the cube is only 30x30x30 cm^3, it is likely that the voxel intensities near the edge of the cube are similar and there is no artificial high frequency. Still, when compared to the prescreener, all configurations of the 3D FFT feature have increases of 5-10% in PD for any given FAR.

4. CONCLUSION

In this paper, we showed that RUSBoost allows us to build an effective classifier model from a highly unbalanced dataset and the 3D FFT feature is able to identify the signature frequency of SAEHs which improves the detection rate over the prescreener. We also showed that extracting the 3D FFT feature from a proper data cube size has two advantages. First, the 3D FFT feature is insensitive to the bandwidth parameter. Therefore, we are able to increase the bandwidth from 2 to 5 which reduces the size of the feature vector from 1,800 dimensions to 108 dimensions without lowering the discriminative power. Second, it does not require pre-processing the data. By sizing the data cube to perfectly fit the SAEH, there is no artificial high frequency from the Fourier transform, hence it is meaningless to apply a window function.

For future work, we wish to improve the 3D FFT feature by using it in conjunction with other 3D features. We also seek the possibility of using the 3D FFT feature with Stalker data in the complex form.

ACKNOWLEDGEMENTS

This work was funded by Army Research Office grant number 57940-EV to support the U.S. Army RDECOM CERDEC NVESD.

REFERENCES

[1] D. M. Sheen, T. E. Hall, D. L. McMakin *et al.*, "Three-dimensional radar imaging techniques and systems for near-field applications." 9829, 12.

[2] A. Buck, J. M. Keller, M. Popescu *et al.*, "Target detection in high-resolution 3D radar imagery." 10182, 10.

[3] D. Shaw, K. C. Ho, K. Stone *et al.*, "Explosive hazard detection using MIMO forward-looking ground penetrating radar." 9454, 94540Z-94540Z-14.

[4] C. Drummond, and R. C. Holte, [C4.5, Class Imbalance, and Cost Sensitivity: Why Under-Sampling beats OverSampling], (2003).

[5] M. Crosskey, P. Wang, R. Sakaguchi *et al.*, "Physics-based data augmentation for high frequency 3D radar systems."

[6] C. Seiffert, T. M. Khoshgoftaar, J. V. Hulse *et al.*, "RUSBoost: A Hybrid Approach to Alleviating Class Imbalance," IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans, 40 (1), 185-197 (2010).

[7] Y. Freund, and R. E. Schapire, "A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting," Journal of Computer and System Sciences, 55 (1), 119-139 (1997).

[8] L. Breiman, [Classification and regression trees] Wadsworth International Group, Belmont, Calif. (1984).

[9] Matlab, "Fit binary classification decision tree for multiclass classification," Matlab R2017b, <https://www.mathworks.com/help/stats/fitctree.html> (02/07/2018).