An Approach to detecting crowd anomalies for entrance and checkpoint security

Holly Zelnio
An Approach to Detecting Crowd Anomalies for Entrance and Checkpoint Security

by

Holly Zelnio

B.S. Wright State University 2010

A thesis submitted in partial fulfillment of the requirements for the degree of Master of Science in the Chester F. Carlson Center for Imaging Science College of Science Rochester Institute of Technology

October 12, 2012

Signature of the Author

Accepted by

Coordinator, M.S. Degree Program

Date
The M.S. Degree Thesis of Holly Zelnio has been examined and approved by the thesis committee as satisfactory for the thesis required for the M.S. degree in Imaging Science.

Dr. Harvy Rhody, Thesis Advisor

Dr. Michael Long

Dr. Anthony Vodacek

Date
THESIS RELEASE PERMISSION
ROCHESTER INSTITUTE OF TECHNOLOGY
COLLEGE OF SCIENCE
CHESTER F. CARLSON CENTER FOR IMAGING SCIENCE

Title of Thesis:
An Approach to Detecting Crowd Anomalies for Entrance and Checkpoint Security

I, Holly Zelnio, hereby grant permission to Wallace Memorial Library of R.I.T. to reproduce my thesis in whole or in part. Any reproduction will not be for commercial use or profit.

Signature ___________________________________________ Date __________

3
An Approach to Detecting Crowd Anomalies for Entrance and Checkpoint Security

by

Holly Zelnio

Submitted to the
Chester F. Carlson Center for Imaging Science
College of Science
in partial fulfillment of the requirements
for the Master of Science Degree
at the Rochester Institute of Technology

Abstract

This thesis develops an approach for detecting behavioral anomalies using tracks of pedestrians, including specified threat tracks. The application area is installation security with focus on monitoring the entrances of these installations. The approach specifically allows operator interaction to specify threats and to interactively adjust the system parameters depending on the context of the situation. This research has discovered physically meaningful features that are developed and organized in a manner so that features can be systematically added or deleted depending on the situation and operator preference. The features can be used with standard classifiers such as the one class support vector machine that is used in this research. The one class support vector machine is very stable for this application and provides significant insight into the nature of its decision boundary. Its stability and ease of system use stems from a unique automatic tuning approach that is computationally efficient and compares favorable with competing approaches. This automatic tuning approach is believed to be novel and was developed as part of this research. Results are provided using both measured and synthetic data.
## Contents

1 Introduction ........................................... 7

2 Background ............................................ 10
   2.1 Anomaly Detection .................................. 10
   2.2 Feature Extraction .................................. 11
   2.3 Classifiers .......................................... 12
   2.4 Video Anomaly Detection ............................ 15
   2.5 Summary ............................................. 16

3 Approach .............................................. 18

4 Feature Extraction ..................................... 23
   4.1 Feature Structure ................................... 23
      4.1.1 Single Track Features ........................... 24
      4.1.2 Comparative Track Features ................. 24
   4.2 Features ............................................. 25
      4.2.1 Single Features ................................ 26
      4.2.2 Comparison Features ........................... 27
   4.3 Summary ............................................. 28

5 Classifier .............................................. 30
   5.1 SVM .................................................. 30
      5.1.1 SVM Formulation ................................. 30
      5.1.2 Simple SVM Example ............................ 35
      5.1.3 SVM parameters .................................. 37
      5.1.4 Tuning SVM parameters: Relationship with decision boundary and performance ................. 38
   5.2 Conclusion .......................................... 42
6 Experiments

6.1 Data .................................................. 45
   6.1.1 Measured Data: University of California at San Diego (UCSD) data base .................................. 45
   6.1.2 Simulated data .................................. 45

6.2 Results .............................................. 47
   6.2.1 Measured Data Results ....................... 47
   6.2.2 Simulated Data Results ...................... 52

6.3 Summary ............................................ 55

7 Conclusions and Recommendations 61

7.1 Conclusions ....................................... 61
7.2 Future Recommendations ....................... 62

Appendices 65

A Code Description 65
   A.1 Anomaly Detection Description using Simulated Tracks ............. 65

B Source Code 70
   B.1 Feature Extraction ................................ 70
      B.1.1 Calls and runs all the features .................... 70
      B.1.2 Filter Feature Extractors – input: tracks, output: tracks ... 71
      B.1.3 Reduce Feature Extractors – input: tracks, output: vectors ... 71
      B.1.4 Filter Feature Extractors – input: vectors, output: vectors ... 73
      B.1.5 Reduce Feature Extractors – input: vectors, output: scalars ... 74
      B.1.6 Comparison Features – input: 2 sets of tracks, output: scalar ... 75
   B.2 Classifier functions .............................. 76
      B.2.1 Centering and Normalization ..................... 76
      B.2.2 Find Sigma ....................................... 77
      B.2.3 One class SVM using RBF – Training ............... 79
      B.2.4 Calculate SVM Internal Parameters ................. 80
      B.2.5 One class SVM using RBF – Testing ............... 81
   B.3 Experiment with Simulated Data .................. 82
      B.3.1 Experiment Script ................................ 82
      B.3.2 Plots ............................................. 86
Chapter 1

Introduction

Anomaly detection is an area of research that is important to a number of application areas including cyber-intrusion detection, fraud detection, medical anomaly detection, industrial damage detection, textual anomaly detection, and sensor networks [1–3]. One of the difficulties of anomaly detection is understanding what is normal. In pedestrian anomaly detection, it is important to understand normal crowd behavior. Crowd modeling has important applications in social study, civil planning, crowd control, and security [4]. In the areas of crowd control and security, anomaly detection is very important to try to either prevent incidents from happening or to mitigate damage after an incident has happened [5]. Security is an important area to everyone; but it is extremely important to people who provide security for a living or are at risk in performing their duties. One important aspect of security is security at installations, entrances, and checkpoints [1, 2, 6–9]. This type of security is important to many applications including military installations, sports and entertainment venues, gambling casinos, industrial installations, petroleum facilities, food services, banks, and libraries [10]. At entrances and checkpoints, the security can be enhanced by analyzing tracks of pedestrians approaching, entering, and leaving the installation. The need to use electronic means such as video cameras has been documented by the armed services as an important part of their overall security approach [6].

For security applications, it is important to be able to characterize what is normal and what is the threat. Categorization of threatening behavior is not always known beforehand and can likely evolve over time as the situation changes. For example, the Department of Homeland Security categorizes the threat risk with two threat levels. These levels are Elevated Threat Alert and Imminent Threat Alert. Both levels require a credible threat for a threat alert to be issued. The Department of Defense has their own method of categorizing threat risks, and the threat risks are categorized as the Force Protection Condition Measures that are used at military facilities. There are five levels of threat. These five levels are Normal, Alpha, Bravo, Charlie, and Delta. Each level represents an increased threat. Each level has
certain procedures that are followed. [11] The security behavior at gates is different depending on the threat situation.

Given the context of the situation, pedestrian tracks are categorized into three categories: normal tracks, threatening tracks, and anomalous tracks.

- Normal tracks are when the pedestrians are traveling in expected patterns approaching, entering, and exiting installations. These patterns are predominantly determined by extracting video tracks from videos and learning the patterns over time.

- Threatening tracks are tracks determined by the responsible security force to be explicitly threatening. These tracks are described as tracks that enter a no travel area or get too close to a barrier such as a security fence. These can also be described as tracks that travel erratically or at excessive speed. Further, the track attributes such as speed may be threatening conditioned on where the particular attribute is occurring. For example, a pedestrian traveling at high speed near the checkpoint may be threatening whereas a pedestrian traveling at the same speed a distance away from the checkpoint may be considered normal.

- Anomalous tracks are tracks that are different from normal tracks as defined by being outside the decision boundary of the classifier that determines the normal tracks. This definition is data dependent and somewhat arbitrary. The strategy for this research effort is to continually identify tracks living on the boundary to determine whether or not the track is threatening or allowable. If the track is threatening or not allowable, it is removed from the set of normal tracks and the decision boundary self adjusts to no longer consider the track as normal. In this way, an approach is provided that systematically and rigorously adapts to the situation using operator feedback to provide the context for a given situation.

The detection of anomalies is not at all straightforward due to a number of factors including the following:

- The definition of normal behavior is difficult and the difference between normal and anomalous behavior is fuzzy.

- Adversaries who may be sought as anomalous intentionally try to blend into the background.

- Normal behavior is context and time sensitive. It changes over space and time.
• The definition of an anomaly is application specific.

• Labeled data to train for anomalies are often not available.

• The data is often noisy and is variable, and it may be hard to discriminate between natural variability and anomalies. [3]

Due to the difficulty in defining anomalies, this thesis explores a methodology to closely couple the definition of the anomaly and the performance of the system with the operator. This close coupling is based on a two pronged approach: 1) the features that are used are physically based and are easily understandable by the operator, and 2) the classifier uses a decision boundary that intersects the most outlying tracks, hence, providing the operator with a clear definition of the boundary surrounding normal tracks.

The track features are space / time trajectories that are sampled at a constant rate corresponding to the frame rate of the video camera being used. The physical features available are limited to features that can be calculated from position, time, and position as a function of time. There are several features that can be calculated from tracks that are available in the literature. One of the contributions of this thesis is the development of a structured way of organizing and applying the features for the installation security problem [4]. In addition to this structure, the features are easily explainable in terms that make sense to the operator, and further, new features are easily added.

Classifiers, in general, are trained on samples from all classes of interest. However, with anomaly detection, usually only the normal samples are available for training. Several classifiers have been used for anomaly detection including clustering approaches, multi-class classifiers, and one-class classifiers. For this effort, the support vector machine (SVM) is chosen. In particular, the one class support vector machine (OC-SVM) has properties that make it useful for anomaly detection. These properties make it suited for use with a human operator for adaptively learning the decision boundary as the threat environment changes. The support vectors or tracks which lie on the decision boundary provide the operator with explicit examples of the normal tracks that are the closest to being anomalous, thus, making the abstract classification decision boundary concrete.
Chapter 2

Background

Anomalies in crowd detection are difficult to detect because defining normal in a crowd setting is challenging. Pedestrians can walk in erratic patterns and may have different goals for entering an area. If the normalcy of the scene is difficult to define it makes sense that finding anomalies in the scene would be difficult as well. There are several applications that use anomaly detection approaches as delineated in the introduction. Anomaly detection approaches will be reviewed in general and also reviewed specifically in the area of video surveillance. Next, the focus will turn to approaches that are specifically aligned with this thesis – those approaches that extract features from tracks, and approaches that require one class classifiers.

2.1 Anomaly Detection

Anomalies can be categorized into point, contextual, and collective [3]. Point anomalies are based on features extracted from single tracks and do not depend on any other information. These anomalies are different at the data instance level. To train a point anomaly detector, the features would be extracted from all of the tracks in the training set without any subcategories or dependencies. For example, the training data would be one large matrix with each track being a separate column and each feature being a row. Contextual anomalies are anomalous in specific contexts but may not be anomalous in general. These contexts can be spatial (e.g., occurring only at specific locations) or can be temporal (e.g., corresponding to a specific time of day). For example, a track may be anomalous if it exhibits that behavior in a particular region but may not be anomalous if that behavior is exhibited elsewhere. In this case, some features would be calculated only when the track traversed a designated region. Finally, collective anomalies occur as groups but may not be anomalous individually. For example, a track that stops at a particular location is not anomalous but would be anomalous if several tracks stopped at that particular location together at the same time. These types of anomalies are important as they may uncover useful relationships among the data. For example,
for collective anomalies, additional features would need to be defined to specifically extract relationships among the tracks. This thesis concentrates on both point and contextual anomalies. In particular, contextual anomalies are considered by allowing the operator to adapt the anomaly detector based on the context or threat level.

Anomaly detection approaches can also be sub-divided into the categories of supervised anomaly detection, semi-supervised anomaly detection, and unsupervised anomaly detection [3]. In supervised anomaly detection, both the normal data and anomaly data are labeled to allow the application of two class classification approaches. In semi-supervised approaches, part of the data is unlabeled and part of the data is labeled. For the anomaly detection case, semi-supervised approaches will be defined as labeled normal data and unlabeled anomaly data. In this case, approaches that characterize a single class are needed. Finally, in unsupervised approaches neither the normal data nor the anomalies are labeled, and hence, clustering type of approaches are necessary. For this thesis, semi-supervised techniques (normal tracks labeled, anomalies unlabeled) are investigated.

2.2 Feature Extraction

In order to perform a classification task, the first step usually is feature extraction. A feature extractor is a mathematical operator that is applied to the raw data that extracts useful information that can be fed to a classifier for classification. There are many ways of characterizing feature extractors; however, for the purpose of this thesis, the feature extractors are classified as either 1) physical or structural features or 2) abstract or mapping features [12–14]. As their name implies, physical or structural features are chosen based on their ability to describe the physical nature of the object or activity that they are characterizing. Hence, the feature is usually understandable by human domain experts as it has a basis in physics or geometry. On the other hand, abstract or mapping features are not, in general, understandable by humans and their value is not understood until they are used for the classification task. This thesis concentrates on physically motivated features as it is important that the operators understand the meaning of the features.

After choosing an initial set of features, often feature selection techniques are used to reduce the set of features fed into the classifier. Feature selection methods include techniques that 1) select subsets of the features or 2) combine features or transform the features into a lower dimensional space [15, 16]. Techniques that select subsets of features are grouped into three types: filters, wrappers, and embedded [17, 18]. Filters select a subset of features by studying the features themselves without
resorting to a classification step. Filter techniques may look at the correlation of features or may consider information measures to determine the number of features to keep. Wrapper techniques use classifiers to determine the best features to keep. In this case, the features that give the best classification results are chosen. In wrapper techniques, first feature selection is performed, and then, classification is performed to determine the utility of the selected features. Embedded techniques perform the feature selection and classification jointly often as part of an optimization process.

Techniques that transform data into lower dimensional spaces are termed dimensionality reduction techniques. These techniques include principle component analysis, factor analysis, projection pursuit, independent component analysis, and random projections [19]. These techniques often have strong theoretical foundations, but they can lose the meaning of the initial features as the features are transformed and/or combined into new features that can obscure their relationship to the original set of features. One technique to reduce the dimensionality of features without losing the meaning of features is feature clustering [20, 21]. In this case, the cluster can be represented by a typical feature or may have a common meaning.

This thesis does not focus on feature selection techniques, but it is observed that the subset selection techniques would be consistent with the spirit of the thesis. The subset techniques just reduce the number of features maintaining their physical meaning, hence, promoting operator understanding of the anomaly detection process.

2.3 Classifiers

Classifiers can be characterized in multiple different ways; however, for anomaly detection, a taxonomy developed by Tax [22] in his dissertation work for one class classifiers appears the most appropriate. In one class classifiers, it is assumed that training data is only available for the single class. This assumption describes the situation where normal tracks are available for training and anomalous tracks are not available which is consistent with the problem being addressed by this thesis. Tax divided the one class classifiers into the categories: density methods, reconstruction methods, and boundary methods.

Density methods are methods that attempt to approximate a probability density function, \( p(x|C_1) \), which is the probability of the data or features, \( x \), given the single class, \( C_1 \). Given the probability density function, anomalies can be defined by setting a threshold at a low probability of belonging to class \( C_1 \). The criteria for determining the “tail probabilities” that constitute anomalies is problematic, however. The estimation of the probability density can be performed either para-
metrically or non-parametrically. Parametric methods assume a parametric form for the probability density function. An example parametric method for density estimation is Gaussian density estimation [23]. For this method the data is used to estimate both the mean and covariance matrix which is sufficient to determine the distribution. Many times, however, the parametric form does not correctly characterize the distribution. The underlying data may be multi-modal or may be fundamentally different that the parametric density chosen. Another approach that is more flexible is called a mixture of Gaussians [24]. In this case, the underlying distribution is approximated by weighting and combining a number of Gaussians each with different means and covariance matrices. This case is more flexible as it can approximate multi-modal and other distributions with non-Gaussian shapes; however, it requires the estimation of the means and variances of each Gaussian in addition to the mixture components for each Gaussian. This classifier has been used to detect anomalies in video data of crowded scenes [25]. As a final example of density estimation, Parzen density estimators can be used. Parzen density estimators center a kernel at each data or feature point to ”fill in” the density between the points. The kernel can be any smoothing function. A popular one is the Gaussian probability density function. For the Parzen density estimator, a key parameter that must be chosen is the smoothing parameter. For the Gaussian probability density function, $\sigma$, the variance, is the smoothing parameter. The Parzen or kernel estimator has been used to find anomalies in video data [26]. For both the mixture of Gaussians and the Parzen estimator, each kernel is weighted so that the kernels sum to one to be consistent with the definition of a probability density function.

The reconstruction methods attempt to represent the data or features by fitting the data to a representation or some model of the data. These methods are often used to cluster data as they make assumptions of how the data clusters in either the feature space or some subspace. An anomaly is declared based on the error of the fit of the anomaly with the model. As in the density methods, there is a requirement to declare an anomaly based on a threshold; in this case, based on a threshold on the error. Again, the amount of error which constitutes an anomaly is problematic. In the density methods, the threshold is based on the probability that the anomaly belongs to the distribution of normal tracks. For the reconstruction methods, there is no principled interpretation of where the threshold should be placed. An example of a reconstruction method is the k-means algorithm. In this case, the data is represented by a mean value of $k$ subsets of the data or or $k$ clusters. The k-means algorithm iteratively determines the location of the $k$ means by measuring the distance of each point to the current locations of the $k$ means and then associating each point to the closest mean value. After association, the
mean is recomputed and this process continues until the $k$ mean values do not change. The representation of the data is simply the location of the $k$ means, and the determination of an anomaly is made when the distance from the closest mean or error is greater than some pre-determined value or threshold. Another method that is used to detect anomalies is principal component analysis or PCA [27]. In this case, the principal components are determined by performing an eigenvalue analysis of the covariance matrix of the data or feature vectors. The data is modeled by keeping the eigenvectors corresponding to the principal components that explain a certain percentage of the data variance. This percentage is a parameter that must be chosen with again, no principled way to choose this parameter. Once the principled components are chosen, an anomaly is declared if its reconstruction error defined as the squared error of the difference between the anomaly value and its reconstructed value is below, again, a predetermined threshold. The reconstructed value is determined by projecting the value onto the chosen principal components. Hence, using anomaly detection for PCA requires the selection of two parameters, the percentage of variance which determines the number of principal components kept in the model, and the reconstruction error threshold which determines the size of the error required to declare the anomaly. This approach has been used for anomalous intrusion detection [27–30]. Another reconstruction or cluster approach is based on spectral clustering. Spectral clustering is similar to PCA as it is based on an eigenvector analysis. However, whereas PCA is based on an eigenvector analysis of the feature covariance matrix, spectral clustering is based on an eigenvector analysis of the graph Laplacian of a feature similarity matrix. This approach is used to cluster the data into separate clusters. Spectral clustering has been used for anomaly detection [31] for detection of computer intrusions. In this effort, anomalies are declared if events belong to small clusters. Spectral clustering has been used as an integral element in several anomaly detection efforts [31–33]. In these efforts, several different strategies are used to declare anomalies.

Boundary methods attempt to compute a boundary around the normal class such that the data outside the boundary are anomalies. These methods require less data than the density methods as only the boundary is being estimated. One example boundary technique is the $k$-centers technique which uses $k$ hyperspheres to cover every data point. The number of centers, the position of the centers, and the radius of the spheres are all parameters that must be chosen and estimated to form the boundary [34]. These parameters are adaptively chosen to include a fixed percentage of the training points. The optimization process to choose these parameters is very computationally complex. Further, the volume enclosed by the boundary can be larger than with other boundary approaches due to the fixed shape
of the hyperspheres. Another boundary technique is the nearest neighbor distances (NN-d) method [22]. It computes the distance of the test track \( b \) to its nearest neighbor in the training set. It then computes the distance of the tracks’ nearest neighbor to the tracks nearest neighbors nearest neighbor also in the training set. A threshold on the ratio of these two distances is used to detect anomalies. There is, again, no principled way to select the threshold nor is there a principled way to say anything about the nature of the boundary. The final boundary method that is reviewed is the one class support vector machine (OC-SVM). This method is the classification method that will be used in this thesis, and hence, will be reviewed in detail in Chapter 5. The property that makes it most attractive for the anomaly detection application is the control and clear interpretation of the classification boundary. The OC-SVM used for this effort encloses all the training data and has the most outlying training tracks as support vectors on the boundary. These properties are used effectively to help operator interaction with the classifier to improve performance and to tailor the classifier to the specific needs of the operator.

2.4 Video Anomaly Detection

This section reviews several video anomaly detection papers. These papers are compared based on the categorizations developed earlier in this chapter. These techniques extract features directly from the video instead of first performing tracks. Hence, the details of the feature extraction will not be of specific interest; however, the type of anomaly detection, type of features, method of feature selection, and type of classification will be considered and contrasted. In [35], the average flow of patches of video are statistically characterized and clustered to detect anomalous collective behavior. In [36], a similar feature extraction method was used; however, the patches were clustered using spectral clustering and the clusters’ histogram was used to determine anomalies. In [37], patches were statistically characterized and compared over time using the Kullback-Liebler (KL) divergence. If the KL divergence changed too dramatically according to a self-organizing map classifier, an anomaly was declared. In [25], the optical flow along trajectories is modeled using chaotic dynamics. These trajectories are clustered using the Jenson-Shannon divergence and then are modeled using a Gaussian mixture model (GMM). An anomaly is declared at a certain error rate based on the GMM. In [38], the video is divided up into cells and space-time histogram features are computed in each cell. Anomalies are declared when the new histogram features are sufficiently statistically distant from the trained histogram features. The features are continually updated and the process is unsupervised. In [26], the correlation among space-time optical
flow are used as features which in turn are represented as kernel density estimators. An anomaly is declared based on its probability given the estimated density. In [39], optical flow is calculated and then clustered. The orientation of the flow is quantized and the flow is fitted to the social force model. An anomaly is declared when an abrupt change happens in the combined orientation and social force model. In [40], flow, size, and texture features are computed in non-overlapping cells and kernel smoothed histograms are used to represent each feature. Low confidence events in any of the feature distributions triggers an anomaly detection.

The video anomaly detection techniques reviewed above are summarized in Table 2.1. The databases used in Table 2.1 are the University of Minnesota UMN, the University of California at San Diego UCSD, Performance Evaluation of Tracking and Surveillance PETS, and the title video is data made for that particular study. The UCSD data set is a single camera view of a walkway on a campus setting that captures mainly pedestrians. The UCSD data set was hand tracked and used in this study. The UMN data set is a single camera view of a scene that has a group of people that walk around and then scatter out of the frame at the same time. The PETS data set is a single camera view of a sidewalk near a building that captures pedestrians and groups of people walking along the path. Note that in addition to features being extracted directly from the video instead of extracting features from tracks, none of the techniques used boundary methods to do the classification. This thesis extracts the features from tracks instead of the video directly, and the classification technique used in this thesis is a boundary technique. These approaches to anomaly detection from video used in this thesis are different than the approaches reviewed in this section. The specific features and classification technique used in this thesis are described in detail in later chapters.

2.5 Summary

Anomaly detection systems can be characterized by the types of anomalies that they detect: point, contextual, or collective. They can also be characterized by whether they are supervised, semi-supervised, or unsupervised. Feature extraction techniques can be either physical or abstract, and feature selection techniques either select subsets of features, cluster features, or combine features in various ways. There are many different classification approaches, but three categories are particularly appropriate for anomaly detection. These categories are for one class classifiers: distribution techniques, reconstruction techniques, and boundary techniques. Several efforts performing anomaly detection with video were reviewed and characterized based on the type of anomaly detection, the type of feature extraction, and
the type of classification. This background provides the context for the physically based features and the one class classifier pursued in this thesis.
Chapter 3

Approach

The approach taken for this effort is to develop a system that can be adapted easily by the operator. This adaptability is important because the definition of an anomaly or threat changes as a function of threat level. Human operator based adaptation allows the operator knowledge of what is a threat under a set of particular conditions to be used to change the behavior of the anomaly detection system. In this effort, track behavior is broken out into three categories: normal behavior, anomalous behavior, and threats. Normal behavior is based on the available training data for a particular scenario of interest. For this effort, it is assumed that examples of anomalous data are not available as only normal data is available. Hence, anomalies are defined as being not normal or outside the decision boundary of the normal tracks. Threats, on the other hand, are specifically defined by the operator based on the operators knowledge of the scenario.

The operator may modify the anomaly detector during the training process, or may modify the anomaly detector during operation by examining the tracks classified as anomalies and determining whether the system is operating consistent with the operators expectation. If the behavior of the anomaly detector is deemed to be inconsistent, then the operator can take corrective action.

A block diagram of the system is shown in Figure 3.1. Although in principle, any set of features and any classifier could be used as part of the system concept, the features selected and the classifier selected were chosen to mesh with the approach of this thesis. The physically based features and the one class SVM (OC-SVM) make it easier for the operator to adapt the performance of the anomaly detection system. In addition to the usual block diagram where the feature extractor feeds into a classifier, this system explicitly displays the results of the classifier in a manner that allows the operator to understand the classification and in a manner to help the operator modify the anomaly detector. This modification includes both selecting the data that is used to train the classifier and adding features feeding into the classifier. Note that with the anomaly detection approach, there is no need for the operator to directly interact with the classifier as the classifier automatically tunes
itself and establishes the decision boundary solely based on the training data and features.

For example, the operator feedback is depicted as part of the training stage in Figure 3.2. The OC-SVM classifier is designed so that the most outlying track examples always lie on the decision boundary. In this way, the operator can be shown track examples depicted on a video frame of tracks overlayed on a video frame as shown in Figure 3.3 and can determine if she/he considers those tracks anomalous. If those tracks are considered anomalous then the tracks are removed from the training set and a new boundary is found. This boundary by definition will not contain the offending track, and should be a boundary that more tightly bounds the normal data. This process can be iterated until the operator is satisfied with the anomaly detector performance. Note that for this process, the operator only has to decide whether or not to remove tracks from the training data. The rest of the process is automated including displaying the tracks associated with support vectors and automatically tuning any parameters associated with the construction of the decision boundary. The data used in Figure 3.3 is hand tracked from the UCSD data set.

The operator can also modify the classifier during operation, or for example, at the end of day during a testing phase of the system. As depicted in the block diagram in Figure 3.4, the operator can visualize the tracks detected as anomalies overlaid on an image of a scene as shown in the image in Figure 3.5. If any of these tracks are considered normal, they can be added to the training data and the classifier can be retrained, again, without further operator interaction. This loop can be repeated until the operator is satisfied with the results. In addition, the operator can visualize all the classifications performed during the day to see if any of the normal tracks should be anomalies or threats. In this case, changing the boundary is not straightforward and is likely not the reason for the misclassification. The likely cause is that the set of features used by the classifier are not sensitive to the
CHAPTER 3. APPROACH

Figure 3.2: Operator Interaction During Training

Figure 3.3: Training Data (left) with Support Vectors in Red; Labeled Support Vectors Only (right)
particular property of the track that causes it to be a threat or an anomaly. Hence, the operator would need to add another feature to the classifier. In this case, the importance of the physically based features is apparent as the operator can directly design the feature needed to detect the particular threat or anomaly. This process can also be iterated as depicted in Figure 3.4.

In the next two chapters, the feature structure and features will be detailed and the classifier will be thoroughly explained. These explanations should shed light on the usefulness of these features and classifier for anomaly detection and, in particular, should explain the usefulness of the features and classifier for an anomaly detector that accepts feedback from the operator and adapts it performance based on that operator feedback.
Figure 3.5: Testing Data (left) with Anomalies in Red; Labeled Anomalies Only (right)
Chapter 4

Feature Extraction

The features defined for this thesis are based on physically derivable measurements from tracks. A track is not very complicated. It is a space / time trajectory. For this effort the space is 2D. A track can be represented either as a sequence of triples \((x, y, t)\); or if the sampling time for the tracks is constant, the track can be simply represented as a sequence of \((x, y)\) locations. Hence, the features can be spatial features, temporal features, and features derived from coupled space / time such as velocity. In addition, the features can be computed on a single track or can be computed comparatively. The comparative features are important as a track can be compared to existing tracks in the training set known to be normal. Further, many barriers can be represented as linear features such as fences or as the outlines of ”keep out” areas and, hence, comparative features can be used to compare tracks and their relationship with barriers or other linear features that relate to threat-like behavior.

Given the limited degrees of freedom of tracks, the features can be computed in a structured way that facilitates the addition of new features if required either based on human derived features as done in this thesis or potentially by automated means which could be performed in future work. Further, given the emphasis on human derived features, the structured features favor features that have physical meanings that favor human comprehension and interpretation of the features.

4.1 Feature Structure

The feature structure is examined both for single tracks and for comparing tracks. Although the structures are similar, there are important distinctions between the two types of features. Both are general and capable of different types of discrimination between normal and anomalous tracks.
4.1.1 Single Track Features

The single track feature structure is illustrated in Figure 4.1. In general, each step of the structure is performed; however, for some features, one or more steps can be skipped as long as a track is the input and a scalar is the output of the feature extraction process. To illustrate this process, a feature is depicted which computes whether or not a track is loitering in a particular area in the image. In the first stage, the track is spatially filtered to lie in the area of interest. In the second stage the speed is calculated at each time step resulting in a speed vector. In the third step, the speed is filtered with a moving median window filter with the size of the time window equal to the length of time considered by the operator to be loitering. In the fourth and final step, the minimum operator is applied to the filtered input speed vector providing, in this example, a low value of the scalar $S_L$. This example is illustrated in Figure 4.2.

4.1.2 Comparative Track Features

The comparative track feature structure is illustrated in Figure 4.3 along with a depiction of a specific feature example. In this case, the modified Hausdorff [42, 43] feature is shown. The modified Hausdorff feature is a distance measure between two tracks. In the first stage, for each member of track A, the distance to the closest member of track B is depicted. In addition in the first stage (Distance), the distance for each member of track B to the closest member of track A is depicted. The input to this stage are two tracks, and the output of this stage are two vectors, one providing the distance from A to B and the second giving the distance from B
CHAPTER 4. FEATURE EXTRACTION

Figure 4.3: Comparative Track Feature Structure Example - Modified Hausdorff Feature

In the second stage (Reduce), the two vectors are the input, and the output are two scalars. In the modified Hausdorff case, the output scalars are the largest entries of each vector representing the largest distance from A to B and B to A. Finally, in the third and final stage (Compare), the minimum of the two scalars are chosen to represent the minimum of the maximum distances between A and B. For the Hausdorff distance, this would normally be the maximum distance; however, in this case a "modified" Hausdorff distance is computed. This distance is considered more appropriate to compare tracks as the shorter tracks can be compared to a longer track such that the shorter track can be a subset of the longer track and not be penalized. Thus the tracks could travel along the same path but not the same distance along the path.

4.2 Features

In this section, the features are defined mathematically. First the features calculated from a single track will be defined. These include speed, heading, and vorticity. Speed and heading are simply the two components of the track velocity at any point in the track trajectory. The vorticity is the local deviation from a straight line path. These are the fundamental features that can be calculated from a track trajectory. They are used primarily in the first Reduce stage in the Single Track Feature Structure as depicted in 4.1 where a track is the input and a vector is the output of this
4.2.1 Single Features

Single Features (input: tracks, output: vector)

The equations [44] of the three fundamental single track features are given in equations (4.1), (4.2), and (4.3).

- **Speed**
  \[
  s_t = \frac{\|\mathbf{p}_t - \mathbf{p}_{t-w}\|_2}{w}
  \]  
  where 
  \(s_t\) = the speed at time \(t\). 
  \(\mathbf{p}_t\) = Row vector \((x, y)\) where \(x\) and \(y\) are real numbers representing 2 dimensional position. 
  \(t\) = an integer index for time. 
  \(w\) = Number of data points in the moving window.

- **Heading**
  \[
  h_t = \frac{\mathbf{p}_t - \mathbf{p}_{t-w}}{\|\mathbf{p}_t - \mathbf{p}_{t-w}\|_2}
  \]  
  \[
  a_t = \arctan\left(\frac{h_t^{(y)}}{h_t^{(x)}}\right)
  \]  
  where 
  \(h_t\) = the normalized direction vector at time \(t\). 
  \(a_t\) = the direction angle (in radians) at time \(t\).

- **Vorticity**
  \[
  X_C = X - \text{mean}(X)
  \]  
  \[
  [U \Sigma] = \text{eig}(X_C X_C^T / (w - 1))
  \]  
  \[
  \hat{X} = X_C U_R U_R^T + \text{mean}(X)
  \]  
  \[
  V = \|X - \hat{X}\|
  \]  
  \[ \text{(4.3)} \]
where

\[
X = \begin{bmatrix}
p_1 \\
\vdots \\
p_t \\
p_{t+1} \\
\vdots \\
p_w
\end{bmatrix}
\]

\[X_C = \text{Centered Data.}\]
\[\hat{X} = \text{Fitted Line.}\]
\[w = \text{Number of data points in the moving window.}\]
\[\Sigma = \text{Eigenvalue matrix.}\]
\[U = \text{Eigenvector matrix.}\]
\[U_R = \text{Reduced eigenvector matrix.}\]
\[V = \text{Vorticity or line fit error.}\]

**Single Feature (input: vector, output: scalar)**

The second reduce stage in Figure 4.1 uses various mathematical operators including \text{max}, \text{min}, \text{median}, and \text{mean}. These operators either 1) tailor the feature to a specific type of anomaly or threat, or 2) summarize a feature so that it can be put into a feature vector for use in the OC-SVM classifier.

### 4.2.2 Comparison Features

In this section the mathematical description of the comparison features are described. These features have a general structure which is described in equation (4.4). This mathematical structure is quite flexible and useful for comparing two point sets such as tracks. The modified Hausdorff metric illustrated in Figure 4.3 uses the general structure and is mathematically described in equation (4.5). Another useful comparative feature useful in determining the point of closest approach to a barrier or fence is described mathematically in equation (4.6). Both of these features are used in the classifier experiments in Chapter 6.
CHAPTER 4. FEATURE EXTRACTION

- Mathematical comparative feature structure.

\[
d(X, Y) = \max_{p_t^{(x)} \in X} \min_{p_t^{(y)} \in Y} \| p_t^{(x)} - p_t^{(y)} \|
\]

and

\[
d(Y, X) = \max_{p_t^{(y)} \in Y} \min_{p_t^{(x)} \in X} \| p_t^{(y)} - p_t^{(x)} \|
\]

and

\[
D(X, Y) = \min(d(X, Y), d(Y, X))
\]

- Hausdorff distance between two tracks or tracks and linear features

\[
d(X, Y) = \max_{p_t^{(x)} \in X} \min_{p_t^{(y)} \in Y} \| p_t^{(x)} - p_t^{(y)} \|
\]

and

\[
d(Y, X) = \max_{p_t^{(y)} \in Y} \min_{p_t^{(x)} \in X} \| p_t^{(y)} - p_t^{(x)} \|
\]

and

\[
D(X, Y) = \min(d(X, Y), d(Y, X))
\]

- Point of closest approach.

\[
d(X, Y) = \min_{p_t^{(x)} \in X} \min_{p_t^{(y)} \in Y} \| p_t^{(x)} - p_t^{(y)} \|
\]

and

\[
d(Y, X) = \min_{p_t^{(y)} \in Y} \min_{p_t^{(x)} \in X} \| p_t^{(y)} - p_t^{(x)} \|
\]

and

\[
D(X, Y) = \min(d(X, Y), d(Y, X))
\]

4.3 Summary

The features used in this effort are based on physical properties of tracks and allow features to be tailored to specific spatial and temporal conditions for a given
scenario. These features also have a specific structure as described in this chapter. Both of these properties enable the understanding of the usefulness of these features and the ability to construct new features.
Chapter 5

Classifier

This section describes the properties of the chosen classifier the OC-SVM, and highlights those properties that make it easy for an operator to understand and modify its performance. The OC-SVM will be derived and the relationship of the OC-SVM parameters to its performance will be described.

5.1 SVM

The SVM comes in many different flavors. It can be the primal or dual formulation. It can be linear or can use a kernel function. It can be a one-class or a two class classifier. For this thesis, the SVM used is the dual formulation that uses a radial basis kernel function and is a one-class SVM. The mathematical development of this one-class SVM begins with the dual formulation of the two class SVM. The one-class SVM is then developed from the two class SVM by using the origin as one of the classes. Finally, a radial basis kernel is used to complete the development.

One of the powerful techniques that can be applied to the dual formulation of the SVM is called the kernel trick. The kernel trick allows the substitution of a non-linear function for the outer product in the SVM formulation. This substitution lets the boundary between the classes to be effectively non-linear. This resulting non-linear boundary is illustrated in Figure 5.1 by using a radial basis function as the substitution of the outer product. Note the non-linear decision boundary effectively carves the vectors away from the rest of the two dimensional space resulting in a one class classifier.

5.1.1 SVM Formulation

To overview the one class SVM formulation, one begins with the SVM 2 class dual formulation. This formulation can be interpreted as separating two distinct classes each bounded by a convex hull as depicted in Figure 5.2. The SVM finds the hyperplane that is the bisector of a line connecting the closest points between the
two convex hulls. The one-class problem follows straightforwardly from the 2-class problem by substituting the origin for one of the classes as shown in Figure 5.3. Finally, the "kernel trick" is used to obtain the decision boundary shown in Figure 5.4.

In more detail, the derivation of the one class classifier in dual form begins with the two class classifier whose solution is shown in equation (5.2). \cite{45,46} In this equation, A and B are matrices containing the two sets of n element vectors given $m_A$ as the number of vectors in A and $m_B$ as the number of vectors in B. Thus, A is $(m_A \times n)$ and B is $(m_B \times n)$. $u$ is the unknown column vector – of length $m_A$. Similarly, $v$ is the unknown column vector of length $m_B$. The non-zero values of $u$ and $v$ resulting from the solving the optimization in (5.2) will be the indices of the support vectors. Note that the solution is a constrained optimization problem which is convex. In fact, the optimization is formulated by first determining the closest distance between the two sets of points. This distance is the minimum distance between the convex hulls enclosing the points. The linear decision region is given as the orthogonal bisector of the line connecting the two closest points. This geometry is shown in Figure 5.2 as a result of solving equation (5.2) for the points plotted.
CHAPTER 5. CLASSIFIER

Figure 5.2: Two Class SVM Example

\[ u^T A = u_1 A_1 + u_2 A_2 + u_3 A_3 + \cdots + u_m A_m \]

where \( u \in \mathcal{R}^{mA}, \sum_i u_i = 1, u \geq 0 \) \hspace{1cm} (5.1)

\[ \min_{u,v} \frac{1}{2} \|u^T A - v^T B\|^2 \]

s. t. \( \sum_i u_i = 1, \sum_i v_i = 1, u \geq 0, v \geq 0 \) \hspace{1cm} (5.2)

The one-class SVM follows trivially from the two class SVM. [47, 48] One can simply set \( B = 0 \) in equation (5.2) to obtain

\[ \min_u \frac{1}{2} \|u^T A\|^2 \]

s. t. \( \sum_i u_i = 1, u \geq 0 \) \hspace{1cm} (5.3)

This optimization corresponds to finding a separating line or, in general, a hyperplane that separates the vectors in \( A \) from the origin as depicted in Figure 5.3. This depiction is generated using equation (5.3). Note that the boundary is very far
away from the data vectors and hence, this linear one class classifier is not likely to give satisfactory results.

The third step in the derivation of the one class SVM is to use the "kernel trick". Here we note that the term being minimized $\frac{1}{2}||A'u||^2$ can be written as $\frac{1}{2}u^TAA^Tu$. The "kernel trick" [49] uses a kernel matrix $K$ to substitute for $AA^T$. This substitution provides the ability to form non-linear decision boundaries in the data space. These non-linear decision boundaries correspond to linear decision boundaries in a transformed space that results from $K$ operating on the data vectors. The kernel matrix used in one class SVM’s is the radial basis function shown in equation (5.4) where $\sigma$ is called the bandwidth or size of the Gaussian kernel.

$$K(x, y) = \frac{e^{-||x-y||^2}}{2\sigma^2}$$  \hspace{1cm} (5.4)

Using the radial basis function, and substituting into equation (5.3), one obtains the following optimization:

$$\min_{u} \frac{1}{2}u^TKu$$

s. t. $\sum_i u_i = 1, \ u \geq 0$  \hspace{1cm} (5.5)
Using equation (5.5) and applying it to the data in Figure 5.3, one obtains the result shown in Figure 5.4. Note in this case, the boundary "hugs" the data vectors; hence, the introduction of the Gaussian kernel makes a significant difference in the discrimination potential of the decision boundary.

Once having trained the classifier, the classification of the anomalies is performed using equation (5.6) and an anomaly is declared when the expression in equation (5.6) is true.

\[ K(X_{\text{support}}, X_{\text{test}})^T d - \text{mean}(K(X_{\text{train}}, X_{\text{train}})^T d) < 0 \]  

(5.6)

where \( K \) is the kernel, \( X_{\text{train}} \) is the training data, \( X_{\text{support}} \) contains the support vectors, \( X_{\text{test}} \) is the test data, and \( d \) is the vector containing the non-zero values of \( u \). \( u \) is the vector resulting from solving equation (5.5). The training vectors corresponding to the non-zero values of \( u \) or the values of \( d \) are the support vectors.
5.1.2 Simple SVM Example

A simple 5 point example is used to illustrate the training and testing of the one class SVM. The training points are given below and are plotted on the left side of Figure 5.5.

\[
X_{\text{train}} = \begin{pmatrix}
0.9479 & 0.6210 \\
0.0821 & 0.5732 \\
0.1057 & 0.0521 \\
0.1420 & 0.9312 \\
0.1665 & 0.7287
\end{pmatrix}
\]

Equation 5.4 is used to calculate the kernel, \( K(X_{\text{train}}, X_{\text{train}}) \) using \( \sigma = .5 \).
The kernel, \( K \), is used to calculate \( u \) in Equation 5.5. 

\[
K(X_{train}, X_{train}) = \begin{pmatrix}
1 & .2223 & .1267 & .2251 & .2881 \\
.2223 & 1 & .5797 & .7689 & .9396 \\
.1267 & .5797 & 1 & .2126 & .3974 \\
.2251 & .7689 & .2126 & 1 & .9201 \\
.2881 & .9396 & .3974 & .9201 & 1
\end{pmatrix}
\]

The kernel, \( K \), is used to calculate \( u \) in Equation 5.5.

\[
u = \begin{pmatrix}
.3448 \\
0 \\
.3491 \\
.3061 \\
0
\end{pmatrix}
\]

Hence, \( d \) equals

\[
d = \begin{pmatrix}
.3448 \\
.3491 \\
.3061
\end{pmatrix}
\]

The support vector values are the training points corresponding to the non-zero values of \( u \) or the values of \( d \) and are given below:

\[
X_{support} = \begin{pmatrix}
.9479 & .6210 \\
.1057 & .0521 \\
.1420 & .9312
\end{pmatrix}
\]

Next, test values are selected. One is selected as a normal example and the other is chosen as an anomaly.

\[
X_{test} = \begin{pmatrix}
.75 & .75 \\
1 & 1
\end{pmatrix}
\]

Using these \( X_{test} \) values, a new kernel \( K(X_{support}, X_{test}) \) is calculated using Equation 5.4.

\[
K(X_{support}, X_{test}) = \begin{pmatrix}
.8943 & .1646 & .4471 \\
.7462 & .0335 & .2273
\end{pmatrix}
\]
Equation 5.6 can now be used to determine whether each point is a normal point or an anomaly. If the result of Equation 5.6 is true or \(< 0\), than the point is an anomaly. In contrast, if the result is false or \(> 0\), then the point is normal. For the point \(x_{\text{test}} = (0.75, 0.75)\), the result is 0.0448. Point \(x_{\text{test}} = (0.75, 0.75)\) is a normal point. For the point \(x_{\text{test}} = (1, 1)\), the result is −0.1194. Point \(x_{\text{test}} = (1, 1)\) is an anomaly. All of the training points and test points are shown on the right side of Figure 5.5. The training points are the blue dots with the support vectors circled. The normal test point is a green plus sign, and the anomaly test point is a red square.

### 5.1.3 SVM parameters

For this one class formulation, the support vectors all lie on the decision surface boundary and the remainder of the training feature vectors lie inside the decision boundary. The shape and smoothness of the boundary and the number of support vectors are driven by only one parameter \(\sigma\) in the radial basis function kernel as defined above. The effect of \(\sigma\) is illustrated in Figure 5.1.3. Note that as \(\sigma\) decreases the boundary becomes less smooth and the more support vectors are needed to support the more complex boundary. The fact that the number of support vectors grows with boundary complexity is one of the properties used in automatically tuning the \(\sigma\) parameter in the next section.

![Figure 5.6: Sensitivity of SVM Decision Boundary to \(\sigma\)](image)

These support vectors represent potential anomalies in the training set and are the vectors shown to the operator to make sure that these vectors are indeed normal and not anomalies. If the operator decides that any one of the vectors is an anomaly, then the vector is eliminated from the training set and the support vector classifier is retrained. Hence, the only parameter that needs to be determined is \(\sigma\) which is found by using a novel L-curve procedure.
5.1.4 Tuning SVM parameters: Relationship with decision boundary and performance

A standard way of tuning classifiers is to use the leave one out method or the n-fold cross validation scheme. The leave one out method trains the classifier with all training samples but one, and then determines if the sample that was left out is correctly classified. This process is repeated for all training samples and for all potential values of $\sigma$. The value of $\sigma$ is chosen that gives the highest classification performance. Clearly, this process is computationally complex as the classifier must be trained $N$ times $M$ where $N$ is the number of training samples and $M$ is the number of $\sigma$ values under consideration. One method to reduce the complexity of this process is to perform n-fold cross validation which approximates the leave one out method by choosing $n$ random train/test partitions of the data and determining the classification performance by averaging over the scores of the $n$ partitions. This procedure reduces the complexity to $n$ times $M$ where $n << M$. With this methodology, if $\sigma$ is too small, the boundary becomes too complex and the classifier becomes over trained: it essentially memorizes the training data and does not generalize. Thus, errors become larger with either the leave one out or n-fold cross validation procedure. This type of error also applies to the one class SVM.

On the other hand, when $\sigma$ becomes too large, the boundary grows and becomes too smooth and the decision boundary is not sufficiently complex or discriminative. Hence, the classifier over generalizes. This type of error is not detectable for a one class classifier as there is no other class for the classifier to mis-classify when the boundary becomes smoother and smoother. If classification performance is plotted as a function of $\sigma$, the curve takes the shape of an L. The performance gradually improves as $\sigma$ grows from small to large and then the performance flattens out or asymptotes as the decision boundary becomes sufficiently smooth to encompass all of the training data.

To illustrate this performance behavior, consider the two dimensional feature vectors extracted from a subset of the UCSD data. These data are plotted in Figures 5.7 and 5.8 along with the decision boundary contours resulting from changing $\sigma$. As can be seen, a small $\sigma$ makes the boundary complex requiring a larger number of support vectors. Further, this complex boundary does not appear to be a reasonable decision boundary. As $\sigma$ becomes larger, eventually the boundary and the number of support vectors changes little if at all. This behavior is illustrated in Figure 5.9 where the percent error is plotted using 10-fold cross validation as a function of $\sigma$ on the left and the percent support vectors of the total number of vectors is plotted as a function of $\sigma$ on the right. A similar L shape is observed for both these plots. This similar behavior is due to the relationship of the number of
support vectors with the boundary complexity, and the relationship of the boundary complexity to classifier performance. A overly complex boundary will overtrain the classifier and hurt classifier performance as observed on the plot on the left. Further, the number of support vectors are directly related to the boundary complexity as observed in Figures 5.7 and 5.8. These plots motivate the use of the number of support vectors to determine $\sigma$ due to their direct relationship to cross-validation results – an accepted method for determining $\sigma$ for two class SVM’s. By using the number of support vectors to find $\sigma$ compared to n-fold cross validation, there is a savings in computation from $n \times M$ down to $M$.

Now that we have a methodology for efficiently generating the L-curve, it remains to develop a strategy of using the L-curve to find $\sigma$. Fortunately, the use of the L-curve to do parameter selection is a common problem in many optimization settings including SVM. Unfortunately, there is no theoretical or even empirically agreed upon way to use the L-curve to find the parameter of interest. For this thesis, a
Figure 5.8: Decision Surface vs. $\sigma$

Figure 5.9: Comparison of 10 fold Cross Validation vs. Support Vector Count
new method is used to choose \( \sigma \). It borrows two ideas from other L-curve selection methods. First, it uses the fact that the L-curve is often plotted on a log-log scale as a first step in the parameter selection process. The log scale on the y-axis used to plot the number of support vectors makes sense as the support vectors grow exponentially as the \( \sigma \) parameter gets very small. On the other hand, based on the fact that the data is centered and normalized, the values of \( \sigma \) vary over a fixed and linear range; hence, the x-axis is plotted as linear. Secondly, the point on the L-curve that is chosen to select \( \sigma \) is the point of highest curvature as this represents the point on the curve that gives the best tradeoff between boundary complexity and smoothness. Again, unfortunately, finding the point of highest curvature in an often very noisy L-curve is a difficult problem with no agreed upon solution. For this thesis, a novel method for determining an approximation to this curvature point with a noisy L-curve is developed and is illustrated in Figures 5.10, 5.11, 5.12, and 5.13. Note that in each case, the automatic \( \sigma \) selection procedure gives reasonable decision surfaces. It would be more satisfying to be more quantitative about the effectiveness of the decision surfaces, but for this thesis, the assumption is that anomalies are not available for training. Without anomaly data to score, the training process cannot use classification performance as an effectiveness measure as is done in standard two-class SVM’s. The heuristic process for finding \( \sigma \) is as follows:

1. Center and normalize the data robustly using median and median of absolute deviations from the data’s median (MAD). For a single vector, \( x \), we have

   \[
   \text{MAD} = \text{median}_i (|x_i - \text{median}_j (x_j)|) \tag{5.7}
   \]

   Hence the normalization is

   \[
   x_i^{\text{norm}} = (x_i - \text{median}_j (x_j)) / \text{MAD} \tag{5.8}
   \]

2. Next train the one class SVM for each \( \sigma_i = \{.3,.325,.35,.375, \cdots , 6\} \)

3. For every pair of \( (\sigma_i, \sigma_j) \) with \( i < j \), fit every combination of three lines with the endpoints: \( (\sigma = .3, \sigma = \sigma_i), (\sigma = \sigma_i, \sigma = \sigma_j) \), and \( (\sigma = \sigma_j, \sigma = 6) \). The fit uses the first principle component of the SVD to compute the best fit line that minimizes the orthogonal distance. This fit uses the same procedure as the calculation of the vorticity in Equation 4.3.

4. The next step is heuristically determined by observing a correlation between the angle of the lines comprising the fit, the perceptible point of maximum
curvature, and the resulting boundary smoothness based on experiments conducted in 2D. The heuristic is as follows:

(a) If the absolute value of the angle of the \((\sigma = .3, \sigma = \sigma_i)\) line is greater than 30 degrees, then set \(\sigma = \sigma_i + k_1\).

(b) Else if the absolute value of the difference between the angle of the \((\sigma = .3, \sigma = \sigma_i)\) line and the angle of the \((\sigma = \sigma_i, \sigma = \sigma_j)\) line is greater than the absolute value of the angle of the \((\sigma = \sigma_i, \sigma = \sigma_j)\) line, then set \(\sigma = \sigma_j + k_2\).

(c) Else set \(\sigma = \sigma_j\)

(d) For all experiments in this thesis, \(k_1 = .2\) and \(k_2 = 1\) as these \(k\) positions represented the perceptible offset of the curvature point to the endpoints of the best fit lines.

5.2 Conclusion

The one-class SVM was chosen as the classifier based on several factors explained earlier. These factors allowed a clear physical interpretation of the SVM performance which made the SVM a practical choice for this application. In concert with the
Figure 5.11: Illustration of Automated \( \sigma \) Selection using Line Segment Fitted L-curve

Figure 5.12: Illustration of Automated \( \sigma \) Selection using Line Segment Fitted L-curve
physical interpretation of SVM performance, a novel automated way of selecting the single tuning parameter for this one class SVM classifier makes the classifier easily tunable and usable. This tunability is important as the application must adapt to different threat levels which practically means that accepted behavior for one situation is anomalous behavior in another situation. Hence, the operator can systematically select behavior or tracks that are representative of the current threat condition. With this training data, the system automatically tunes itself to the new conditions. In addition, since the support vectors are on the decision boundary, the support vectors represent extreme cases whose tracks can be visualized by the operator to ensure satisfaction with the classifier sensitivity. The stable and transparent performance of the one class SVM make it a good choice for this application.
Chapter 6

Experiments

6.1 Data

The experiments were performed with both measured and synthetic data. The measured data is a standard data set for detecting anomalies. The setting is along a pathway so it would constitute a portion of the installation scenario. The other set of data is simulated and is developed using Google Maps images so that the paths of the pedestrians entering and exiting the installations were realistic and in context.

6.1.1 Measured Data: University of California at San Diego (UCSD) data base

The tracks were obtained from the UCSD Database [50] by hand tracking the pedestrians in the video data. This effort starts with tracks instead of video inputs. The UCSD data has a single vantage point camera pointed at a pedestrian walkway that is a campus setting. Pedestrians and occasionally bikes, etc. traversed the path. Both training and testing conditions were provided. In particular, frame sequences with anomalies were specifically labeled. The labels were at the frame sequence level instead of the individual track level; however, by watching the video, it is easy to pick out the anomalous tracks. An example frame is shown in Figure 6.1

To create the tracks, the x and y coordinates of the pedestrians were tracked individually for each track for each frame. The x and y coordinates were tracked using ImageJ, a publicly available image analysis program. The tracks were then read into MATLAB, translated, and analyzed using MATLAB. The handtracking was labor intensive so the entire data base was not used in this effort.

6.1.2 Simulated data

The simulated tracks were developed by writing a track simulator to automatically generate tracks within a velocity and spatial trajectory range given baseline trajectory profiles and baseline velocity profiles that were given as a function of position along a path. These profiles were based on trajectories overlaid on Google
Map images of the real installation including an actual entranceway used to enter the installation. The installation used was the military installation Wright Patterson Air Force Base in Dayton, Ohio. The trajectories were generated using a spline fitting program with knots that were selected to provide that nominal pathway to the installation entrance and exit. After choosing the anchor points, the baseline spline was fitted and the samples along the spline were determined by making a velocity profile as a function of position along the pathway. After the baseline trajectory and baseline velocity profiles were generated (several), noise is added to the initial track at several spatial scales to generate the tracks used to train and test the classifier. The training and test sets were generated separately to make sure that there is sufficient separation between the training and test sets. The normal tracks (in blue), the anomalies (in red), and fences (in green) are shown in Figure 6.2.
6.2 Results

Results will be given with both measured and simulated data. In particular, likely interactions with a hypothesized customer will be staged to demonstrate the flexibility of the approach to include the change in performance as the customer changes the context of the situation. Results will be given as correct and false detections of anomalies along with visual examples of both type of errors.

6.2.1 Measured Data Results

Recall that the measured data is taken from a campus walkway and consists of both training data and test data. The training data consists of only normal tracks
while the test data consists of both normal and anomalous tracks. The one class SVM is scored with the following three metrics:

\[
TPR = \frac{\sum_i TP_i}{P}
\]

\[
FPR = \frac{\sum_i FP_i}{N}
\]

\[
ACC = \frac{\sum_i TP_i + \sum_i TN_i}{P + N}
\]

TPR stands for the True Positive Rate with \(TP_i\) equal to a true positive (correctly classified anomaly) and \(P\) equal to the number of anomalies in the test set. FPR stands for False Positive Rate with \(FP_i\) equal to a False Positive (calling a normal track an anomaly) and \(N\) equals the number of normal tracks in the test set. ACC stands for accuracy with \(TN_i\) equal to a true negative (correctly classified normal track).

Several combinations of features were tested; however, the two feature classifier using the Loiter Feature and the Centered Modified Hausdorff Distance performed as well as any other combination tested and hence is examined in detail. The only input to the one class SVM algorithm to train was the data itself. For all cases, the automated \(\sigma\) selection algorithm selected the \(\sigma\) used to train the classifier. The two feature classifier is shown in Figures 6.3, 6.4, 6.5, and 6.6. Figure 6.3 shows the L-curve fit that the \(\sigma\) selection algorithm calculated. Recall again that this \(\sigma\) selection is performed automatically by plotting the log of the number of support vectors vs. various \(\sigma\) values and fitting line segments to determine a "knee" in the plotted curve. Figure 6.4 shows the SVM decision surface with both the support vectors and the classified anomalies encircled in red and annotated with a unique number identifying the particular track. The left panel shows only the training data and the support vectors whereas the right panel shows both the training and test data. In Figure 6.5, the left panel shows all the training tracks with the tracks corresponding to the support vectors shown in red. The right panel shows only the tracks of the support vectors all in different colors with the legend showing the unique numbers corresponding to the support vectors shown in Figure 6.4. In Figure 6.6, the left panel shows all the test tracks with the tracks corresponding to the classified anomalies shown in red. The right panel shows only the classified anomalous tracks all in different colors with the legend showing the unique numbers corresponding to the vectors shown in Figure 6.4. The performance for this classifier was \(TPR = .85\), the \(FPR = .058\), and the \(ACC = .925\).
Figure 6.3: L-curve for Measured Training Data using Median Loiter Speed and Hausdorff Feature

The ability to visualize the classifier in two dimensions as shown in Figure 6.4 provides insight into what is happening with the SVM classifier. As can be seen, the decision surface does, indeed, go through the support vectors and encloses all of the training data. Also, the data is smoothly enclosed by the decision surface which is the result of normalizing the features and the $\sigma$ selection process. This visualization also helps explain what is happening for higher dimension SVM’s whose decision surface cannot be visualized.

Figure 6.5 provides a visualization that would be useful to the guard or operator as the specific trajectories used for support vectors are displayed as overlays on the video imagery. This overlay allows the tracks to be visualized to see if these “outliers” should be considered threats given the current threat situation, and also, the classified anomalies depicted in Figure 6.6 could be shown to the operator to determine if these detected anomalies are indeed considered anomalous. This feedback from the operator as described in Chapter 3, is the essential advantage of the anomaly detection approach in this thesis.

It is also useful to delve into the specific results in more detail to explain the results based on the physically based features. The two features used were the Loiter Feature and the Centered Modified Hausdorff Feature or equivalently the Min Max...
Distance Centered Feature. The Loiter Feature was described previously in Chapter 4 and illustrated in Figure 4.2. As its name indicates, it determines if pedestrians are loitering or "just standing around" in the scene. This behavior would be considered anomalous if loitering was not permitted and, hence, would not be in the training data as the training data only contains "normal" behavior. The Modified Hausdorff feature was also described in Chapter 4 and depicted in Figure 4.3. Figure 4.3 provides insight into how the feature is calculated, but it is also useful to get an intuitive feel for the feature as well. Figure 6.7 shows that the Hausdorff distance between two tracks or two point sets finds the largest, "closest" distance between the two sets where as the Point of Closest Approach finds the smallest, "closest" distance between the two sets. Hence, the Hausdorff distance will detect any deviation from a track path which is clearly important for finding anomalous tracks. Conversely, the Point of Closest Approach will detect any time a track gets close to another track or perhaps a barrier such as a fence which also can be represented as a set of points. Thus, the two features, Centered Modified Hausdorff Feature and the Loiter Feature are complementary as they measure shape and speed, respectively.
For example, Figure 6.8 shows the correct anomaly detections. In the upper left hand corner, the person on the skateboard and the four pedestrians that stopped together are anomalies that were sensitive to the speed or Loiter Feature whereas the pedestrian that veered off to the right created a track that was sensitive to the shape or Centered Modified Hausdorff Feature. Clearly, the two anomalies in the upper right hand corner were people on bicycles that were sensitive to the Loiter Feature as the minimum short duration speed was faster than the minimum speed short duration speed of normally walking pedestrians. Also in the lower left hand corner, sensitivity to the Loiter or speed feature is likely due to the two bicycles and the skateboard. However, in addition, the trajectories taken on two of the tracks seem to be atypical of a normal pedestrian path. Finally on the lower right panel, the speed feature would be sensitive to both the bicycles and the pedestrians who are stopped and apparently talking.

In Figure 6.9, the panel on the left represents incorrect detections of anomalies, and the panel on the right represents anomalies that were missed. For the incorrect detections in the left panel, these detections represent interactions between pedestrians. The single track to the right interacted and walked around the four people stopped shown in the upper right panel in Figure 6.8 where as the two tracks on the left are walking together. Although these tracks were not considered anomalous, their false detections are understandable as this system does not have features
that detect these type of interactions. For the anomalous tracks on the right hand panel, all the people were either on bicycles or skateboards; hence, they should be sensitive to the Loiter Feature. It may be that their speed was not sufficiently fast to be considered anomalous. Since the data set had high enough resolution to have many pixels on the pedestrians, many of the anomaly detection approaches that use video can estimate the shape of the pedestrian to detect bicycles, etc. For track only anomaly detectors as was developed in this thesis, this shape information is not available to the classifier and, hence, the detection of slowly moving bicycle type tracks with track only information may be problematic.

6.2.2 Simulated Data Results

For the simulated data test, the same features are tried: Loiter Feature and the Centered Modified Hausdorff Feature. Using these two features, $\sigma$ is found as shown in Figure 6.10 and the resulting decision surface surrounding the training data is shown on the left panel of Figure 6.11. The right panel of Figure 6.11 shows the training data, decision surface, and the test data. The operator can observe the training tracks including those which were selected as support vectors in Figure 6.12. As with the measured data, the left panel shows all the training data with the tracks corresponding to the support vectors shown in red whereas the right panel
shows just the support vectors which are individually labeled and identified. If some of the support vectors were considered anomalous, the operator could remove these tracks from the training set and retrain the classifier. Continuing on this example, Figure 6.13 shows the detected anomalies. Again, the left panel includes all the test data including the detected anomalies in red, and the right panel shows just the data detected as anomalies. These anomalies are labeled and identified. With these two features, the performance numbers were ACC = .9615, the FPR = .0395, and the TPR = 1.

It is useful to add a new feature to the classifier to show how to adapt the classifier and change its behavior. In this case, the operator is concerned that the pedestrians are getting too close to the right fence. Hence, the Point of Closest Approach (PoCA) feature described above is added to the classifier to sensitize the classifier to proximity to the fence. If a track was close to the fence (i.e., closer than any of the training data), that track would be considered an anomaly. In this case the Loiter Feature is dropped and the PoCA feature is added. Figures 6.14, 6.15, 6.16, and 6.17 show the sequence of plots associated with the classifier. Two observations are of interest. First, as designed, the tracks close to the fence are now classified as anomalies. This can be seen by looking at Figure 6.18 which is a blow
Figure 6.8: Correctly Detected Anomalies

Figure 6.9: Correctly Detected Anomalies
Figure 6.10: L-curve for Synthetic Training Data using Median Loiter Speed and Hausdorff Feature

up of Figure 6.17. The second observation is that the performance improved over the first two set of features with the scores of ACC = .9773, the FPR = .0233, and the TPR = 1. In this case, the anomalies near the fence are counted as misses; yet, performance is still better than before. It is clear, that the main difference between the normal data and the anomalies are shape rather than speed for the simulations run for these experiments. The operator controls how close is close to the fence through the training examples. If a particular training example is too close to the fence, it is dropped as a training track. Hence, the operator controls the boundary by selection of training examples which is more intuitive than the tuning of algorithm parameters or the choice of a confidence number.

6.3 Summary

Both simulated and measured data were used in experiments to test the anomaly detection system and process. Two features were selected for both the measured and simulated data to allow visualization of the decision boundaries to understand the operation of the OC-SVM for this application. Tracks were also visualized both in training to show the tracks selected as support vectors and in testing to show which
tracks were detected as anomalies. The physically based features used were explained in detail to show that the classifier behavior made sense and was understandable. Finally, the PoCA feature was added to show how the physically motivated features could be used to change the behavior of the classifier to find anomalies based on the operator needs.
Figure 6.12: 2 Feature Support Vector Trajectories

Figure 6.13: 2 Feature Declared Anomalies
CHAPTER 6. EXPERIMENTS

Figure 6.14: L-curve for Synthetic Training Data using Median Loiter Speed and Hausdorff Feature

Figure 6.15: Decision Surfaces for 2D Single Class SVM
Figure 6.16: 2 Feature Support Vector Trajectories

Figure 6.17: 2 Feature Declared Anomalies
Figure 6.18: 2 Feature Declared Anomalies
Chapter 7

Conclusions and Recommendations

7.1 Conclusions

An anomaly detection approach was developed for monitoring entrances at installations to take into account the dynamic threat level that often is associated with security applications. A principle strategy associated with this approach was to involve an operator in adapting the sensitivity of the anomaly detector to the threat level. It was anticipated that the operator would not be an algorithm expert, so the adaptation either amounted to identifying tracks as either normal or anomalous, or by suggesting physically based features that would be sensitive to a threatening behavior. The operator would not be required to tune any algorithm parameters as all algorithm parameters were automatically tuned given selection of the training tracks. Further, the operator could also examine results overlayed on photos during operation and adapt the performance of the anomaly detector by pointing out errors in operation. Given these errors, the training set could be adjusted which results in the movement of the decision boundary to be consistent with the operator sensitivities. As referenced above, in some circumstances, features may be added to detect some anomalies that are not detectable with the current feature set.

The features were structured in a systematic way to promote the understanding of the features used and to facilitate the addition of new features to the system. Further, given the definition of tracks as 2D space / time, synchronously sampled trajectories, the types of features that could be defined are constrained by this definition. The type of features that could be defined was further constrained by the requirement that the features be based on physical principles which are again limited to the dynamics of tracks. In particular, these dynamics were decomposed into local speed and heading features, and into shape features which compare two track trajectories. Also, the features were decomposed into a sequential set of operators that either transform tracks and associated vectors or reduce the tracks from tracks to vectors or vectors to scalars. Again the physically understandable features and the feature structure help the operator to understand the anomaly detector operation
and help the operator to add features to the system when desired.

Although in principle this system approach could run with any classifier approach, the one class support vector machine was selected for use in this system because of its very consistent performance, the ability to run it with a single tuning parameter, and most importantly, its interpretable decision boundary. These properties allow the operator to control the anomaly detector’s performance by training data selection without having to be concerned about any parameter tuning. An automatic parameter tuning algorithm was developed to work with the one class SVM classifier which is more efficient than cross validation approaches. Also, since the support vectors live on the decision boundary, the operator always has access to the tracks that are on the edge of normal versus anomalous behavior. This definition of the decision boundary is concrete and promotes manipulation of the decision boundary by inclusion or exclusion of the support vectors as part of the training data.

The anomaly detection system was tested on both measured and simulated tracks with very good results. But more importantly, the results were very understandable based both on the physical features and the decision boundary. The tests were performed with two features mainly to allow visualization of the decision boundary to facilitate understanding of the classifier and the anomaly detection system. However, several tests were performed with multiple features but were not included in this report as the performance was not more informative or any better than the two feature classifiers. The experimentation included the demonstration of adapting the classifier by adding features to sensitize the classifier to anomalies that the operator wanted to detect.

7.2 Future Recommendations

Although the anomaly detection was designed with an operator in the loop, it was only tested with an imaging scientist in the loop. It would be very important to actually test this system with an operator in the loop to validate its primary hypothesis that the system is operator friendly. However, for this experimentation to take place, several extensions to the work should be considered. First, it would be important to integrate this classifier with an actual video tracker. Currently, the video tracks that were used were obtained by using a human to track the pedestrians. This step is labor intensive and would not be amenable to a field test situation due to the slow turn around time between the collected video and the available tracks. Secondly, the software is research quality but would require some improvements for the software to be as modular as its design. For large data sets, it would also be
useful to write the code in a language like C to improve the speed of the anomaly detection system over its current MATLAB implementation.

Another area of improvements would include adding additional capability to the anomaly detection algorithms. One area where improvements may be achievable is automatic feature selection. The current system selects the features manually which is facilitated by the physical interpretation of the features, but automatic feature selection could improve performance and potentially make the system even more usable by the operator. To be consistent with the physical understanding of the features, the subset type of feature selection would be most appropriate.

Finally, additional testing of the system would be desirable. It would be useful to better understand the strengths and weaknesses of the system. The availability of a large set of video tracks would greatly facilitate additional testing. In lieu of a set of tracks, integrating this system with a reliable video tracker would significantly increase the available data for testing.
Appendices
Appendix A

Code Description

This section will include a description of the code developed on this effort. The code will include the features, classifier, and a script that runs simulated data that exemplifies the anomaly detection approach.

A.1 Anomaly Detection Description using Simulated Tracks

Figure A.1 shows the flow of the anomaly detection algorithm using the simulated data and algorithm code listed in Appendix B. The Experimental Script is shown to the left of Figure A.1 and consists of the scripts and functions shown in the middle of the figure. The first step in the flow chart is to generate the simulated tracks as shown on the left side of Figure A.2. Next, feature extraction is performed using the functions shown to the right in Figure A.1. After these features are extracted, they are put into a feature matrix and then normalized.

Next sigma is determined as described in Chapter 5, and the fitted L-curve resulting from this process is shown in Figure A.3. Sigma is then used to train the one class SVM resulting in the support vectors being identified as shown on the left side in red and identified on the right side of Figure A.2. The resulting classification boundary resulting from these support vectors are shown on the left side of Figure A.4.

From the support vectors, additional parameters required by the classification function are computed. Next, the test data is generated and fed into the feature extractor. The same features are extracted and are then normalized using the normalization parameters calculated from the training data. These features are fed into the classification stage of the SVM described in Chapter 5, and the results shown in Figure A.5. The left side has all the test tracks with the anomalies detected shown in red, and the right side has the normal tracks misclassified as anomalies. In addition, the right side of Figure A.4 has the test tracks plotted in conjunction with the decision boundary. The classification results are given in Table A.1.
Figure A.1: Program Sequence for Simulation Example
Figure A.2: Training Data and Support Vectors

Figure A.3: L-curve fit to determine $\sigma$
Figure A.4: Classification Boundaries and Classification Results

Figure A.5: Classification Results
Classification Results

<table>
<thead>
<tr>
<th>True Positives</th>
<th>30</th>
</tr>
</thead>
<tbody>
<tr>
<td>True Negatives</td>
<td>25</td>
</tr>
<tr>
<td>False Positives</td>
<td>5</td>
</tr>
<tr>
<td>False Negatives</td>
<td>0</td>
</tr>
<tr>
<td>False Positive Rate</td>
<td>0.167</td>
</tr>
<tr>
<td>True Positive Rate</td>
<td>1.000</td>
</tr>
<tr>
<td>Accuracy</td>
<td>0.917</td>
</tr>
</tbody>
</table>

Table A.1: Classification Results
Appendix B

Source Code

B.1 Feature Extraction

B.1.1 Calls and runs all the features

featureExtraction.m

function F = featureExtraction(Xtest,Xcomp)

%% Center Tracks
XtestC = tracksCenter(Xtest,'mean');
XcompC = tracksCenter(Xcomp,'mean');

%% Compute Centered Modified Hausdorff
F.CentModHaus = compareTracks(XtestC,XcompC,'min','max','min',
    'min',1);

%% Compute Modified Hausdorff
F.ModHaus = compareTracks(Xtest,Xcomp,'min','max','min',
    'min',1);

%% Compute Speed Vector
Xspeed = computeSpeeds(Xtest, 5);

%% Compute Average Speed
F.SpeedMean = compute1Dscalar(Xspeed,'mean');

%% Compute Max Speed
F.SpeedMax = compute1Dscalar(Xspeed,'max');

%% Compute Windowed Speed Vector
XWspeed = computeWindowedSpeeds(Xspeed, 40, 'median');

%% Compute Windowed Speed Minimum
F.WinMedSpeedMin = compute1Dscalar(XWspeed,'min');

%% Compute Windowed Speed Maximum
F.WinMedSpeedMax = compute1Dscalar(XWspeed,'max');

%% Compute Heading Vector
Xhead = computeHeadings(Xtest, 5);
APPENDIX B. SOURCE CODE

%% Compute Mean Heading
F.HeadMean = compute1Dscalar(Xhead,'mean');

%% Compute Vorticity Vector
Xvorticity = computeVorticity(Xtest, 5);

%% Compute Mean Vorticity
F.VortMean = compute1Dscalar(Xvorticity,'mean');

B.1.2 Filter Feature Extractors – input: tracks, output: tracks

tracksCenter.m

function ctracks = tracksCenter(X, func)
  % Input
  % Output
  C = length(X);
  for ii = 1:C
    N1 = length(X{ii});
    s = ['(X{ii})'];
    mu = eval(s);
    ctracks{ii,1} = X{ii} - repmat(mu, N1, 1);
  end

B.1.3 Reduce Feature Extractors – input: tracks, output: vectors

computeSpeeds.m

function out = computeSpeeds(tracks, w);
  % Function: computes speed of tracks over local window and store
  % in cell array
  % Inputs
  % (C x 1) cell array of tracks
  % w = window length to compute speed
  % Output
  % (C x 1) cell array of speed vectors
  [C n] = size(tracks);
  for ii = 1:C
    int = speed(tracks{ii},w);
    out{ii,1} = int(w + 1:end);
  end
function out = speed(track, w)
% Function: computes speed of tracks over local window
% INPUTS:
% track = x y data in rows for (N x 2) matrix
% w = length of window, scalar
% OUTPUT:
% out = vector of speeds

N = length(track(:,1));

for tt = w + 1 : N
    out(tt,1) = norm(track(tt,:) − track(tt−w,:))./w;
end

function out = computeHeadings(tracks, w);
% Function: computes headings of tracks over local window and store
% in cell array
% Inputs
% (C x 1) cell array of tracks
% w = window length to compute heading
% Output
% (C x 1) cell array of local heading vectors

[C n] = size(tracks);
for ii = 1:C
    int = heading(tracks{ii},w);
    ii_nan = find(isnan(int));
    int(ii_nan) = [];
    int = reshape(int,numel(int)/2,2);
    out{ii,1} = atan2(int(w + 1:end,2),int(w + 1:end,1));
end

function h = heading(X, w)
% INPUTS:
% X = x y data in rows for (N x 2) matrix
% w = length of window. scalar
% OUTPUT:
% h = direction the track is heading [200 x 2]

N = length(X(:,1));

for tt = w + 1 : N
    h(tt,:) = (X(tt,:)−X(tt−w,:))./norm(X(tt,:)−X(tt−w,:));
end
computeVorticity.m

function out = compute_vorticity(tracks, w)
%% Function:
% Measure goodness of fit of line to local window and store in
% cell array
% Measures wiggliness of track
%% INPUTS:
% tracks = (C x 1) cell array of tracks
% w = length of window, scalar
% OUTPUT:
% out = cell array of vorticity vectors

[C n] = size(tracks);
for ii = 1:C
    out{ii,1} = vorticity(tracks{ii}, w);
end

function out = vorticity(X, w)
%% Function:
% Measure goodness of fit of line to local window
% Measures wiggliness of track
%% INPUTS:
% X = x y data in rows for (N x 2) matrix
% w = length of window, scalar
% OUTPUT:
% out = Vorticity vector (N x 1)
N = length(X(:,1));
for ii = 1:N−w
    XT = X(ii:ii+w,:);
    XF = fit_line(XT);
    out(ii,1) = norm(XT−XF)./w;
end

function XF = fit_line(X)
%% Function
% fits line to sequence of N dimensional points
N = length(X);
mu = mean(X);
XC = X − repmat(mu, N, 1);
[U Sig V] = svd(XC'*XC/(N−1));
XF = XC*U(:,1)*U(:,1)' + repmat(mu, N, 1);

B.1.4 Filter Feature Extractors – input: vectors, output: vectors

computeWindowedSpeeds.m
APPENDIX B. SOURCE CODE

function out = computeWindowedSpeeds(Xspeeds, w, func)
" Function: computes a function on moving window on cell array of
" speed vectors
" e.g., computing mean on moving window gives moving average filter
" Inputs
" (C x 1) cell array of speeds
" w = window length to compute windowed function on speed
" Output
" (C x 1) cell array of windowed speed vectors

[C n] = size(Xspeeds);
for ii = 1:C
    out{ii,1} = win_filt_1D(Xspeeds{ii}, w, func);
end

function out = win_filt_1D(X, w, func)
" Function: computes a function on moving window on speed vector
" e.g., computing mean on moving window gives moving average filter
" Inputs
" X = vector of speeds (N x 1)
" w = length of window
" Output
" Out = vector that has been moving window filtered

N = length(X(:,1));
if w ≥ N
    out(1,1) = eval(['(X)']);
end
for tt = w + 1 : N
    out(tt-w,1) = eval(['(X(tt-w:tt,1))']);
end

B.1.5 Reduce Feature Extractors – input: vectors, output: scalars

compute1Dscalar.m

function out = compute1Dscalar(Xvec, func);
" Function
" Input
" Output

[C n] = size(Xvec);
for ii = 1:C
    out(ii,1) = eval(['(Xvec{ii})']);
end
APPENDIX B. SOURCE CODE

B.1.6 Comparison Features – input: 2 sets of tracks, output: scalar

`compareTracks.m`

```matlab
function out = compareTracks(track1, track2, func1,...
    func2, func3, func4, Sflag)
    % function
    % compares every combination of the tracks in track1 with the tracks
    % in track 2
    % e.g., func1 = 'min', func2 = 'min', func3 = 'min', func 4 = 'min'
    % computes the closest point of closest approach among all the pairs
    % of track1 and track2
    % input
    % track1 = cell array of tracks
    % track2 = cell array of tracks
    % func1, func2, func3, func4 are function that are applied
    % Sflag ==1 if track1 = track2
    % output
    % scalar
    % initialize
    [C1 n] = size(track1);
    [C2 n] = size(track2);

    for ii = 1:C1
        tracka = track1{ii};
        for jj = 1:C2
            trackb = track2{jj};
            if ii ~= jj || ~Sflag % proceed if index different or not the same
                disVec(jj) = gen_distance(tracka,trackb,...
                    func1, func2, func3);
            else
                disVec(jj) = inf;
            end
        end
        out(ii,1) = eval(['(disVec)']);
    end

    function out = gen_distance(a,b,func1,func2,func3)
    N1 = length(a);
    N2 = length(b);

    N = round(numel(a)/N1);
    a = reshape(a,N1,N);
```
b = reshape(b,N2,N);

%% Create Matrix of all Pairwise Distances
disMat = zeros(N2,N1);
for ii = 1:N1
c = b - repmat(a(ii,:),N2,1);
disMat(:,ii) = sqrt(sum(c.^2,2));
end

%% First Comparison (point to point based on each track)
% distance from a to b, and b to a
% output 2 vectors
switch func1
  case {'max','min'}
disVec = eval(['[func1 '(disMat, [], 1)']']);
  case {'mean','median','mode'}
disVec = eval(['[func1 '(disMat, 1)']']);
end

switch func1
  case {'max','min'}
disVec1 = eval(['[func1 '(disMat, [], 2)']']);
  case {'mean','median','mode'}
disVec1 = eval(['[func1 '(disMat, 2)']']);
end

%% Second Comparison (Reduce 2 vectors to scalars)
% computes an operator on each vector, the a to b vector, and
% the b to a vector
% output 2 scalars
discale = eval(['[func2 'disVec']']);
discale1 = eval(['[func2 'disVec1']']);

%% Third Comparison (compare the two scalars)
% compares the a to b scalar and the b to a scalar
% output 1 scalar
out = eval(['[func3 'discale, discale1']]);

B.2 Classifier functions

B.2.1 Centering and Normalization

centNormMedianTrain.m

function [Y mu disp] = centNormMedianTrain(X)
% Robust Centering and Normalization using median and MAD statistics for
APPENDIX B. SOURCE CODE

% use with training vectors
% INPUT: Matrix X (Features x Tracks)
% OUTPUT: Centered and Normalized Matrix Y (Features x Tracks)
% mu = row vector of medians
% disp = row vector of dispersions
mu = median(X);
y = X - repmat(mu, size(X,1), 1);
disp = madn(y) ./ .6745;
Y = y ./ repmat(disp, size(y,1), 1);

function f = madn(x)
mu = median(x);
f = median(abs(x - repmat(mu, size(x,1), 1)));

centNormMedianTest.m

function [Y] = centNormMedianTest(X, mu, disp)
% Robust Centering and Normalization using median and MAD statistics for
% use with test vectors
% mu = row vector of medians
% disp = row vector of dispersions
% INPUT: Matrix X (Features x Tracks)
% OUTPUT: Centered and Normalized Matrix Y (Features x Tracks)
y = X - repmat(mu, size(X,1), 1);
Y = y ./ repmat(disp, size(y,1), 1);

B.2.2 Find Sigma

Lsigma.m

function [sig_out fit] = Lsigma(data,k2,k1)
% Lsigma.m finds the "optimal" sigma by fitting line
% segments to the sigma vs log(percent support vectors)
% and heuristically finding sigma based on the relative
% angles of the intersecting line segments
% INPUT: data = data matrix (features x tracks)
% k1, k2 = scalar parameters for finding sigma from L curve
% OUTPUT: sig_out = scalar which is calculated sigma value
% fit is structure fit.X1, fit.XF1, fit.X2, fit.XF2. fit.X3, fit.XF3 used to
% plot the three line segment fit for L-curve visualization

% initialize
sigma = [.3:.025:6];
Nsigma = length(sigma);
[Ndata Nfeat] = size(data);
per_sv = zeros(Nsigma, 1);
%% iterate for each sigma
for ii = 1 : Nsigma
    u = trainSVMocRBF(data,sigma(ii));
    sv = find(u > .0001);
    per_sv(ii) = length(sv)./Ndata;
end

%% fit line segments
XX = [sigma' log(per_sv)];
% set parameters and initialize
nn = 3; % length of smallest line segment

err = zeros(Nsigma);
for ii = nn+1:Nsigma
    for jj = ii+nn:Nsigma-2*nn
        X = XX(1:ii,:);
        XF = fit_line(X);
        err1 = norm(XF-X);

        X = XX(ii:jj,:);
        XF = fit_line(X);
        err2 = norm(XF-X);

        X = XX(jj:Nsigma,:);
        XF = fit_line(X);
        err3 = norm(XF-X);

        err(ii,jj) = err1 + err2 + err3;
    end
end

%% Find lowest error fit ("best" fit)
kk = find(err == 0);
err(kk) = inf;

pp = find(min(err(:)) == err(:));
[ii jj] = ind2sub([Nsigma Nsigma],pp);

corner1 = sigma(ii);
corner2 = sigma(jj);

%% determine angles of best fit segments
fit.X1 = XX(1:ii,:);
fit.XF1 = fit_line(fit.X1);
angA = ang_deg(fit.XF1(1,:),fit.XF1(end,:));
APPENDIX B. SOURCE CODE

fit.X2 = XX(ii:jj,:);
fit.XF2 = fit_line(fit.X2);
angB = ang_deg(fit.XF2(1,:),fit.XF2(end,:));

fit.X3 = XX(jj:Nsigma,:);
fit.XF3 = fit_line(fit.X3);
angC = ang_deg(fit.XF3(1,:),fit.XF3(end,:));

if angB−angA > 35
    sig_out = corner1 + k1;
elseif angB − angA > angC − angB
    sig_out = corner1 + k2;
else
    sig_out = corner2 + k1;
end

function XF = fit_line(X)
N = length(X);
mu = mean(X);
XC = X − repmat(mu,N,1);
[U Sig V] = svd(XC'*XC/(N−1));
XF = XC*U(:,1)*U(:,1)' + repmat(mu,N,1);

function ang = ang_deg(pt1,pt2)
ang = 180/pi*atan((pt1(1,2)−pt2(1,2))/(pt1(1,1)−pt2(1,1)));

B.2.3 One class SVM using RBF – Training

trainSVMocRBF.m

function u = trainSVMocRBF(data,sigma)
% trains the RBF One Class SVM using MATLAB "quadprog"
% which is a quadratic programming optimization routine
% requires the optimization toolbox
% INPUT: feature vectors matrix (features x tracks)
% sigma = sigma parameter which controls smoothness
% OUTPUT: u = support vectors

% Set the parameters for "quadprog"
[n1 n2] = size(data);
f = zeros(n1,1);
A = −eye(n1);
b = zeros(n1,1);
Aeq = ones(1,n1);
beq = 1;
lb = []; ub = []; x0 = [];
options = optimset('Display','off');
warning('off','all');

%% Calculate Training RBF
K = rbfTrain(data,sigma)

%% Perform Optimization
u = quadprog(K,f,A,b,Aeq,beq,lb,ub,x0,options);

%% Reset Optimization and Warnings
options = optimset('Display','on');
warning('on','all');

%% Additional functions to calculate RBF kernel
function K = rbfTrain(xTrain,sigma)
% efficiently computes square K
arg = dis_sq(xTrain);
K = exp(-arg./(2*sigma.^2));

function D = dis_sq(X)
% computes distance squared
% taken from book
% Convex optimization and Euclidean distance geometry
% Dattorro, J., 2005
[N,n] = size(X);
e = ones(N,1);
d = diag(X*X');
D = d*e' + e*d' - 2*X*X';

B.2.4 Calculate SVM Internal Parameters

SVMparam.m

function param = SVMparam(u,trainData,param)
% Generates parameters for SVM after training
% u = result of optimization with training data
% param.ind = indices of support vectors
% param.dVal = data values of support vectors
% param.uVal = u values of support vectors
% param.sigma = sigma
% param.rho = rho

param.ind = find(u > .0001);
param.dVal = trainData(param.ind,:);
param.uVal = u(param.ind);
K = rbfTrain(param.dVal,param.sigma);
param.rho = mean(K*param.uVal);

function K = rbfTrain(xTrain,sigma)
% efficiently computes square K
arg = dis_sq(xTrain);
K = exp(-arg./(2*sigma.^2));

function D = dis_sq(X)
% computes distance squared
[N,n] = size(X);
e = ones(N,1);
d = diag(X*X');
D = d*e' + e*d' - 2*X*X';

B.2.5 One class SVM using RBF – Testing
testSVMocRBF.m

function y = testSVMocRBF(testData, param)
% SVM Classification
% INPUT: testData = matrix (features x tracks)
% param = parameter data structure
% OUTPUT: y = classification test function
K = rbfTest(testData,param.dVal,param.sigma)
y = K*param.uVal - param.rho;

function kernel=rbfTest(x1,x2,sig)
% Compute Radial Basis Kernel for training
% x1 and x2 can be different dimensions in rows but
% same dimensions in columns where rows = number of training vectors
% and columns = number of features
[n1 dum] = size(x1);
[n2 dum] = size(x2);
kernel = zeros(n1,n2);
for ii = 1:n1
    for jj = 1:n2
        kernel(ii,jj) = exp(-norm(x1(ii,:)-x2(jj,:)).^2./(2*sig.^2));
    end
end
B.3 Experiment with Simulated Data

B.3.1 Experiment Script

**SVMsymExp.m**

% Experiment with Simulated Data to understand how anomaly detection works

clear,close
rng(1)

%% Plot Number
n1 = 1; n2 = 2; n3 = 3; n4 = 4;

%% Get Data
genSynData

%% Run vs. Get Stored Results
% % Choose features and if two features, choose whether to plot
% decision surface

featureSelect = [1 5]; % [1 5] % [1 5 8] % [1 5 6 8] % [1 2 5 6 8]
num_features = length(featureSelect);
feature_name = {'Centered Modified Hausdorff', 'Modified Hausdorff',...
    'Mean Speed', 'Max Speed', 'Min Windowed Speed', 'Max Windowed Speed',...
    'Mean Heading', 'Mean Vorticity'};
if num_features == 2, twoD_plot = 1; else twoD_plot = 0; end

%% Training data
X_cell_train = X_cell_train';

%% Feature Extraction of Training Data
F = featureExtraction(X_cell_train,X_cell_train);
xtrain = [F.CentModHaus F.ModHaus F.SpeedMean F.SpeedMax...
    F.WinMedSpeedMin F.WinMedSpeedMax F.HeadMean F.VortMean];

%% Train Classifier
xtrain = xtrain(:,featureSelect);
% Center and Normalize Data
[out mu disp] = centNormMedianTrain(xtrain);
k1 = .2; k2 = 1.0;
[paramsigma fit] = Lsigma(out,k2,k1);

%% Train SVM with Chosen Sigma
u = trainSVMocRBF(out,paramsigma);
params = SVMparam(u,out,params);
%% Plot Results
if twoD_plot == 1
    figure(n1)
    subplot(121)
    plot2dDecSurf(out,param)
    hold on
    for ii = 1:length(param.ind)
        text(out(param.ind(ii),1),...
        out(param.ind(ii),2),num2str(param.ind(ii)))
    end
    hold off
    title({['2D Single Class SVM Classification'],...
        ['Training Data, \sigma = ',...
        num2str(param.sigma)]},'FontSize',14)
    xlabel(feature.name(featureSelect(1)),'FontSize',12)
    ylabel(feature.name(featureSelect(2)),'FontSize',12)
end

figure(n4)
plotLcurveFit

figure(n2)
im = 'layover.jpg';
im = [];
title1 = 'Training Trajectories';
title2 = 'Support Vector Trajectories';
plotTraj(X_cell_train,im,param.ind,0, title1,title2);

%% Testing Data
X_cell_test = X_cell_test';

%% Compute Features of Test Data
Ft = featureExtraction(X_cell_test,X_cell_train);
xtest = [Ft.CentModHaus Ft.ModHaus Ft.SpeedMean Ft.SpeedMax...
        Ft.WinMedSpeedMin Ft.WinMedSpeedMax Ft.HeadMean Ft.VortMean];

%% Select Features
xtest = xtest(:,featureSelect);

%% Center and Normalize Data
[outTest] = centNormMedianTest(xtest, mu, disp);

%% Test SVM
y = testSVMocRBF(outTest,param);
ind_test = find(y < .0001);
if twoD_plot == 1
    figure(n1), subplot(122)
    plot2dDecSurf(out, param)
    hold on, plot(outTest(:,1), outTest(:,2), '.r',... 
                   outTest(ind_test,1), outTest(ind_test,2), 'or'), hold off
    hold on
    for ii = 1:length(ind_test)
        text(outTest(ind_test(ii),1),...
            outTest(ind_test(ii),2), num2str(ind_test(ii)))
    end
    hold off
    title({['2D Single Class SVM Classification'],...
            ['Training and Testing Data, \sigma = ',...
             num2str(param.sigma)],},'FontSize',14)
    xlabel(feature_name(featureSelect(1)),'FontSize',12)
    ylabel(feature_name(featureSelect(2)),'FontSize',12)
    axis([min(outTest(:,1))−1 max(outTest(:,1))+1 ...
          min(outTest(:,2))−1 max(outTest(:,2))+1])
end

%% Plot results
figure(n3)
title1 = 'Test Trajectories';
title2 = 'Mis-Classified Outlier Trajectories';
plotTraj(X_cell_test, im, ind_test, 30, title1, title2);

%% Compute Score
MeasScoreSym

MeasScoreSym.m

% Compute Score

truth = [1:30];
for ii = 1:length(truth)
    truePos(ii) = sum(truth(ii)==ind_test);
end

Total = length(xtest);
Positives = length(truth);
Negatives = Total − Positives;

TP = sum(truePos);
TPR = sum(TP)/Positives;

for ii = 1:length(ind_test)
falsePos(ii) = ~sum(ind.test(ii)==truth);
end

FP = sum(falsePos);
FN = Positives - TP;
TN = Negatives - FP;
FPR = FP / Negatives;
ACC = (TP + TN) / Total;

fprintf('  \nTrue Positives = %i
',TP);
fprintf('True Negatives = %i\n',TN);
fprintf('False Positives = %i\n',FP);
fprintf('False Negatives = %i\n',FN);
fprintf('False Positive Rate = %2.3f\n',FPR);
fprintf('True Positive Rate = %2.3f\n',TPR);
fprintf('Accuracy = %2.3f\n',ACC);

% This script generates synthetic training tracks and test tracks
% to demonstrate operation of classifier
% OUTPUTS: X_cell_test — cell array with 60 cells (tracks) each 100 x 2
% representing the x and y location of 100 track points
% the 60 tracks are distributed with 30 normal tracks and 30 anomalous
% tracks
% OUTPUTS: X_cell_train — cell array with 30 cells (tracks) each 100 x 2
% representing the x and y location of 100 track points
% the 30 tracks are all normal
% The script also plots the Training Data and the Test Data

subplot(122)
for ii = 1:15 % Anomaly Data — Testing
x = [ 0 .25 .5 .75 1];
x = x + .1.*rand(1,5);
y = .2*rand(1,5);
xtest1 = linspace(min(x),max(x),100)';
ytest1 = interp1(x,y,xtest1,'spline');
plot(xtest1,ytest1,'.',x,y,'or')
hold on
x_cell_test1{ii} = [xtest1 ytest1];
end

for ii = 1:30 % Normal Data — Testing
APPENDIX B. SOURCE CODE

x = [ 0 .5 1];
x = x + .1.*rand(1,3);
y = [-.05 .05 -.05];
y = y + .1* rand(1,3);
xtest3 = linspace(min(x),max(x),100)';
ytest3 = interp1(x,y,xtest3,'spline');
plot(xtest3,ytest3,'.m',x,y,'or')
x_cell_test3{ii} = [xtest3 ytest3];
end

for ii = 1:15 % Anomaly Data — Training
x = [ 0 .25 .5 .75 1];
x = x + .1.*rand(1,5);
y = .2*rand(1,5);
indx = ceil(2*rand+1);
xtest2 = [linspace(min(x),x(indx),25)';
            linspace(x(indx),x(indx+1),50)';
            linspace(x(indx+1),max(x),25)'];
ytest2 = interp1(x,y,xtest2,'spline');
plot(xtest2,ytest2,'.g',x,y,'or')
x_cell_test2{ii} = [xtest2 ytest2];
end
hold off
axis([0 1.25 -.1 .3])
title('Testing Data')
X_cell_test = [x_cell_test1 x_cell_test2 x_cell_test3];

subplot(121)
for ii = 1:30 % Normal Data — Training
x = [ 0 .5 1];
x = x + .1.*rand(1,3);
y = [-.05 .05 -.05];
y = y + .1* rand(1,3);
xtrain1 = linspace(min(x),max(x),100)';
ytrain1 = interp1(x,y,xtrain1,'spline');
plot(xtrain1,ytrain1,'.m',x,y,'or')
hold on
X_cell_train{ii} = [xtrain1 ytrain1];
end
axis([0 1.25 -.1 .3])
hold off
title('Training Data')

B.3.2 Plots

plot2dDecSurf.m
function plot2DDecSurf(trainData, param)

%% Determine Dimensions of plot
px1 = floor(min(trainData(:,1))); px2 = ceil(max(trainData(:,1)));
py1 = floor(min(trainData(:,2))); py2 = ceil(max(trainData(:,2)));
pxm = px2 - px1; pym = py2 - py1;
px1 = px1 - pxm/2; px2 = px2 + pxm/2;
py1 = py1 - pym/2; py2 = py2 + pym/2;
px = linspace(px1, px2, 50);
py = linspace(py1, py2, 50);

%% Create test vector from grid
[xx yy] = meshgrid(px, py);
testData = [xx(:) yy(:)];

%% Classify points in plane
yout1 = testSVMocRBF(testData, param);
yout = reshape(yout1, size(xx));

%% Plot
contour(xx, yy, yout, [0 0], 'k', 'LineWidth', 2);
hold on, plot(trainData(:,1), trainData(:,2), '.
plot(param.dVal(:,1), param.dVal(:,2), 'o', 'MarkerSize', 16), hold off

plotLcurveFit.m

function plotLcurveFit

%% Plot L-Curve Fit
plot(fit.XF1(:,1), fit.XF1(:,2), 'r', fit.X1(:,1), fit.X1(:,2), ...
    'b', 'MarkerSize', 14, 'LineWidth', 4)
hold on
plot(fit.XF2(:,1), fit.XF2(:,2), 'r', fit.X2(:,1), fit.X2(:,2), ...
    'b', 'MarkerSize', 14, 'LineWidth', 4)
plot(fit.XF3(:,1), fit.XF3(:,2), 'r', fit.X3(:,1), fit.X3(:,2), ...
    'b', 'MarkerSize', 14, 'LineWidth', 4)
xlabel('\sigma values', 'FontSize', 12)
ylabel('log(number of support vectors)', 'FontSize', 12)
hold off
title(['L-curve fit, \sigma = ', num2str(param.sigma)], 'FontSize', 14)

plotTraj.m

function plotTraj

%% PLOT RESULTS
subplot(121)
plot_allTrajectories(X, im_name, '.', 0);
colormap gray
```matlab
function plot_all_trajectories(X, im_name, type, flag)
% X = cell array of trajectories { [x y] [x y] ... }
% im_name = name of underlying image
% type = type of plot_all_trajectories
% flag = 1 if plot
if flag == 1
    % read in image
    A = imread(im_name);
    % Determine Image Size
    [y_end x_end dum] = size(A);
    % Plot Coordinates
    x_start = 0;
    y_start = 0;
    % Plot Image
    imagesc(A)
    axis([x_start x_end y_start y_end])
end
hold on
% plots all
num_tr_feat = length(X);
for ii = 1:num_tr_feat
    switch flag
    case 0
        plot(X{ii}(:,1),X{ii}(:,2),type)
    case 1
        plot(X{ii}(:,1),X{ii}(:,2),type)
        axis([x_start x_end y_start y_end])
    case 2
        plot(X{ii}(:,1),X{ii}(:,2),type,'Color',rand(3,1),...
'MarkerSize', 18)
end
end
hold off
Bibliography


