PIR SENSING ARRAY FOR FALL DETECTION

A thesis presented to

the Faculty of the Graduate School

at the University of Missouri – Columbia

In Partial Fulfillment

of the Requirements for the Degree

Master of Science

by

MICHAEL J. MOORE

Dr. Marjorie Skubic, Thesis Supervisor

MAY 2011

The undersigned, appointed by the Dean of the Graduate School, have examined the thesis entitled:

PIR SENSING ARRAY FOR FALL DETECTION

Presented by Michael J. Moore,

a candidate for the degree of Master of Science,

and hereby certify that, in their opinion, it is worthy of acceptance.

Dr. Marjorie Skubic, Ph.D.

Dr. James Keller, Ph.D.

Dr. Mihail Popescu, Ph.D.

Acknowledgements

I would like to thank my wife for all of her support throughout my career as a student. I look forward to the rest of our lives together.

I would also like to thank my parents, Steve and Suzie Moore for their constant encouragement. Their confidence in me as made me believe that I can do anything I want to do.

I thank my advisor Dr. Marjorie Skubic, for providing me with a topic of research that fits my interests. Also, I would like to thank her for allowing me to work with such a great, diverse group of researchers at the Center for Eldercare and Rehabilitation Technology. Working with her and the other members of the group has shown me how important it is to love what you do.

I would also like to thank Dr. James Keller and Dr. Mihail Popescu for providing their expert knowledge and guidance in introducing me to the world of computational intelligence.

I am also very grateful to my colleagues Tanvi Banerjee, David Heise, and Erik Stone for their encouragement and helpful suggestions throughout this research.

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

Abstract

The purpose of the Fuzzy PIR Fall Detection Array is to keep the elderly safe by providing a means for an immediate response to falls while still allowing them to enjoy the same independence they felt before fall detection was necessary. To accomplish this goal, a vertical array of passive infrared (PIR) motion sensors can be positioned anywhere in the home near where a fall may occur. A fall is considered to be observed by the sensor array when the sensors, first, detect motion, then, stop detecting motion in order from top to bottom. To differentiate between a legitimate fall and normal motion, pattern recognition techniques were used to observe the signals from the sensing array and classify whether a window of data was observed during a fall or a non-fall. To accomplish this goal, a Gaussian Parzen Window (GPW) and a relevance vector machine (RVM) were used with some success. This research shows that, for this application, the RVM is a superior classification method to the Parzen Window, where the RVM was able to detect falls with an accuracy of about 80% to the Parzen Window's about 75%. Besides being more accurate, the RVM algorithm has a faster run time for classifying the data. The sensing array explored in this research could be a viable option as a non-wearable means for protecting the elderly in the event that they should fall in their home.

Chapter 1 - Introduction

1.1 Problem Statement

No one wants to fall, as it could cause severe pain or maybe some long term discomfort. However, for the elderly there is much more at stake than pain and discomfort when even the slightest fall can be something that causes damage that they may never recover from [1]. As technology continues to evolve many new methods have become available to protect our older population such as wearable accelerometers that can detect a fall [2] and wearable pendants that allow the elderly person to call for help [3]. These new advancements might allow relatives to rest easy knowing that their family members are safer, but many of these safeguards leave their users feeling as though their freedom has been compromised, sometimes so much so that they may choose to leave these devices behind. In this paper, a method of detecting falls through the use of passive infrared (PIR) motion sensors and pattern recognition techniques is discussed.

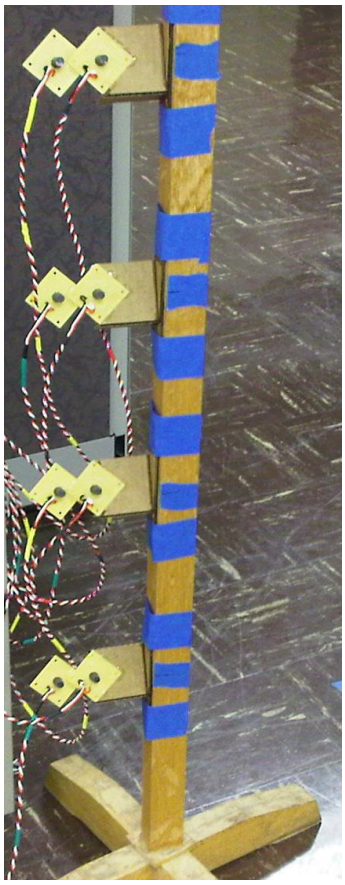

Figure 1.1: Vertical array of passive infrared (PIR) motion sensors

The overall goal of this research is to develop a vertical array of PIR motion sensors which is capable of detecting falls that may occur in the home of an elderly

person. To accomplish this, a fall sensor was configured to observe test falls performed by a stunt actress who attempted to re-create accidental falls by an elderly person. This is done using a vertical array of PIR motion sensors which is pictured in Figure 1.1. As discussed in this paper, these sensors were used to detect test falls by recording output from each of the sensors and then performing preprocessing on the data to aid the feature extraction process. Next, features were extracted to be used in the classification process. To determine whether a given sequence of motion is indeed a fall or not, two pattern recognition techniques were explored; the Parzen Window and the Relevance Vector Machine (RVM) were used to classify data as either a fall or a non-fall. Once the classification step is done, post processing can be carried out on the classified data to determine whether a fall has occurred. If a fall has occurred, the same sensor array can monitor a person's post-fall activity to validate whether a fall has actually occurred and, if it is necessary, to call for help.

1.2 Overview

In developing a sensor that was able to detect falls, but did not have to be worn by the person for whom it was made to protect, PIR motion sensors were used as the base sensing element. Eight of these sensors were arranged in 4 pairs oriented vertically with respect to each other and evenly spaced. From the eight sensors come 8 analog signals. Each of these signals was connected to an analog to digital converter (ADC) which interfaced to a laptop through its USB port. Using data acquisition software, the ADC was able to store the instantaneous values for each of the analog channels at a user defined frequency. The data from the sensors were stored in a text file which could then

be opened and read into MATLAB, a mathematical analysis software package which, throughout this work, is used to manipulate data and implement pattern recognition algorithms. Once the data was read into MATLAB, it then had to be preprocessed so that features could be extracted from it. After the data was preprocessed, the slope of the data and the difference between the average output of each adjacent sensor pair were extracted as features. Next, the feature data from each fall run was separated between testing and training data. Next, the Parzen Window and RVM algorithms were used to determine if a fall had occurred in each of the data runs. Each of the pattern recognition algorithms was programmed to output a confidence value that a fall had occurred. From this confidence output, a receiver operating characteristic (ROC) curve was used to determine a good threshold to be used in the post-processing phase where the output must be above this threshold value to be considered a fall.

This thesis begins by covering the background and related work where the details of fall detection, passive infrared motion sensing, the Parzen Window classifier, and the relevance vector machine are discussed in detail. Next, in chapter 3, the methodology for carrying out this research is described. The development of the sensing array is laid out first. Then, the data acquisition techniques used to collect motion data are detailed; a stunt actress performed falls according to the fall protocol in Appendix A. After that, in chapter 3, the preprocessing of the incoming data is discussed. Next, this thesis talks about the features that were extracted from the preprocessed signals. Then, the methods used to implement the Parzen Window and RVM are described. Finally, to close out chapter 3, the post-processing techniques that were used in this research are

examined. Chapter 4 goes over the experimental results and analysis of this research. In chapter 5 research results are discussed. Lastly, chapter 6 concludes this thesis.

Chapter 2 - Background and Related Work

2.1 Fall Detection

There are many different ways to go about detecting falls that occur in an elderly person's home, whether it is in a nursing home or an assisted living community. Two of the major approaches to fall detection are fall detection using wearable sensors and fall detection using non-wearable sensors. In the case of wearable sensors, a resident must wear a device that will either automatically detect a fall and call for help, or relies on the patient to press a button when they feel help is needed. Alternatively, in the non-wearable case, sensors are placed throughout the home of an elderly person to collect and analyze data which can then be used to automatically detect falls and call for help.

Typically, wearable techniques which are capable of automatically detecting falls use some type of multi-axis motion sensor, such as an accelerometer or a gyroscope, to monitor an elderly resident's movements. The data that are collected from the motion of an elderly resident are then compared to a threshold value to determine if a fall has occurred [2] [4]. These techniques can be very effective at recognizing falls; however, they rely on the resident to be willing to wear the device. Another wearable technique is for an elderly resident to wear a pendant that is capable of alerting caretakers that a person needs help if the elderly resident presses a button [3]. This technique relies on the elderly resident to be coherent enough to press the button, and to voluntarily summon help. Furthermore, these wearable techniques do not protect a person when they are performing activities without clothing such as bathing or changing their clothes.

Alternatively, non-wearable techniques do not rely on a person to be willing or able to call for help, as they are capable of automatically calling for help, and are less likely to decrease a person's feeling of independence by having to be worn on the body. Non-wearable techniques utilize non-contact sensors such as cameras [5][6], microphones [7], and infrared sensors [8]. Cameras can be used in many different ways to detect falls. A camera system for fall detection may consist of only 1 or more cameras, and in the case of multiple cameras, can be used to create a 3 dimensional voxel space representation of a person. A camera system for fall detection can provide caretakers with a greater amount of information. In addition to fall data, a camera based system can be used to gather clinical data. Examples of this include sit-to-stand times to assess mobility [9] or foot fall information that can be used for gait analysis [10]. All of the information available from cameras can be a great advantage in protecting an elderly resident, but it can be invasive if the person feels that his privacy is compromised due to cameras being installed in the home.

This research explores the use of PIR motion sensors as a means for fall detection. The PIR array, discussed throughout this research, can detect falls without being worn, and since it is not collecting video information about a person, they may feel more comfortable having it monitor them in a bedroom or bathroom setting.

2.2 Passive Infrared (PIR) Motion Sensing

In this work, PIR motion sensors are used as the basic sensing elements for human motion detection. Objects that give off heat reflect electromagnetic radiation in

the infrared spectrum between 0.7 micrometers and 300 micrometers. The infrared light that is reflected from the human body has a wavelength of about 10 micrometers [11]. The sensors that were used in this research have an infrared filter which allows only infrared light with a 10 micrometer wavelength [12]. This helps to reduce the effect that non-targeted sources of infrared radiation (other than humans) have on the sensing elements. The enclosure, which houses the sensing element, also has built-in Fresnel

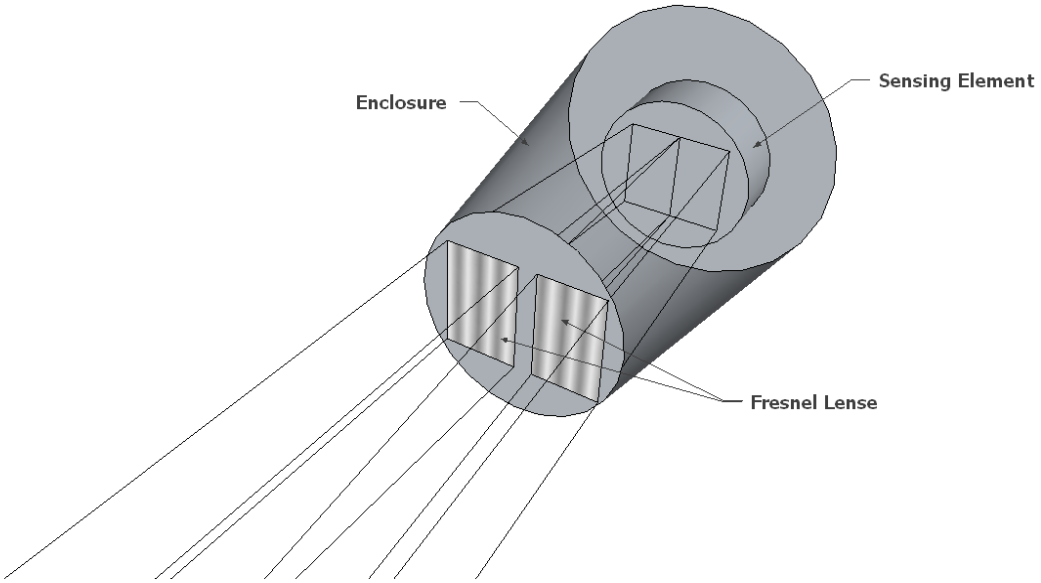

Figure 2.1: Simple PIR Sensor

lenses which direct the incoming infrared light to four sensing areas on the surface of the internal sensing element. Typically, PIR sensors have two or more distinct surfaces on the face of the sensing element, which develop a charge proportional to the amount of incoming infrared light. In this research, the sensing element uses 6 Fresnel lenses to focus light onto a sensing element with four individual sensing surfaces. However, in this section, for simplicity, a sensor with 2 Fresnel lenses which focus light onto a sensing element with 2 sensing surfaces, such as the one shown in Figure 2.1, is described. The

way the simplified PIR sensor works is as follows. First, infrared light is reflected off the surface of an object according to its emissivity. Once the reflected infrared radiation reaches the enclosure of the sensor, it is focused by each of the Fresnel lenses towards

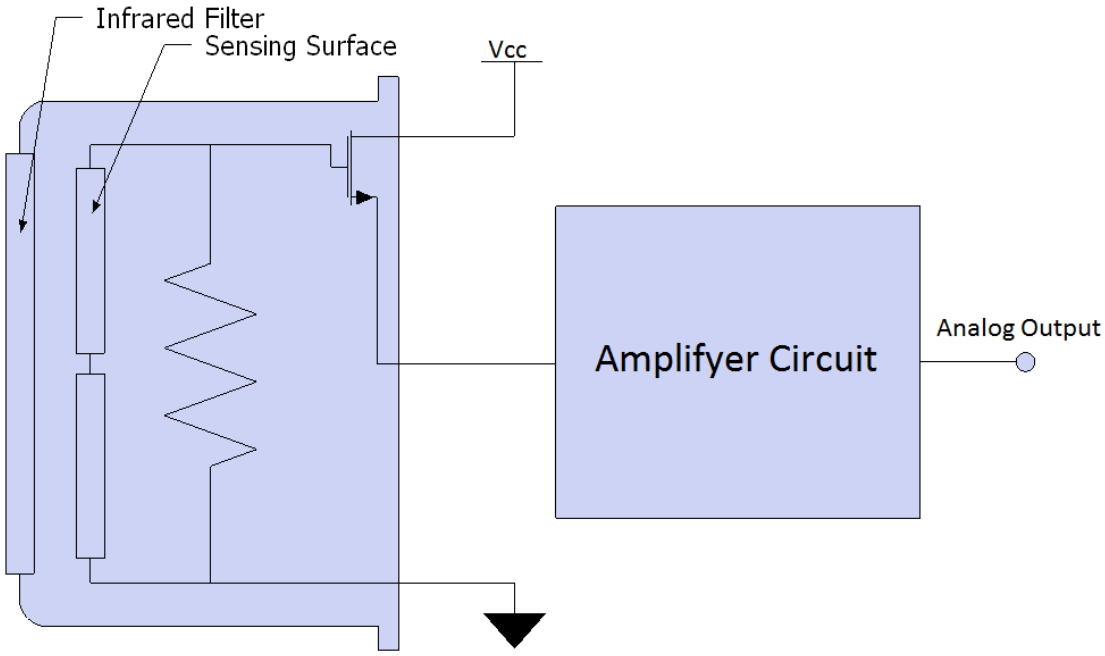

Figure 2.2: Sensing Element Internal Circuit

the surface of the sensing element where the Fresnel lenses direct light towards a specific portion of the sensing surface of the sensing element. Once the light is focused on the sensing surfaces, the light is then filtered by the infrared filter to only allow light with a wavelength of about 10 micrometers to pass through. Once the light reaches the surface of the sensing element, a surface charge proportional to the amount of infrared energy directed, develops on the sensing surface. The sensing surfaces within the

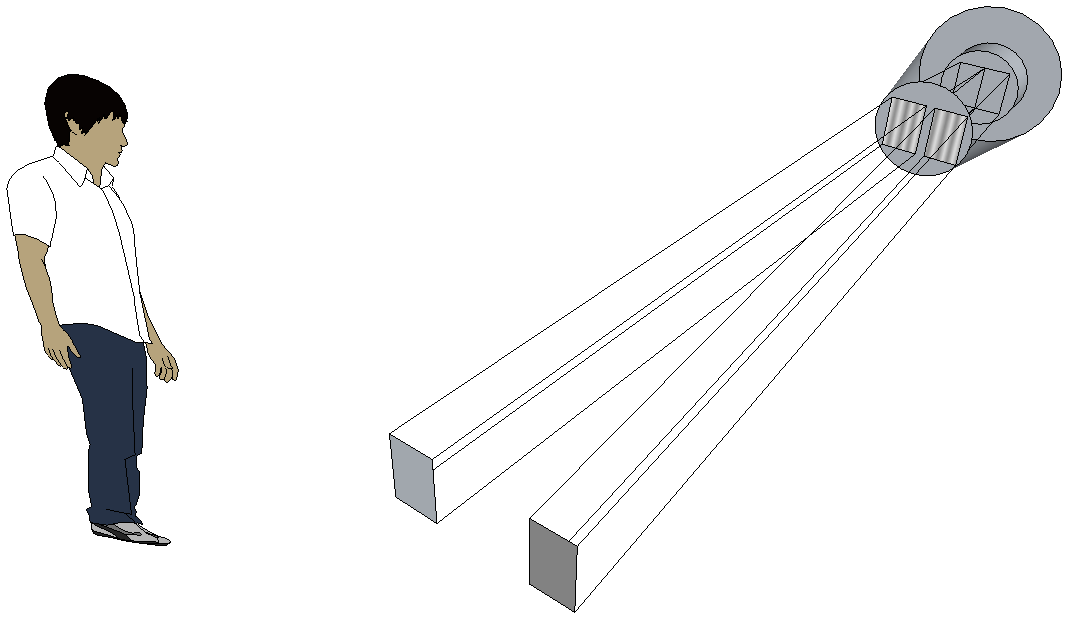

Figure 2.3: Sensor field of view (FOV) with person

sensing element are part of a circuit in which the sensing surfaces are connected in parallel to a resistor and then connected to a MOSFET at one end and grounded on the other end [13]. This parallel configuration ensures that the sensor only generates electrical pulses when there is a potential difference between the two sensing surfaces. This helps to reduce the effect that an air conditioner coming on, or sunlight coming in the windows in the morning would have on the sensor, because these types of events affect both sensing surfaces and would effectively cancel. As can be seen in the figure above, when a human walks through the field of view of the sensing element, the infrared energy which is reflected from their body activates, first the left sensing surface and then the right. An example of the signal that the sensor would generate is shown in Figure 2.4. In this research, the output signals from one sensor are analyzed with the output signals from seven other sensors and are processed to estimate the behavior of

an individual that is moving within the field of view of the sensor array.

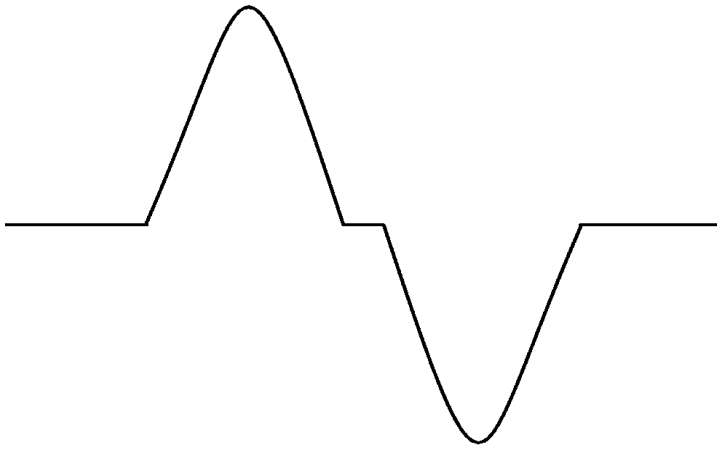

Figure 2.4: Example sensor output signal

2.3 Parzen Window

In this research, a Parzen Window classifier was used to indicate whether fall information belongs to either of two classes, fall and non-fall. Due to its proven performance and clear theoretical background [14] the Parzen Window is often chosen as the basis for machine learning and pattern recognition research. The Parzen Window is a particular method of non-parametric kernel density estimation where densities are estimated using a fixed volume about individual data points and few assumptions are made of the system from which the data was gathered [15] [16].

The overall goal of kernel density estimation is to use a probability density $p(\boldsymbol{x})$ which exists in a d dimensional hyperspace as a base kernel which is used to classify n sample data points. In the Parzen Window approach to kernel density estimation, the kernel function is evaluated for a volume V which is centered about each data point of

training data. To make explanation more straightforward, V is assumed to be a d dimensional hypercube. The volume equation for each sample is shown below in equation 2-1 where h is one side of the hypercube.

$$V_n = h_n^d \tag{2-1}$$

The probability density is then estimated in equation 2-2 where k is the number of samples which fall within the bounds of the hypercube.

$$p(\boldsymbol{x}) \cong \frac{K}{N * V} \tag{2-2}$$

When the volume is a d dimensional hypercube, the window function can be defined by equation 2-3 from [16].

$$\varphi(\boldsymbol{u}) = \begin{cases} 1 & |u_j| \leq 1/2 \\ 0 & otherwise. \end{cases} \quad j = 1, \ldots, d \tag{2-3}$$

From this, given a d dimensional test point and a d dimensional training point, it can be determined that $\varphi(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h_n})$ is equal to 1 when a test sample $\boldsymbol{x}_i$ is within the volume of a hypercube which is centered at training point $\boldsymbol{x}$ and 0 when the test point is out of the bounds of the volume defined by the hypercube. From this, the number of samples which fall within the hypercube can be calculated using equation 2-4.

$$k = \sum_{i=1}^{N} \varphi\left(\frac{\boldsymbol{x} - \boldsymbol{x}_i}{h}\right) \tag{2-4}$$

Substituting equation 2-4 back into equation 2-2 yields equation 2-5 shown below.

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{V}\varphi\left(\frac{\boldsymbol{x}-\boldsymbol{x}_i}{h}\right) \qquad 2\text{-}5$$

Equation 2-5 would be the kernel density function or Parzen Window for a hypercube with sides of length h. The results of using the hypercube create false discontinuities at the boundaries of the hypercube and thus a more smooth window function can be used to provide smoother results [15]. The Gaussian function, in theory, never goes to zero and thus will provide the more desirably smooth results. Replacing the hypercube with a Gaussian function produces the resulting kernel function shown below in equation 2-6.

$$p(\boldsymbol{x}) = \frac{1}{N}\sum_{i=1}^{N}\frac{1}{(2\pi h^2)^{d/2}}exp\left(\frac{\|\boldsymbol{x}-\boldsymbol{x}_i\|^2}{2h^2}\right) \qquad 2\text{-}6$$

In equation 2-6, h represents the standard deviation of the Gaussian function. Using the Gaussian distribution as the window function effectively centers a Gaussian distribution function over a training sample where results can be derived by calculating the cumulative distribution function from each sample and averaging the results from each calculation. Calculating the average effectively normalizes the results.

After the Parzen Window kernel that will be used is determined, the magnitude of the h parameter must be tuned. Finding the best magnitude of the h parameter is

dependent on the number of testing samples used. When the magnitude of h is very large, the results of the Parzen Window calculation will be more smooth and provide results with lower accuracy. When the h parameter is very small the results of the Parzen Window calculation will be more rough and there will be too much statistical variability [16]. In this research, the h parameter was chosen by classifying data and using the numbers reported by a confusion matrix to choose the best value for the magnitude of the h parameter.

Once the Parzen Window kernel is established, this method can be used to classify data. For this research, a two class classifier was used where data was classified as belonging in either class 1 or class 2. To classify the data, 50 sample windows were classified where each testing point in the window was calculated in the Gaussian function with each training point from each of the classes. Once that was complete, the results from each class, compared to each testing sample, were averaged. Next, the results from averaging the values from each class were compared against one another and each window of data was considered to belong with the class which has the highest average value. This process was repeated until each window of testing points was classified.

The Parzen Window classification process requires little upfront investment as it requires virtually no training. To train the classifier, all that is required is to organize data into training and testing sets. A low amount of required training can be convenient, but for the purposes of this research, there is no advantage in choosing a classifier that

requires a shorter training period. This is because, for processing time, all that matters is how fast it can classify training data. Also, because the Parzen Window requires calculations to be made between each testing sample and each training sample, the computational complexity for running the Parzen Window classifier is very high and requires a large amount of memory for data storage [14]. Thus it would require a high performance computer to run the classification algorithm in real time, whereas more sparse classification algorithms could be optimized to classify incoming fall data more quickly with less computational power.

2.4 Relevance Vector Machine (RVM)

After demonstrating the feasibility of this research and building a working software package with the Parzen Window, an algorithm that can be used for classification, which requires less computational complexity and storage space, was needed. For this, the Relevance Vector Machine (RVM) was chosen. The RVM is based on Bayesian statistics where the goal is to constrain the classification parameters by placing priors over them and later integrating out or marginalizing these parameters to obtain the predictive distribution that allows classification of new data. The RVM takes input samples and pairs them with class labels (-1, 1) to model the probability distribution using logistic regression [17]. One important attribute of the RVM algorithm is that training the regression model yields a sparse set of training data, which means that it uses only the most relevant data samples. The number of training samples is determined through the training process. This process prunes training samples that are determined to be least effective when separating data. This essentially reduces

computational complexity as well as lowering the amount of memory needed to store the relevance vectors. Another advantage is that it is a kernel based classification method that can be extended to non-linearly separable data.

The RVM is a regression algorithm that provides results in the form of probabilistic classification. Typically, the goal of regression algorithms is to use a training set consisting of input training vectors $\boldsymbol{x}$ to find a parameter vector $\boldsymbol{w}$ as well as a single offset c [18]. Those parameters are then used to predict an output y using a set of input data $\boldsymbol{X}$ that consists of unknown input vectors**.**

$$y = \boldsymbol{w}^{\boldsymbol{T}}\boldsymbol{x} + \mathrm{c} \tag{2-7}$$

In this research, it is assumed that the input data has a non-linear relationship and thus the input data $\boldsymbol{x}$ is transformed to a kernel space $\varphi(\boldsymbol{x})$ [19][15][18]**.**

$$y = \boldsymbol{w}^{\boldsymbol{T}}\varphi(\boldsymbol{x}) \tag{2-8}$$

To calculate the weight vector $\boldsymbol{w}$ during the training portion of the RVM algorithm, the target data $\boldsymbol{t}$ are assumed to represent the model y.

$$\boldsymbol{t} = y = \boldsymbol{w}^{\boldsymbol{T}}\varphi(\boldsymbol{x}) \tag{2-9}$$

The design matrix generated by providing $\boldsymbol{x}$ as an input parameter to the φ basis function yields the kernel $\boldsymbol{\Phi}$. The model of the target data given the input parameters is assumed to be dependent on mean zero Gaussian noise, thus the likelihood function can be represented in the form shown below [19][18].

$$p(\boldsymbol{t}|\boldsymbol{x},\boldsymbol{w},\sigma^2) = (2\pi\sigma^2)^{-N/2} exp\left\{\frac{1}{2\sigma^2}\|\boldsymbol{t}-\boldsymbol{\Phi w}\|^2\right\} \quad 2\text{-}10$$

After that, a prior probability distribution is introduced over the weight vectors $\boldsymbol{w}$. This is done by defining the prior as a zero mean Gaussian prior for each w. For this Gaussian prior, the variance is a precision hyper parameter defined as the inverse of the variance $\sigma = \alpha^{-1}$. So there is an α parameter for each weight which controls the effect of the prior [15].

$$p(\boldsymbol{w}|\alpha) = \prod_{i=1}^{N} N(w_i|0,{\alpha_i}^{-1}) \quad 2\text{-}11$$

In training the RVM, the goal is to find the parameters w, α, and σ^2 that will maximize the posterior probability distribution which is shown below [18] where σ^2 is replaced by β for simplicity.

$$P(\boldsymbol{w},\boldsymbol{\alpha},\sigma^2|\boldsymbol{t}) = P(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha},\beta)P(\boldsymbol{\alpha},\beta|\boldsymbol{t}) \quad 2\text{-}12$$

The first term in the posterior is once again represented as a Gaussian.

$$P(\boldsymbol{w}|\boldsymbol{t},\boldsymbol{\alpha},\beta) = N(\boldsymbol{m},\boldsymbol{\Sigma}) \quad 2\text{-}13$$

According to [19], the equations for the mean $\boldsymbol{m}$ and covariance $\boldsymbol{\Sigma}$ are shown below.

$$\boldsymbol{m} = \beta\boldsymbol{\Sigma}\boldsymbol{\Phi}^T\boldsymbol{t} \quad 2\text{-}14$$

$$\boldsymbol{\Sigma} = (\boldsymbol{A}+\beta\boldsymbol{\Phi}^T\boldsymbol{\Phi})^{-1} \quad 2\text{-}15$$

The variable $\boldsymbol{A}$ is the diagonal vector of $\boldsymbol{\alpha}$. To find the values for $\boldsymbol{m}$ and $\boldsymbol{\Sigma}$, equations for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ must be formed. Solving for the log marginal likelihood for the second part of equation 2-12 and using the evidence approximation procedure yields the equations below for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ [18].

$$\alpha_i = \frac{1 - a_i \Sigma_{ii}}{m_i^2} \tag{2-16}$$

$$\beta = \frac{N - \Sigma_i \gamma_i}{\|\boldsymbol{t} - \boldsymbol{\Phi m}\|^2} \tag{2-17}$$

Values for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ that will maximize the marginal likelihood and thus can be used to calculate target values t for new testing data are found through an iterative process where, first, initial values are chosen for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. After that, equation 2-14 and equation 2-15 are used along with $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$ to calculate values for $\boldsymbol{m}$ and $\boldsymbol{\Sigma}$ which can then be used in equation 2-17 and 2-16 to calculate new values for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$. This process is continued until a convergence criterion is met. A typical convergence criteria could be to choose a very small number which will serve as a minimum value for the difference $\alpha_{old} - \alpha_{new}$. The RVM becomes more sparse during the iterative training process as weights are pruned from the design matrix as their corresponding value for α grows towards infinity. This means that a maximum value for α should be initialized before the iterative training process is started where a weight value will be pruned if its corresponding α is above the threshold. After the convergence criteria is met, the weights, which are not equal to zero and are left after pruning, are considered to be relevance vectors. The iterative training process is outlined below as can be seen in [18].

1. Initialize a kernel function that is appropriate for the data samples and use the kernel function to create a design matrix $\boldsymbol{\Phi}$.
2. Initialize an appropriate minimum value for the convergence criteria. For example, the training process will stop when $(\alpha_{old} - \alpha_{new}) < convergenceThreshold$.
3. Initialize an appropriate maximum value where weights will be pruned if their corresponding α values grow to be larger than this threshold.
4. Initialize values for $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$
5. Calculate $\boldsymbol{m}$ and $\boldsymbol{\Sigma}$
6. Use $\boldsymbol{m}$ and $\boldsymbol{\Sigma}$ to update $\boldsymbol{\alpha}$ and $\boldsymbol{\beta}$
7. Remove relevance vectors whose corresponding $\boldsymbol{\alpha}$ values are above the maximum threshold set for $\boldsymbol{\alpha}$ values.
8. Continue with step 5 while $(\alpha_{old} - \alpha_{new}) < convergenceThreshold$.

Once the training process is complete, the set of relevance vectors or weights $\boldsymbol{w}$ can be used along with the corresponding training sample points $\boldsymbol{x}$ to find target values for a set of testing samples $\boldsymbol{X}$. The testing process begins by calculating a design matrix where some kernel function takes the training data and testing data as input and the result is calculated as part of the design matrix $\Phi(\boldsymbol{X}, \boldsymbol{x}^T)$. After that, the weights are used to calculate the estimated target values as can be seen in the equation below.

$$\boldsymbol{t} = \Phi(\boldsymbol{X}, \boldsymbol{x}^T)\boldsymbol{w} \tag{2-18}$$

Although the training process for the RVM can be time consuming, and requires a great deal of computing power and storage space, most of the time consuming computation is accomplished during this portion of executing the RVM algorithm. For the purposes of this research, a long training phase is not much of a disadvantage because it yields a more sparse training data set which will improve the time required to estimate the target values for new testing data. Theoretically, the RVM appears to be a good fit for this research because it provides probabilistic classification results which are a probability that data falls under a particular class which allows for post processing.

Chapter 3 - Methodology

3.1 Sensor Array

Because the overall goal of this research is to develop a vertical array of PIR motion sensors, which is capable of detecting falls that may occur in the home of an elderly person, the appropriate PIR motion sensor had to be selected to serve this purpose. After searching through the available options for PIR motion sensors, it was apparent that most PIR sensors provided either an analog output or a digital output. As discussed in section 2.2, most PIR sensors have a base sensing element which produces an output signal that oscillates when the sensing element is detecting motion. In the

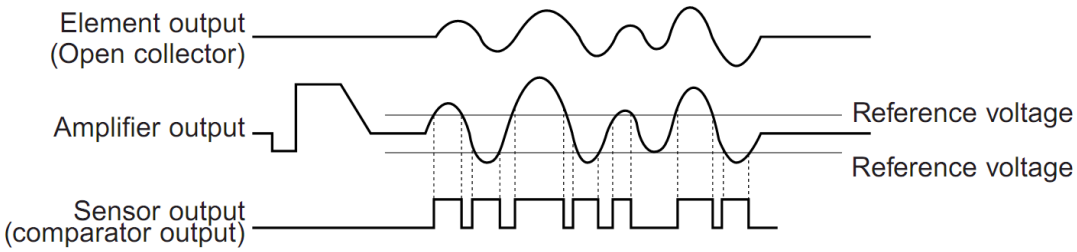

Figure 3.1: PIR motion sensor raw signals

case of the digital output PIR sensor, the element output signal is first sent through an amplifier circuit. Then the output from the amplifier circuit is sent through a comparator circuit that will output a square wave signal which is high when the raw element output signal is above a threshold voltage and low when the element output signal is below this threshold. An example of this can be seen in Figure 3.1, where the example digital output signal is labeled “comparator output” in the figure. Alternatively, in the case of an analog output sensor, the raw element output signal is only sent through the

amplifier circuit. The analog output PIR sensor was chosen for this research because the intention of this research was to use pattern recognition techniques to classify the signals coming from the sensors as either having been generated during a fall or generated during a non fall. It was decided that this could best be achieved using the amplified version of the analog output from the raw element output signal because the analog version of the signal retains more of the motion information than does the digital version. Further discussion of what is done with the analog output signals from the PIR sensors will continue in section 3.3.

As discussed in section 2.2, most PIR sensors come attached to an array of Fresnel lenses which define the sensor's field of view (FOV). The FOV of a particular sensor can be as wide as 180° horizontal and 90° vertical. However, because the sensors were to be set up in an array configuration, if the sensors had too large of a FOV, the individual FOV of the sensors would be more difficult to separate because of their overlapping FOV and would make it more difficult to distinguish between motion in one sensor's FOV vs. another sensor's FOV. For this research, a sensor with a 20° vertical FOV and 40° horizontal FOV was chosen.

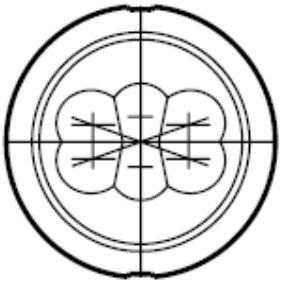

Figure 3.3: Fresnel lens array configuration

The sensor that was ultimately chosen was the Panasonic MP PIR motion sensor which can be seen in Figure 3.2. In addition to showing the overall FOV for the PIR sensors, Figure 3.4 shows the different detection zones within the FOV of the sensors. As a person walks throughout the FOV of the sensors, they pass through the different detection zones. As discussed in Section 2.2, when the person crosses the different detection zones it causes the output signal coming from the sensing element to oscillate.

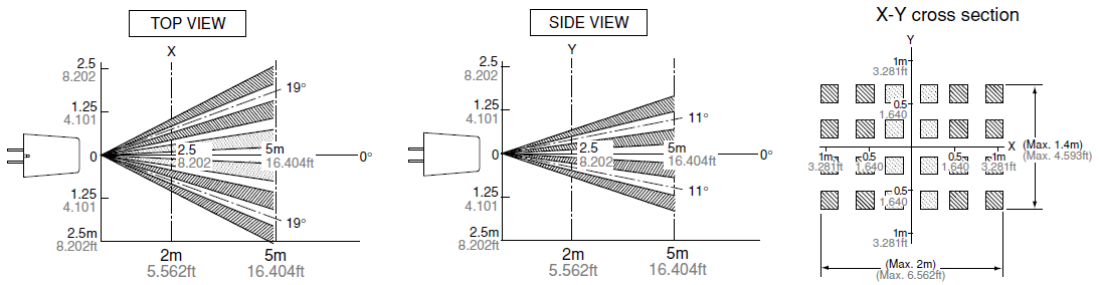

Figure 3.4: Sensor Field of view cross section

The Panasonic MP PIR Motion sensor is configured with an array of 6 fresnel lenses configured in 2 rows and 3 columns as can be seen in Figure 3.3. Initially the PIR sensors were attached to a vertical post and oriented in a horizontal position as can be seen in Figure 3.5. While in this configuration, although the sensors would provide signals where a fall could be visually detected, the signals did not provide consistent results as test falls occurred throughout the FOV of the sensing array. To fix this issue, the sensors were tilted upward 10° to position the lower limit of each of the sensor's FOV horizontally as can be seen in Figure 3.6.

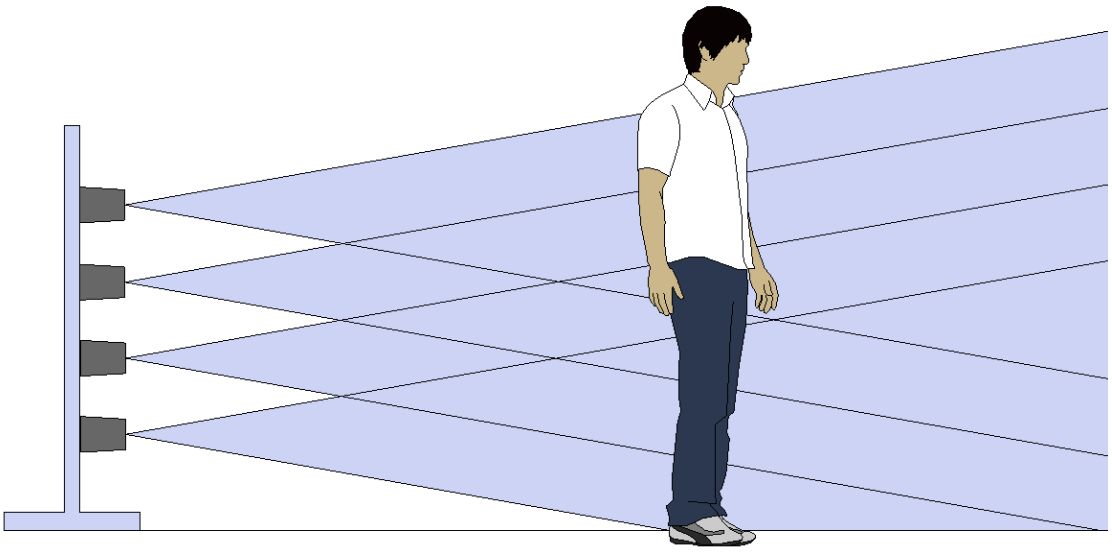

Figure 3.5: Sensor array horizontal FOV

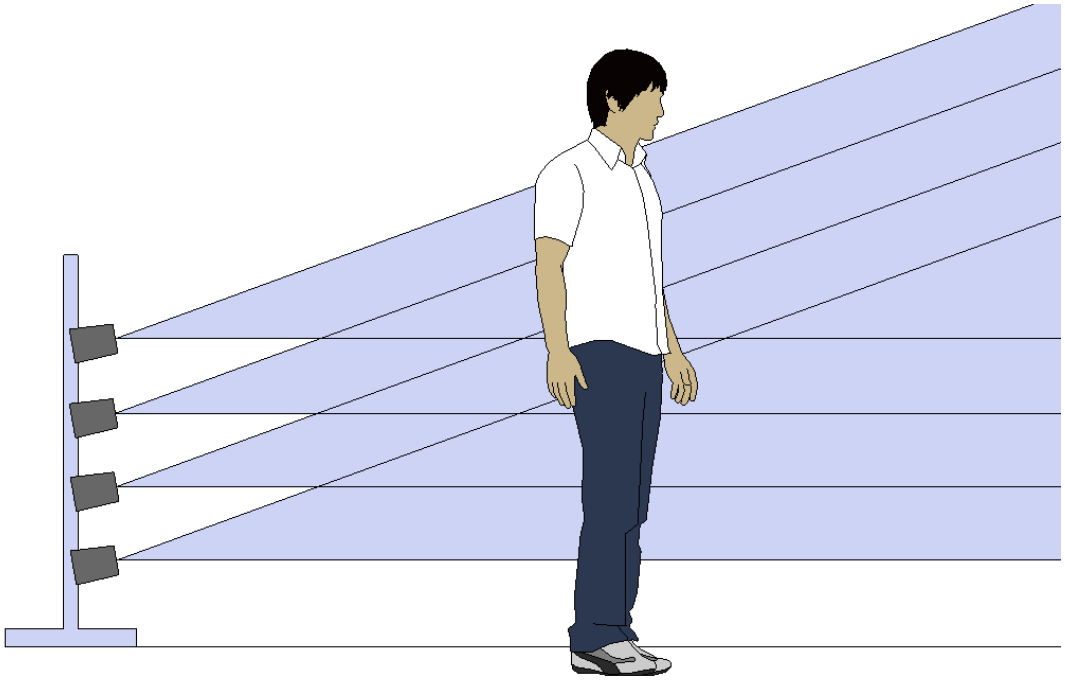

Figure 3.6: Sensor array 10° FOV

Once the sensors were situated in an array configuration and tilted such that the lower boundary of their FOV was horizontal, the signals acquired during a test fall were still not consistent as to provide reliable detection of falls. To solve this issue, the

sensors were then set up in pairs to provide some redundancy for the observed motion at each level of the sensing array, thus leaving the sensing array in its final physical configuration as can be seen in Figure 3.7. In this configuration, there are eight total

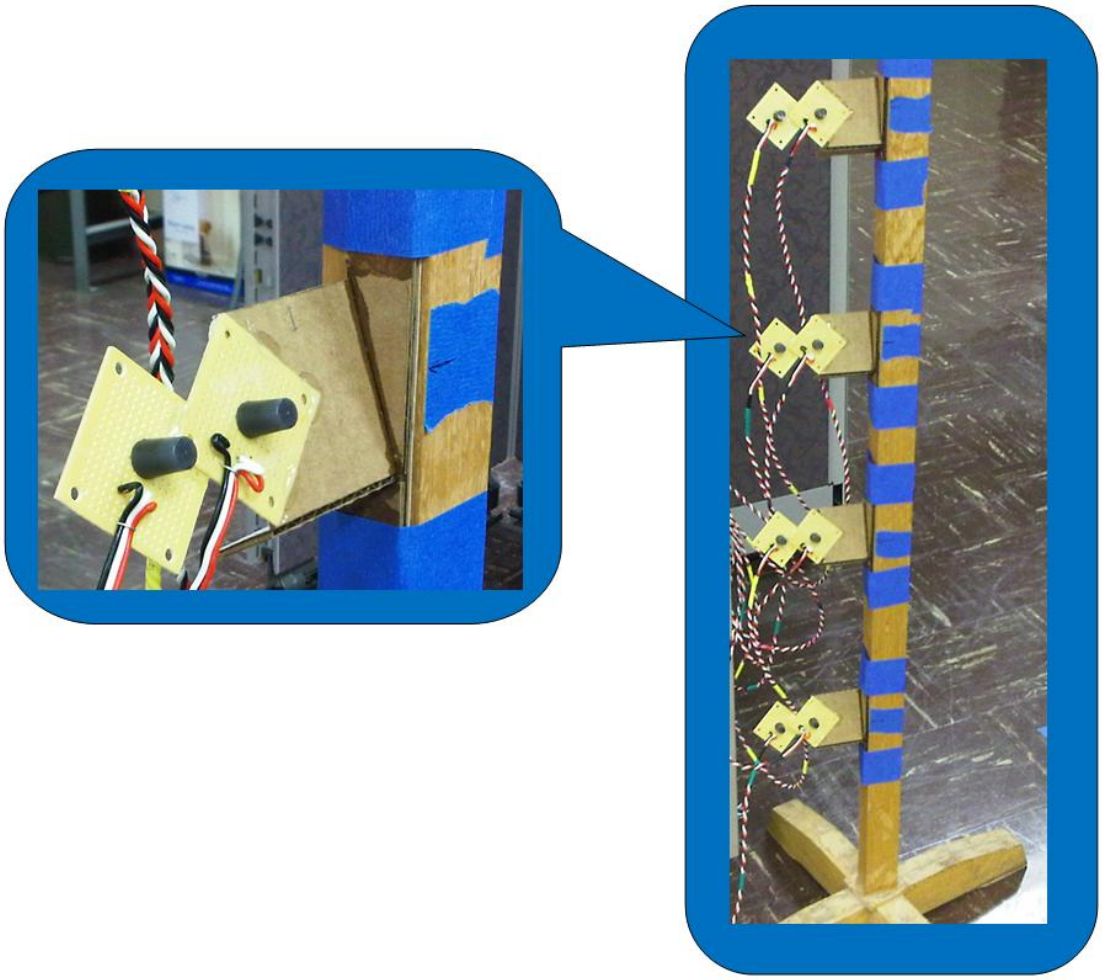

Figure 3.7: Final sensor array configuration

sensors which are paired, leaving four pairs of sensors. These four sensor pairs are then positioned in a vertical array with each sensor pair occupying its own horizontal plane. As the signals come from each of the sensor pairs, they are first preprocessed individually and then they are averaged together. A detailed discussion of signal preprocessing is continued in Section 3.3.

This configuration defines the behavior of the signals generated by the sensing array. If the human subject is moving through the FOV of the sensing array, all sensors will be active and their output signal will be oscillating. If this human subject falls, the output signal of the top most sensors will be the first to stop oscillating due to a lack of observed motion. Next, the output signals from the sensor pair, positioned just below the top most sensors, will stop oscillating. This will continue as the person descends towards the floor. In an ideal situation, when a person falls in front of the array, their falling motion will activate all sensors. Then, as the person falls, each of the sensors will stop detecting motion in sequential order from top to bottom, and their respective output signal will stop oscillating. For demonstration purposes, the preprocessed version of the signals generated from this process can be seen in Figure 3.8. The signals in Figure 3.8, first, indicate no motion where the signals are stable and have very small amplitude. Next, when the subject walks within the FOV of the sensing array, all of the sensor pairs begin to detect motion and thus increase in amplitude. After that, the amplitude from all four of the sensor pairs drops, as the sensors cease to detect motion, as the person has stopped walking and is standing in place. After that is the fall, where all sensor pairs begin to energize with the motion of the fall and soon after de-energize in order from top to bottom as the person descends towards the ground.

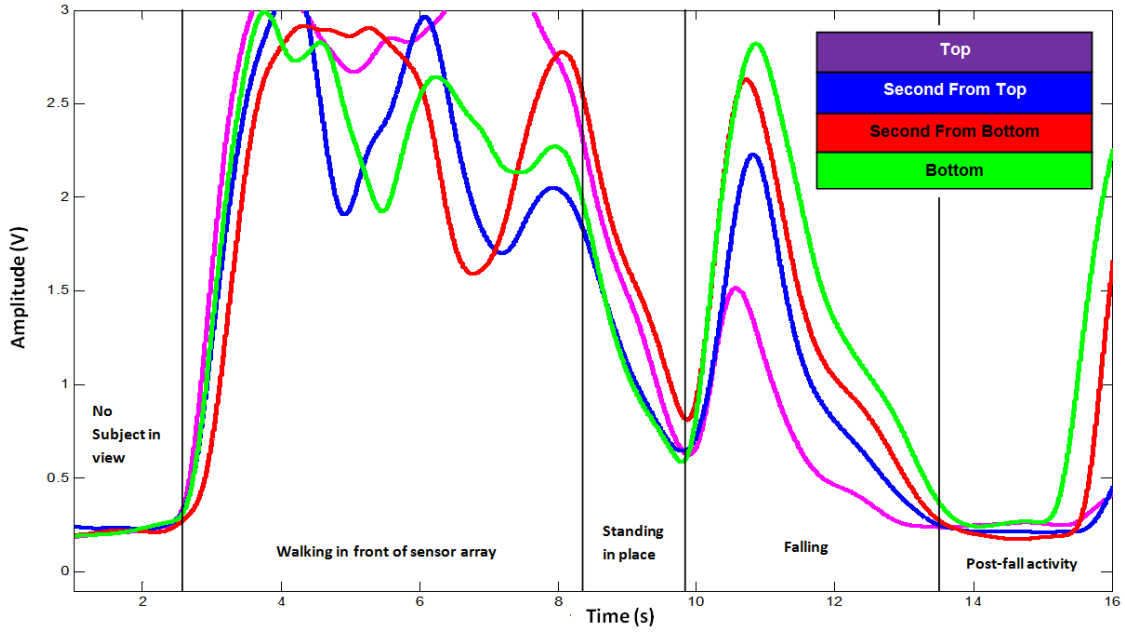

Figure 3.8: Preprocessed fall signals

3.2 Data Acquisition

For the signals coming from the sensing array to be analyzed, they first had to be captured and stored. To do this, an analog to digital converter (ADC) was used. The specific ADC setup, that was used, begins with a laptop which is set up to run National Instruments (NI) SignalExpress software. SignalExpress initiates and facilitates the collection and storage of data recordings. SignalExpress communicates with a NI USB-9162 carrier which is capable of accepting different types of NI data acquisition modules. In this research, a NI 9201 module was attached to the USB carrier. The 9201 is an eight channel analog input module. Thus, the ADC system used in this research was capable of recording the output from eight analog signals, one from each of the eight sensors. A diagram which shows the equipment that was used for capturing the signals in this research is shown in Figure 3.9.

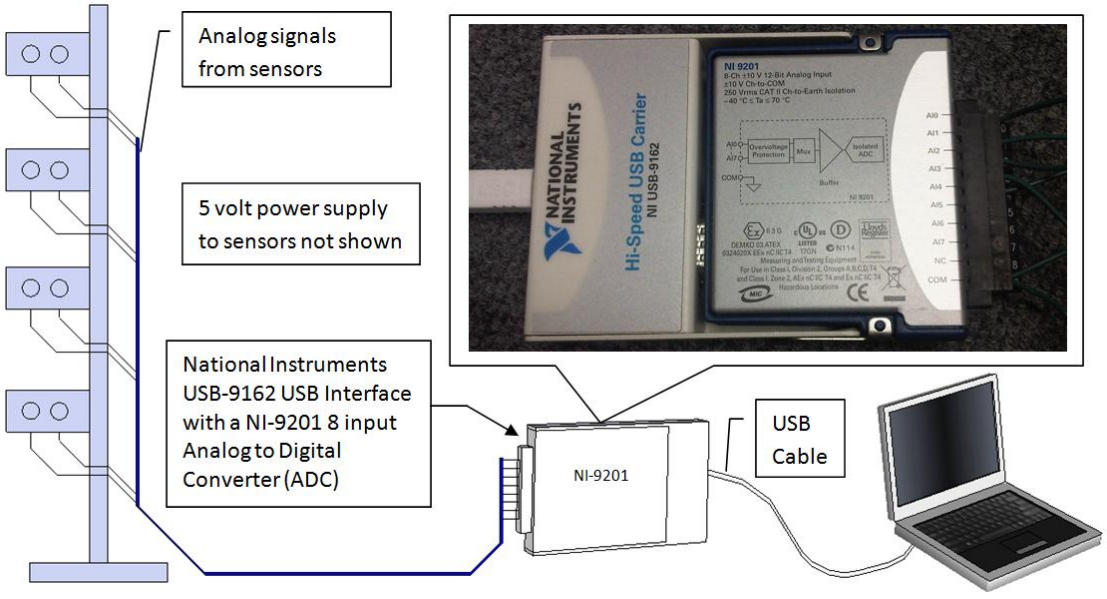

Figure 3.9: Analog to Digital Conversion (ADC) Setup

To test the frequency content of the PIR sensors, they were connected to an HP 3561A Signal Analyzer. Using the signal analyzer, it could be seen that when motion was being detected by a sensor, there was frequency content up to about 100 Hz. From this, it was assumed that oversampling by 10 times would be sufficient to recreate the signals after they are digitized. This led to the choice of a sampling frequency of 1,000 Hz for all data acquisition in this research.

Besides connecting the sensors to the ADC, they also had to be powered with a five volt power source. The power supply that was used was a nine volt switch mode DC power supply that was connected to a wall outlet at one end and the other end was

Table 3-1: PIR sensing array bill of materials

Item	Quantity	Description	Value	Distributor	Part Number
1	1	Electrolytic Capacitor	47µF	N/A	N/A
2	1	Electrolytic Capacitor	1µF	N/A	N/A
3	1	Ceramic Capacitor	.1µF	N/A	N/A
4	1	Linear Voltage Regulator	5V	National Semiconductor	LM1117T-5.0
5	8	PIR Motion Sensor	NA	Panasonic	AMN23111

connected to a breadboard which had a linear voltage regulator that converted the nine volts from the wall down to the five volts required for the PIR sensors. Also, there were three filter capacitors connected between power and ground to ensure that the power going to the sensors was free from transients. The bill of materials (BOM) for the sensing array can be seen in Table 3-1. Also, Figure 3.10 shows a schematic for the sensing array.

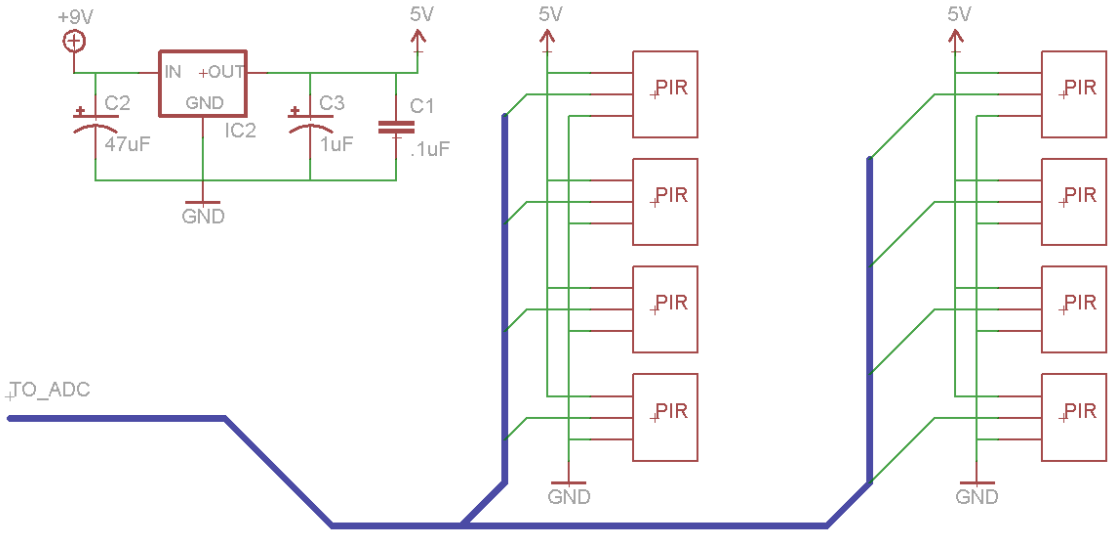

Figure 3.10: PIR array schematic drawing

For this research, data were collected in individual files pertaining to one fall or non-fall activity. For example, a fall to the left side from a walking position was associated with one data file. The data file collection process for this research goes as follows. First, the sensing array and data acquisition equipment must be properly set up and ready to collect data. This means the sensing array must have been powered on for at least 30 seconds to allow the sensing elements to stabilize as indicated in the datasheet [12]. Once the equipment is set up, someone must be ready to initiate the data recording on the computer attached to the ADC. There also must be a test participant who is prepared to perform an action that resembles a fall or non-fall. Once all persons and equipment are ready, the data recording is started, and the participant begins the fall or non-fall action. Once the action has been performed, the person doing the recording stops the data recording and saves the data file that was generated in an

appropriate directory on a file storage device. The file name for each data file is saved as a time stamp. For example, if a recording was taken on the year 2010, 20th day of April and a time of 10:51 AM, the file name for that data run would be 201004201051.txt.

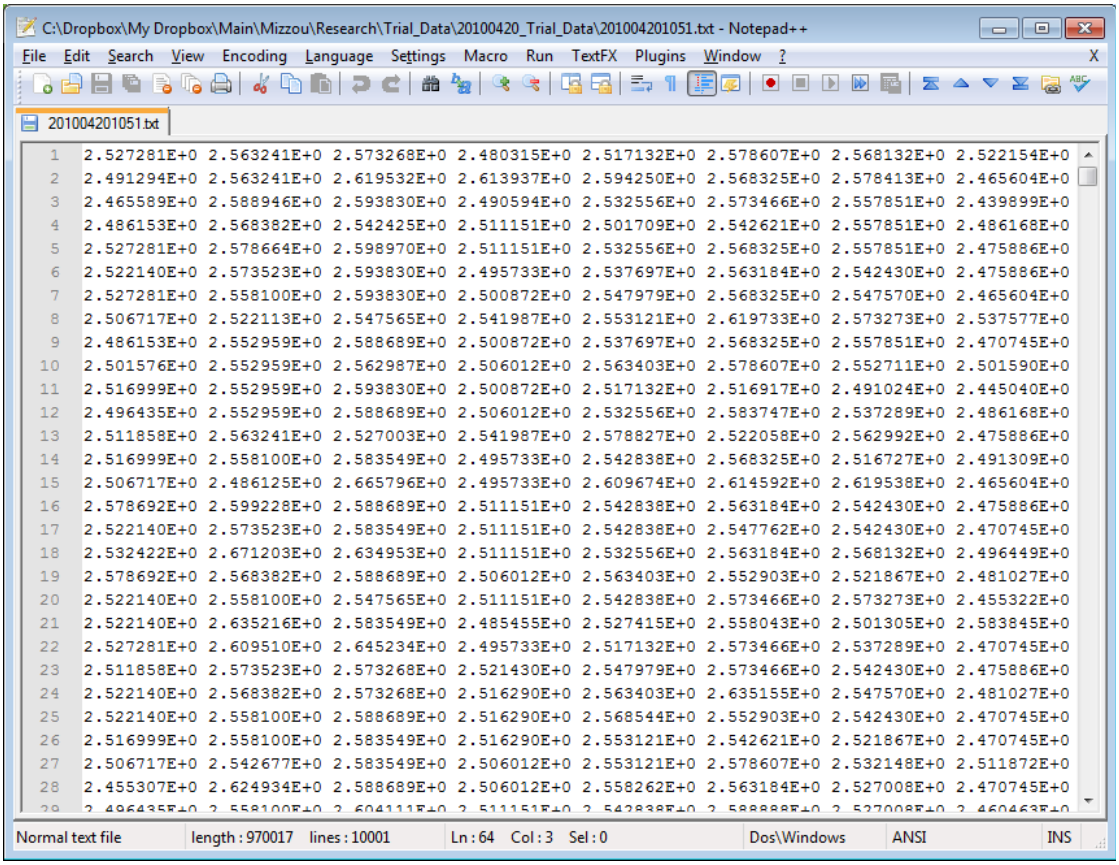

Figure 3.11: Typical ASCII text data file

The files generated by SignalExpress are ASCII formatted text files where each sensor is represented by a column vector. Thus, at each sample period, a value is recorded in the text file for each of the sensors, separated by a space character. After the data is recorded for a sample period, the string of data stored in the text file is terminated with a carriage return character. This format makes this data compatible

with both MATLAB and Excel. Figure 3.11 shows a typical text file where the data for a fall or non-fall action is stored.

3.3 Preprocessing

After data have been collected, the next step is to preprocess it by transforming it for the feature extraction process used in this research. As can be seen in Figure 3.12, the signals that come directly out of the PIR sensing array are not easy to interpret, as it is not immediately clear where a fall has occurred. As discussed in section 3.4, this raw

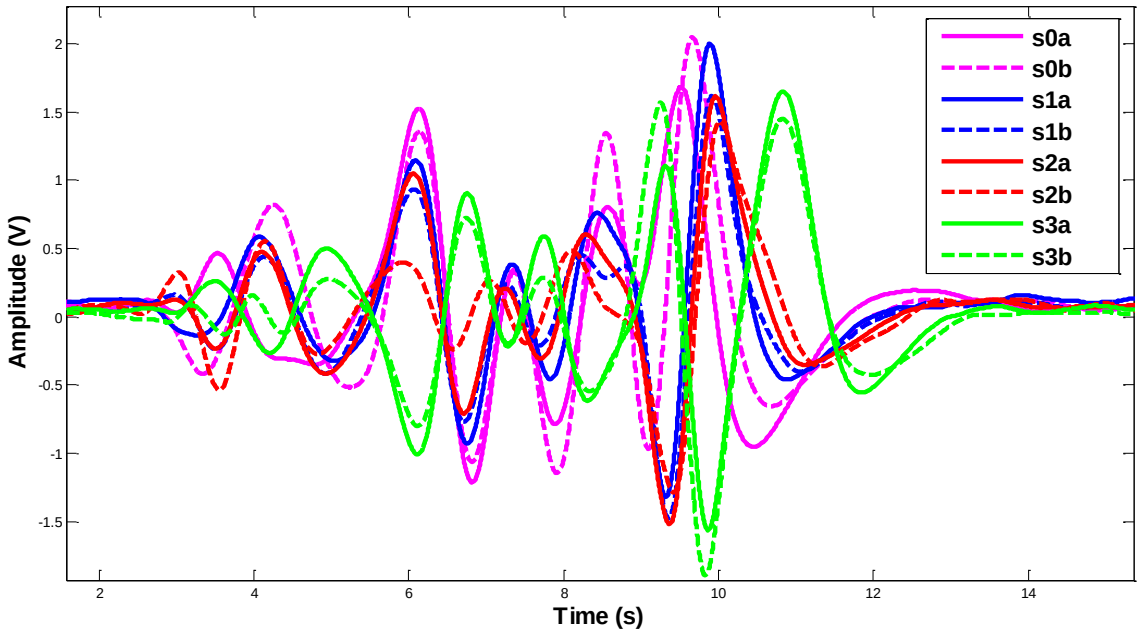

Figure 3.12: Raw data signals

form of the signals makes feature extraction difficult as well. Thus, the raw signals are preprocessed to reveal the features that were used in this research to detect falls. The data in this research were preprocessed according to the algorithm laid out in the flowchart and block diagram shown below in Figure 3.13 and Figure 3.14 respectively.

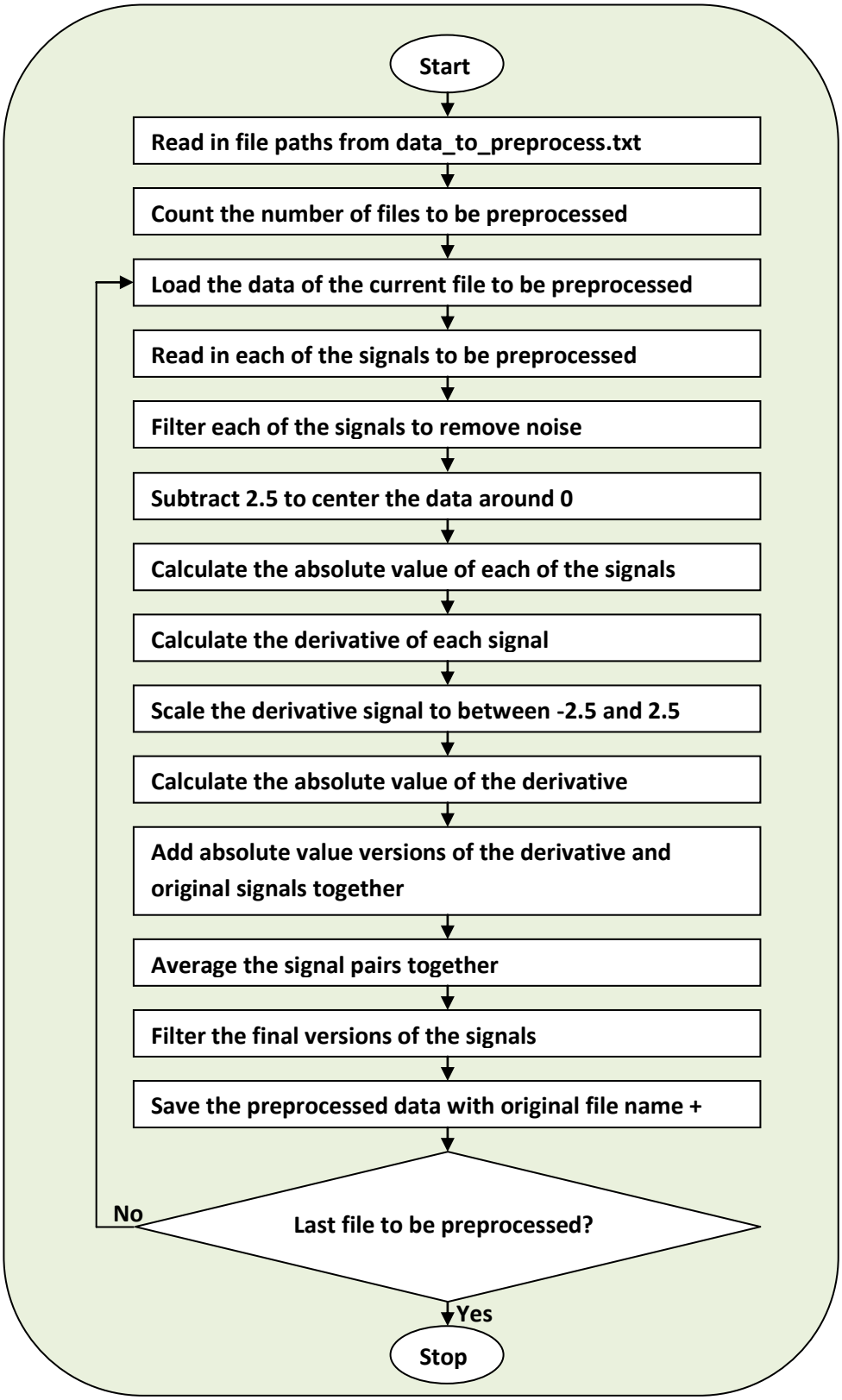

Figure 3.13: Preprocessing software flow chart

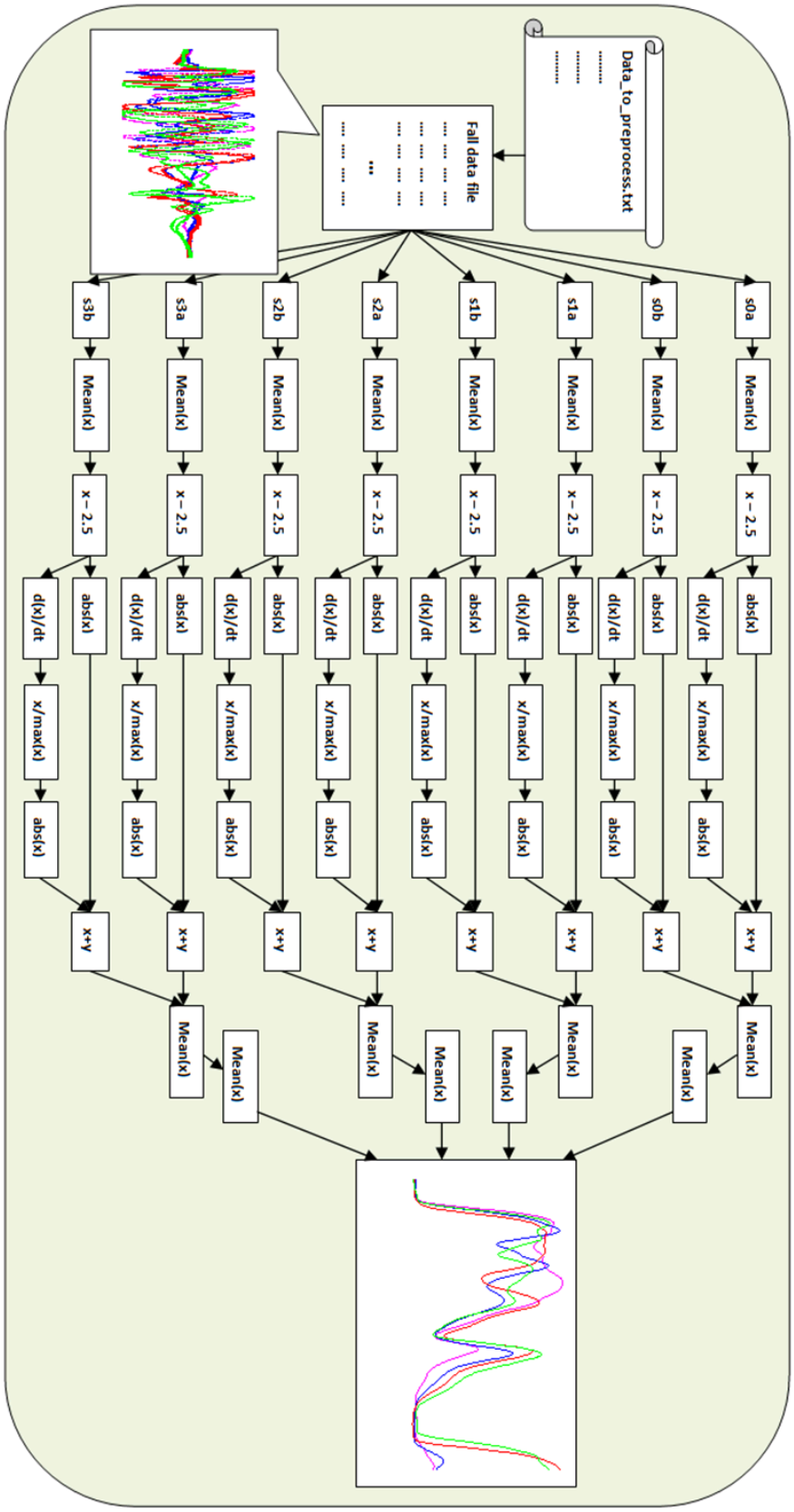

Figure 3.14: Preprocessing block diagram

The general goal of the preprocessing step is to transform the signals coming from the sensors from signals that oscillate when detecting motion to signals that have high amplitude when the sensor is detecting motion and low amplitude when the sensor is not detecting motion.

The preprocessing procedure performed in this research begins by loading the data that is to be preprocessed. After the fall or non-fall data are loaded, each of the eight signals are separated into individual variables. The first transformation operation that is performed on the data signals is to use a mean filter on each of the data signals. The purpose of the initial mean filter is to filter out the small amount of noise on each of the signals which was present when the signals were being saved by the data acquisition

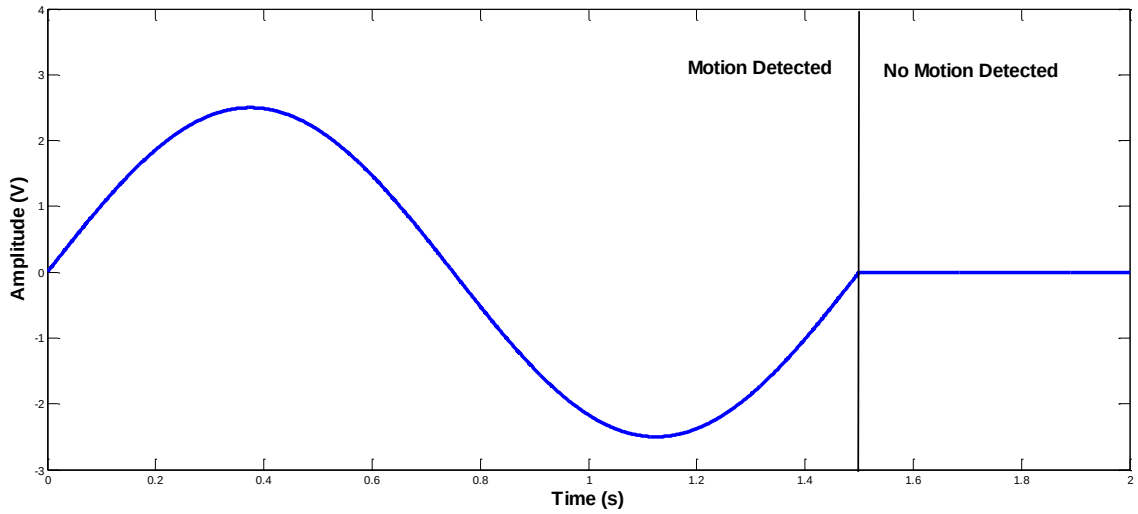

Figure 3.15: Zero centered example signal

step. Since the amplitude of the output signals coming from the sensors is between 0 and 5 volts, the steady state value for the signals coming from the sensors when they are not detecting motion is 2.5 volts giving the signals coming from the sensors a 2.5 volt DC bias. As shown below in the block diagram in Figure 3.14, this 2.5 volt DC bias is

subtracted out of each of the input signals. An example of what this signal might look like can be seen in Figure 3.15, where the example motion sensor signal represents a sensor detecting motion for the first 1.5 seconds and then detecting no motion for the remaining half second. The goal for the preprocessing step is to transform a signal which oscillates when motion is detected to a signal which is represented by higher amplitude

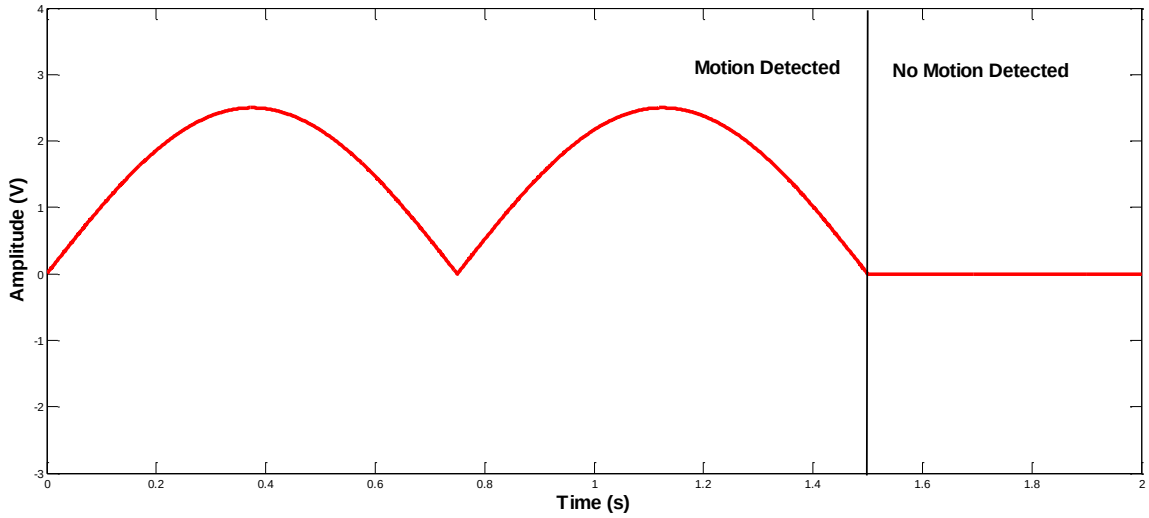

Figure 3.16: Absolute value example signal

when motion is detected and amplitude near 0 volts when no motion is being detected. With this in mind, it can be seen that the portion of the signal which drops below zero is not conducive to the overall goal of the preprocessing step. To solve this issue, the absolute value of the signal is calculated. This can be seen in Figure 3.16. Also visible in Figure 3.16, is where the absolute value of the simulated sensor signal crosses the zero axis even though motion is being detected. This is a negative effect, since it contradicts the goal of the preprocessing step. To mitigate this issue, the derivative of the simulated sensor signal is taken. As with the simulated sensor signal, the absolute value of the

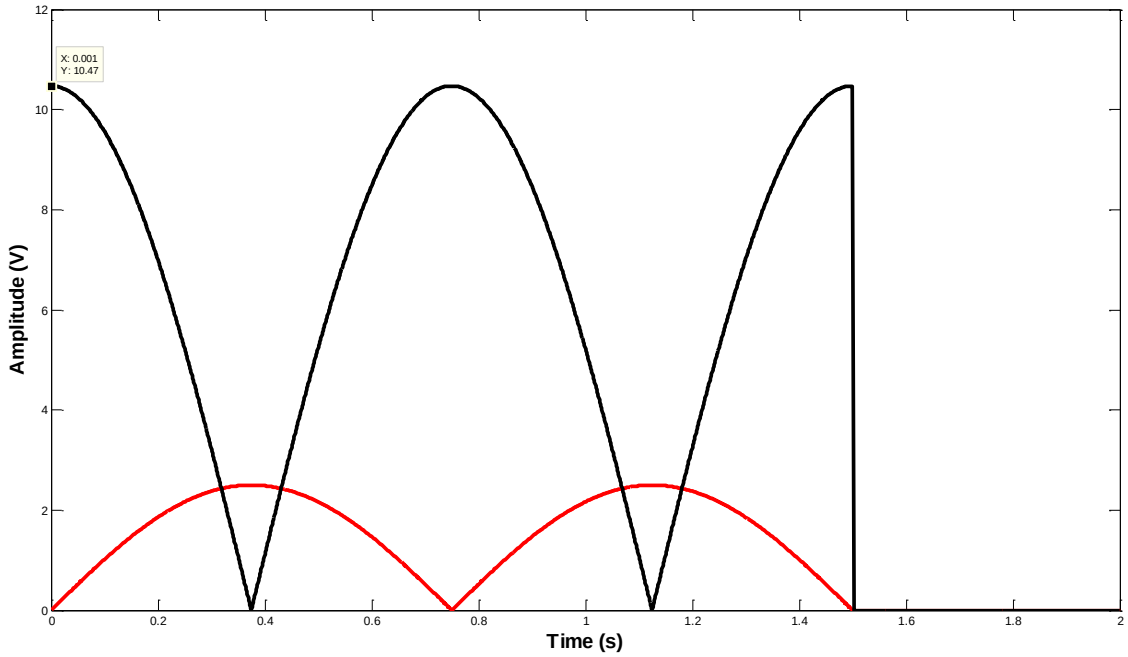

Figure 3.17: Un-scaled derivative signal

derivative was calculated to ensure that the signal reaches higher amplitude when motion is being detected. As shown in Figure 3.17, the issue of the absolute value reaching 0 can be resolved if the signals are added together. However, the combination of the sampling rate and the frequency of the signals yields a derivative signal which has a maximum amplitude of just above 10. So that the two signals can be added together, the derivative must be scaled to be within the same amplitude range as the absolute value of the simulated sensor signal.

$$Y = abs\left(\frac{dX}{dt}\right) * \frac{max\left(abs(X)\right)}{\max\left(abs\left(\frac{dX}{dt}\right)\right)} = \frac{2.5}{10} = .25 => Y = X * .25 \qquad 3\text{-}1$$

This is accomplished by multiplying the derivative signal by the maximum value of the desired signal range and dividing by the maximum value of the derivative signal.

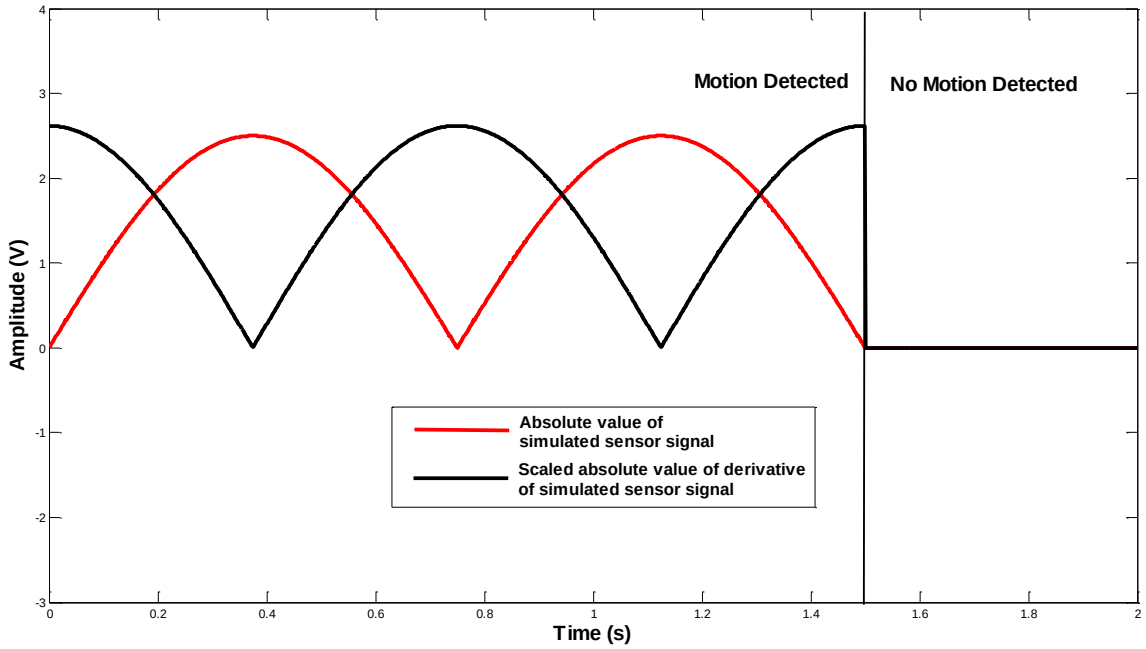

Figure 3.18: Scaled absolute value of derivative of simulated sensor signal

Performing the transform shown in equation 3-1 yields the version of the derivative signal shown in Figure 3.18, which is the absolute value of the simulated sensor signal and the absolute value of the derivative of the simulated sensor signal, which has been scaled down to the same amplitude range. After this, both signals are added together, yielding the signal shown in Figure 3.19. As shown in the figure, this preprocessing method makes it easier to identify where a fall occurs and transforms the signals into a form which reveals the features that are used for feature extraction. Once the preprocessing transformations have been made to each of the eight sensor signals, the signals from each of the four levels of sensor pairs are averaged together.

$$\begin{aligned} outputSignalLevel0 &= (outputSignal0a + outputSignal0b)/2 \\ outputSignalLevel1 &= (outputSignal1a + outputSignal1b)/2 \\ outputSignalLevel2 &= (outputSignal2a + outputSignal2b)/2 \\ outputSignalLevel3 &= (outputSignal3a + outputSignal3b)/2 \end{aligned} \qquad 3\text{-}2$$

This takes the eight initial signals and condenses them to four signals, one for each level of the sensing array. This step provides some redundancy in the output of the signals from each vertical sensing level of the array. An example of the resulting signals

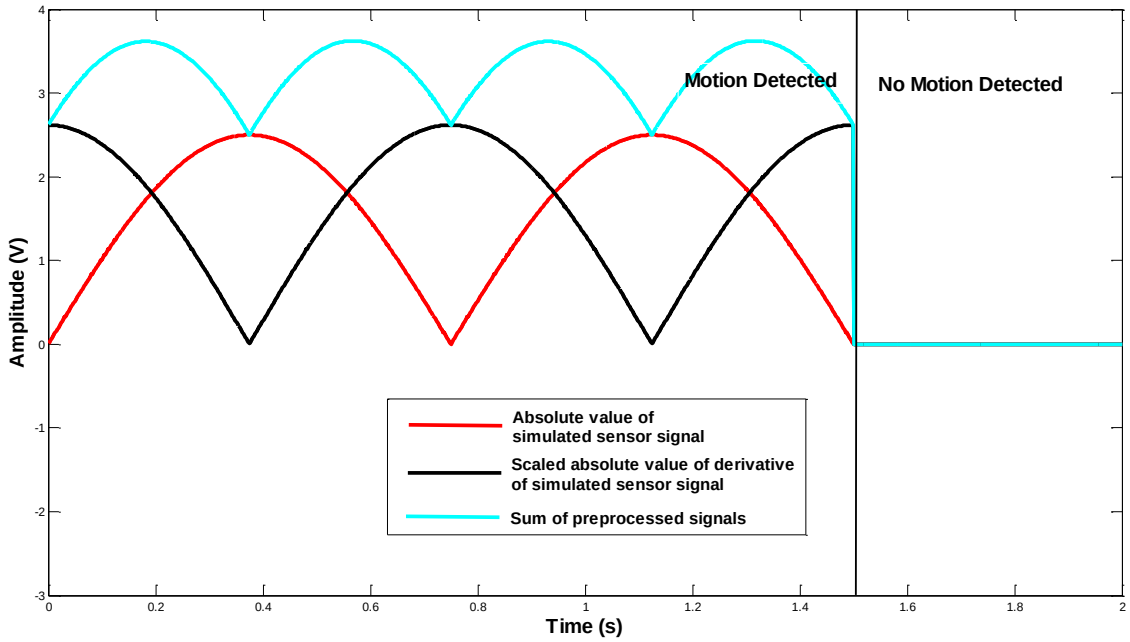

Figure 3.19: Preprocessing results from simulated signals

generated by executing the preprocessing algorithm is shown in Figure 3.20 below.

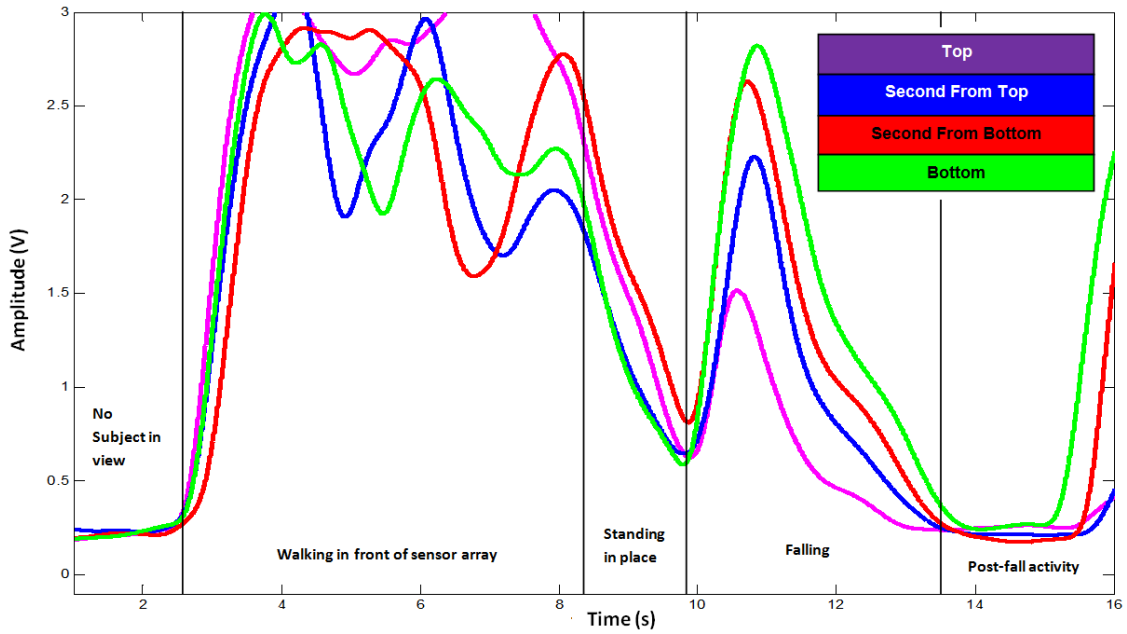

Figure 3.20: Preprocessed fall signals

3.4 Feature Extraction

Once the data has been preprocessed, the next step is to extract the features that will help the classifier identify when a fall has occurred. The features are extracted by performing calculations on the preprocessed data which tend to increase separation between data of different classes. The features that were used in this

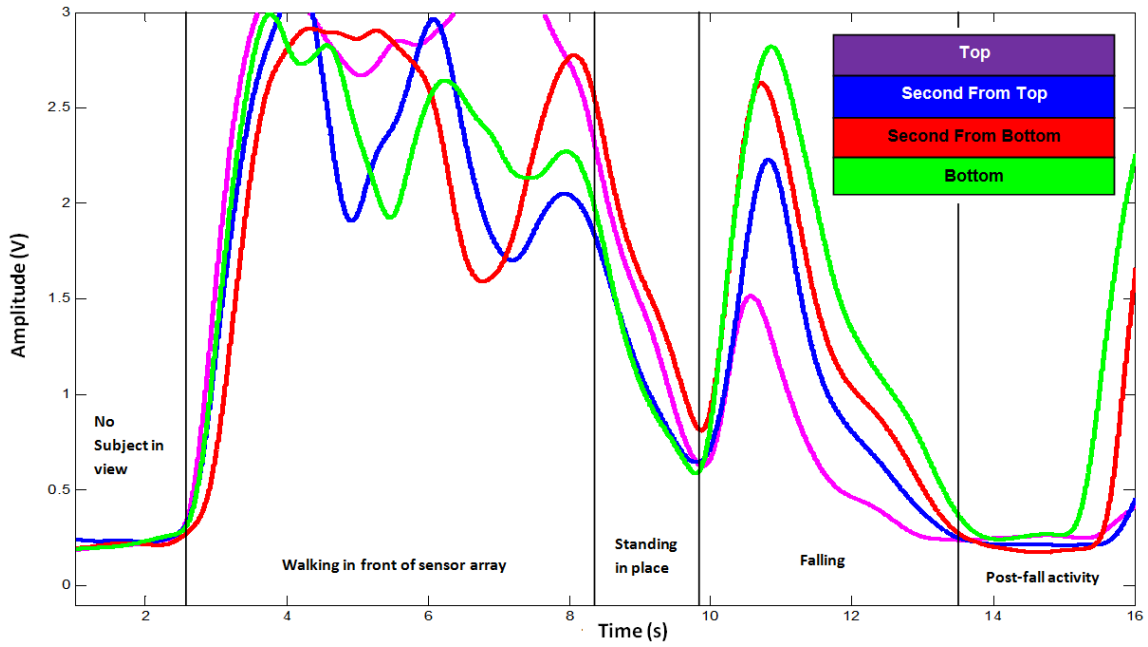

Figure 3.21: Preprocessed signals for feature extraction

research were selected based on inspection of the preprocessed data signals. When looking at preprocessed signals such as the ones shown in Figure 3.21, it can be seen that the signals appear different during the fall event than they do during the rest of the duration of the signals. These differences can be extracted as features. First, as shown in Figure 3.21, during the fall, the signals have a slightly different slope. Also, when fall

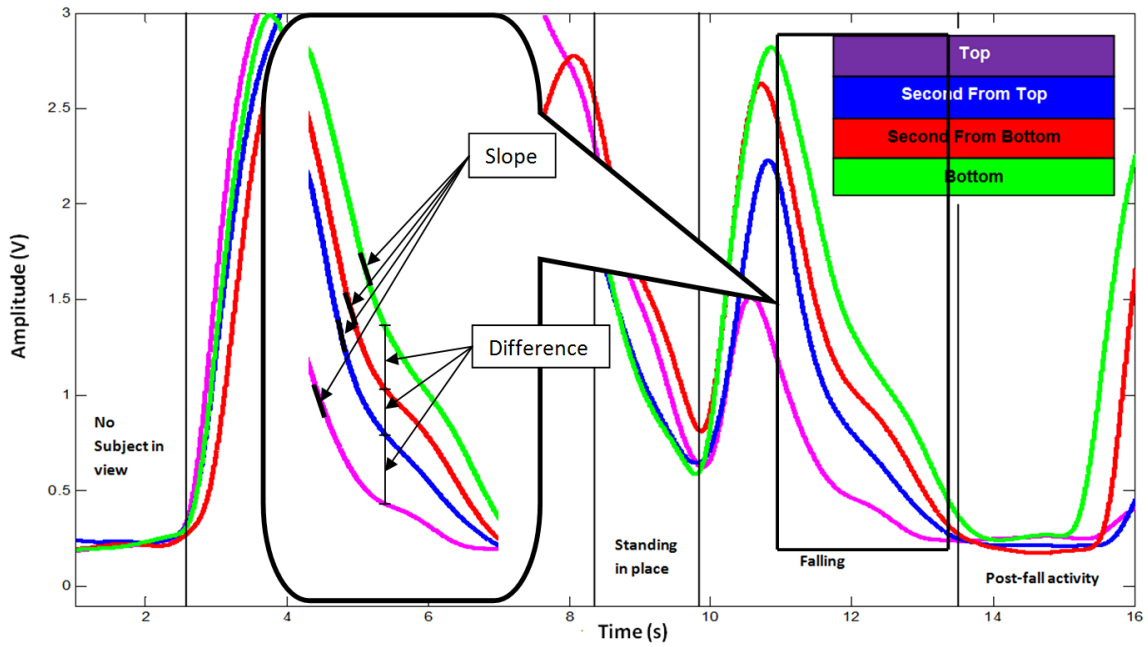

Figure 3.22: Fall signals with slope and difference

activity is observed, the slope of all four signals is typically negative. Besides the slope, when the subject is falling the signals begin to decrease in amplitude in order beginning with the purple signal which is from the top-most sensor pair. Next, the amplitude of the blue signal (the sensor pair second from top) begins to drop. Then, the amplitude from the red signal drops as its sensors cease to detect motion. Finally, the signals from the bottom sensor pair (represented by the green signal) begin to drop in amplitude. The signals continue this trend throughout the remaining duration of the fall. This pattern that was created during the fall is highlighted in Figure 3.22 where it can be seen that the slope and difference between the signals are distinct during the fall and thus, will help create separation between data collected during a fall and data collected during a non-fall.

As outlined in the flowchart and block diagram shown below in Figure 3.23 and Figure 3.24, feature extraction begins by loading the data from which features are to be extracted. After the preprocessed data file is loaded, each of the eight signals are separated into individual variables. Next, the slope feature is extracted from each signal by calculating the derivative. Since the derivative introduces some noise in the signals, a mean filter is then calculated over each signal. Doing this yields four slope features, one for each sensing level. After that, the difference feature is extracted by subtracting the amplitude of each adjacent signal where the signal generated by one sensor pair is subtracted from the signal of the sensor pair that is located below it, at each instance in time. This ends up producing a difference feature signal that is positive during the portion of the signals where a fall has occurred. This yields 3 difference features with one coming from the difference between each sensing level. Combining all of the features yields a 7 dimensional feature vector.

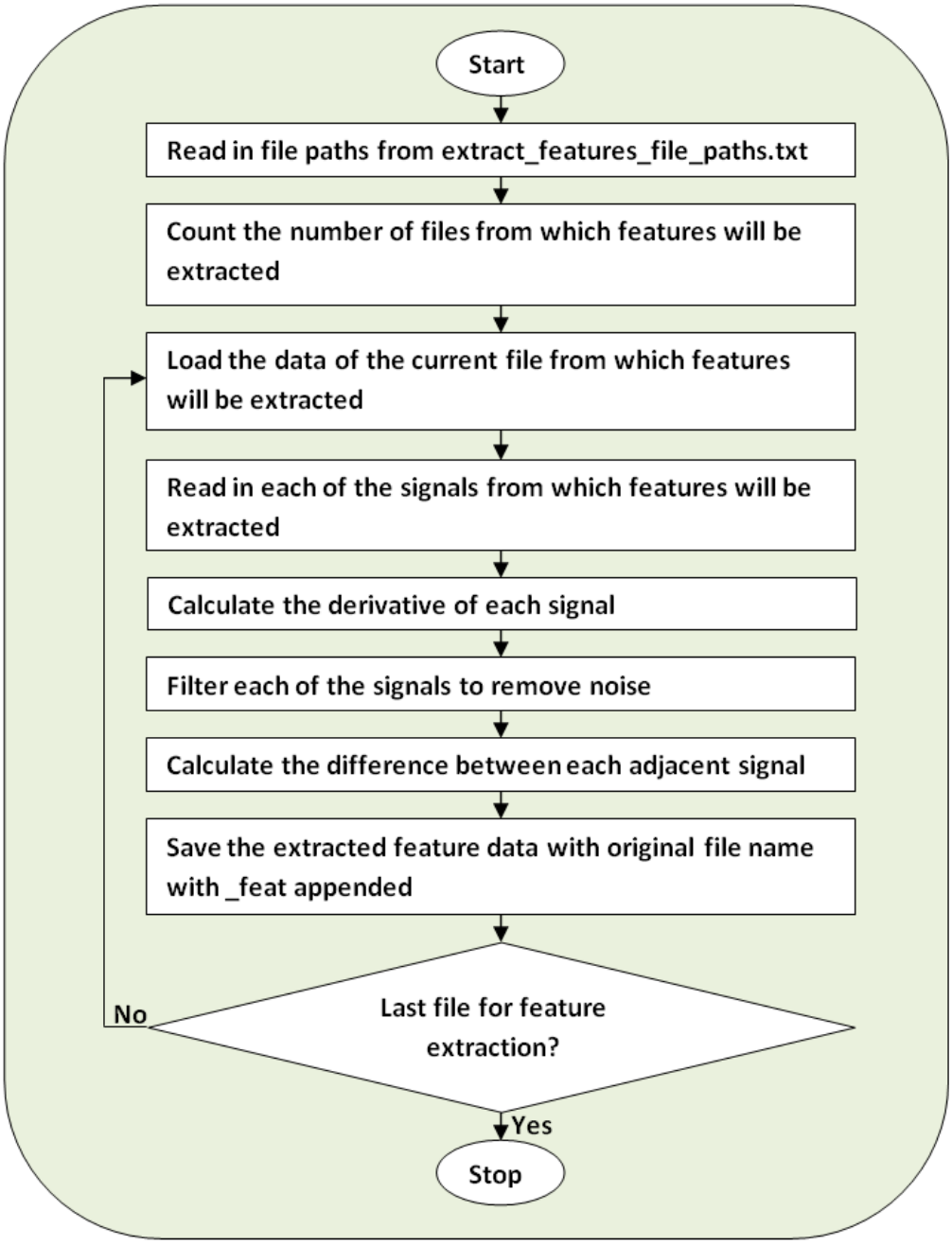

Figure 3.23: Feature extraction software flow chart

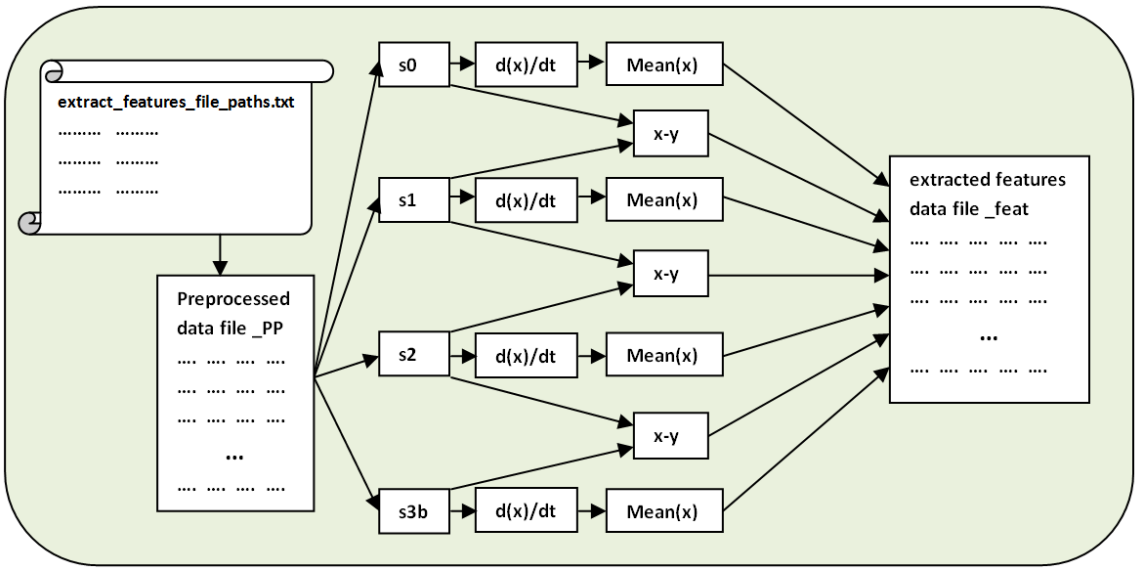

Figure 3.24: Feature extraction block diagram

To visualize the feature space, a principal component analysis (PCA) was performed to reduce the dimensionality of the feature vector from seven dimensions to three then plotted in a three dimensional graph. In Figure 3.25, it can be seen that there is some separation between the fall data and the non-fall data.

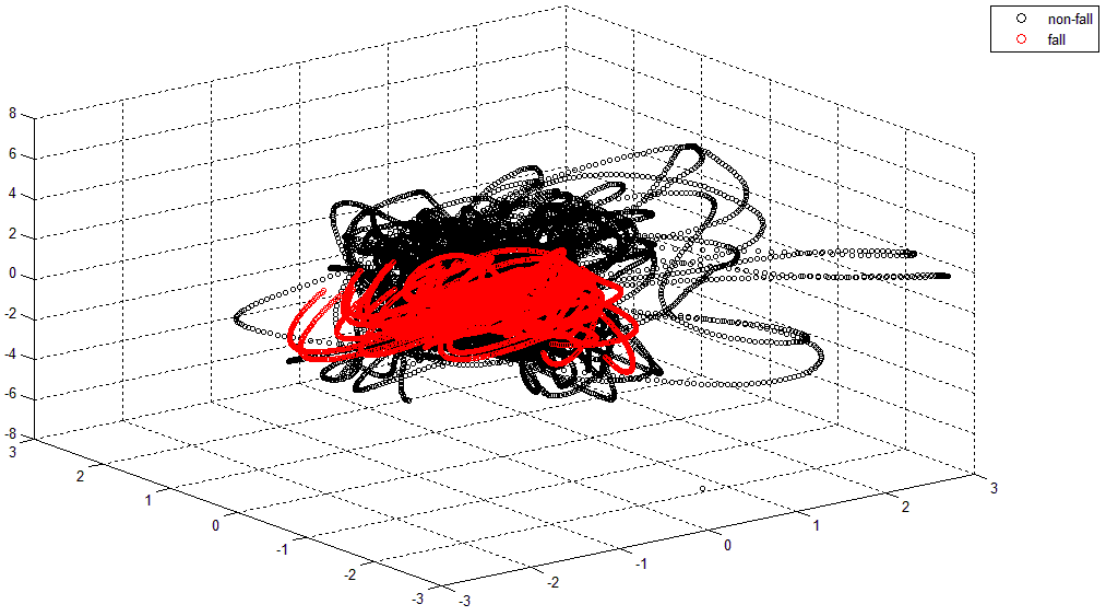

Figure 3.25: 3D feature space plot

3.5 Training Data Set

Once data has been transformed to the appropriate feature space, it is ready for classification. However, before it can be classified, it must be separated into testing and training data sets. After performing a data collection, where a stunt actress performs the actions as defined in the protocol in Appendix A, there were 57 fall and non-fall data files. Of the 57 data files, 42 of them represented fall data, and 15 of them were from the false positive portion of the protocol. When the data collection was executed, each of the fall activities were performed multiple times, with there being 1 fall activity for each data file. Conversely, the non-fall activities were performed two times within the same data file. Since there were multiple data file instances of each of the fall activities, they could easily be split between testing and training data sets. Since each of the false positive data files was collected with the stunt actress performing the individual false positive activities twice for each data file, it was not feasible to separate the false positive data files between the testing and training data sets. It was also not clear where to separate the individual activities within the non-fall data files. Since classification needed to be tested on each of the activities in the fall test protocol, the false positive data files had to be left in the testing data set and it was assumed that there would be enough fall and non-fall data in each of the fall data files in the testing data set to represent each class of data. The data files were distributed amongst the testing and

training data sets as shown below in Table 3-2 and Table 3-3.

Table 3-2: Training Data Files

#	File Name	Training Data Files Activity Description
1	201004201052.txt	Standing Position Looses Balance Fall Backward
2	201004201056.txt	Standing Position Looses Balance Fall Left
3	201004201058.txt	Standing Position Looses Balance Fall Right
4	201004201103.txt	Standing Position Momentary Loss of Consciousness Fall Forward
5	201004201105.txt	Standing Position Momentary Loss of Consciousness Fall Backward
6	201004201108.txt	Standing Position Momentary Loss of Consciousness Fall Left
7	201004201110.txt	Standing Position Momentary Loss of Consciousness Fall Right
8	201004201112.txt	Standing Position Momentary Loss of Consciousness Fall Straight Down
9	201004201117.txt	Tripping and Slipping Trip and Fall Forward
10	201004201141.txt	Tripping and Slipping Slip and Fall Forward
11	201004201143.txt	Tripping and Slipping Slip and Fall Sideways
12	201004201146.txt	Tripping and Slipping Slip and Fall Backward
13	201004201153.txt	Sitting Falling From a Chair Fall Forward
14	201004201155.txt	Sitting Falling From a Chair Fall Left
15	201004201159.txt	Sitting Falling From a Chair Fall Right
16	201004201201.txt	Sitting Falling From a Chair Sliding Forward Out of Chair
17	201004201204.txt	Sitting Falling From a Chair Sliding Backward Out of Chair
18	201004201208.txt	From Bed or Couch Fall to Side Upper Body Falls First
19	201004201211.txt	From Bed or Couch Fall to Side Hips and Shoulders Fall First

Table 3-3: Testing Data Files

Testing Data Files		
#	File Name	Activity Description
1	201004201051.txt	Standing Position Looses Balance Fall Forward
2	201004201053.txt	Standing Position Looses Balance Fall Backward
3	201004201055.txt	Standing Position Looses Balance Fall Backward
4	201004201057.txt	Standing Position Looses Balance Fall Left
5	201004201100.txt	Standing Position Looses Balance Fall Right
6	201004201101.txt	Standing Position Looses Balance Fall Right
7	201004201104.txt	Standing Position Momentary Loss of Consciousness Fall Forward
8	201004201107.txt	Standing Position Momentary Loss of Consciousness Fall Backward
9	201004201109.txt	Standing Position Momentary Loss of Consciousness Fall Left
10	201004201111.txt	Standing Position Momentary Loss of Consciousness Fall Right
11	201004201114.txt	Standing Position Momentary Loss of Consciousness Fall Straight Down
12	201004201138.txt	Tripping and Slipping Trip and Fall Forward
13	201004201139.txt	Tripping and Slipping Trip and Fall Sideways
14	201004201142.txt	Tripping and Slipping Slip and Fall Forward
15	201004201144.txt	Tripping and Slipping Slip and Fall Sideways
16	201004201147.txt	Tripping and Slipping Slip and Fall Backward
17	201004201154.txt	Sitting Falling From a Chair Fall Forward
18	201004201158.txt	Sitting Falling From a Chair Fall Left
19	201004201200.txt	Sitting Falling From a Chair Fall Right
20	201004201203.txt	Sitting Falling From a Chair Sliding Forward Out of Chair
21	201004201206.txt	Sitting Falling From a Chair Sliding Backward Out of Chair
22	201004201210.txt	From Bed or Couch Fall to Side Upper Body Falls First
23	201004201212.txt	From Bed or Couch Fall to Side Hips and Shoulders Fall First
24	201004201310.txt	Standing to Squatting
25	201004201314.txt	Standing to Kneeling
26	201004201315.txt	Standing to Kneeling to Lying on Floor
27	201004201318.txt	Bend Down Plug in Appliance
28	201004201322.txt	Squat to Tie Shoe
29	201004201325.txt	Standing to Sitting Legs Tucked
30	201004201327.txt	Standing to Sitting Legs Extended
31	201004201329.txt	Standing to Situps and Stretches
32	201004201331.txt	Lying to Kneeling to Standing
33	201004201335.txt	Walking Trip But Regain Balance
34	201004201338.txt	Walking Forward and Stop Suddenly
35	201004201340.txt	Walking Forward and Stop Suddenly Turn Around
36	201004201341.txt	Walk to Chair and Sit
37	201004201343.txt	Walk to Chair and Sit Pickup Book
38	201004201344.txt	Walk to Chair and Sit Attempt to Stand

Once the data files were distributed to testing and training data sets, the training data set had to be separated into different classes so that it could be used by a classifier to classify testing data. To identify which data came from each class in each of the fall data files, a person had to inspect the preprocessed data files and identify the sample

where the data collected during a fall begins and the sample where the fall data ends. The data between these samples were assumed to be collected during a fall and was assigned to class 1. Once the class 1 or fall data were identified, the rest of the data were assumed to be class 2 or non-fall data. This yielded a training data set with 35,220 class 1 or fall data points, and 327,800 class 2 or non-fall data points. Combining the fall and non-fall training data gives a training data set with 363,020 data points. With the testing data set that large, it took a very long time to classify just one fall. To solve this issue, the training data set was down-sampled. Since the number of class 2 data was near 1 order of magnitude higher than the number of class 1 data, the class 1 data was down-sampled by 10 and the class 2 data was down-sampled by 100 to even up the number of samples in each class. This gave a training data set with 6,800 data points where 3,522 of them represent class 1 and 3,278 represent class 2. This likely has little negative effect on the results because of the estimated 10 times sampling frequency as discussed in section 3.2. The number of feature vectors included in the training data sets for each of the classifiers and the number of windows used in the testing data are shown below in Figure 3.26.

	Training (Feature Vectors)		Testing (Windows)	
	Falls	Non-falls	Falls	Non-falls
Parzen Window	35,220/10 = 3,522	327,800/100 = 3,278	790	26,030
RVM (after training)	452	1,271	790	26,030

1 Window = 50 Samples

Figure 3.26: Testing and training feature vector and window quantities

Through the remainder of the classification and post-processing process the data is handled in 50 sample windows. This is because the falls in the stunt actress data set typically happen over a .5 second duration, so each window would be .1 times the duration of a fall. This would likely provide enough resolution for fall detection by each of the classifiers.

3.6 Parzen Window

One of the two classification methods chosen for this research is the Parzen Window classifier. Although, as stated in section 2.3, the Parzen Window classifier is a computationally intensive classifier; the results of this classifier can be used as a reference to determine whether falls can be detected using the PIR sensing array

described in section 3.1. The algorithm used in this research to implement the Parzen Window is outlined in the block diagram below in Figure 3.27.

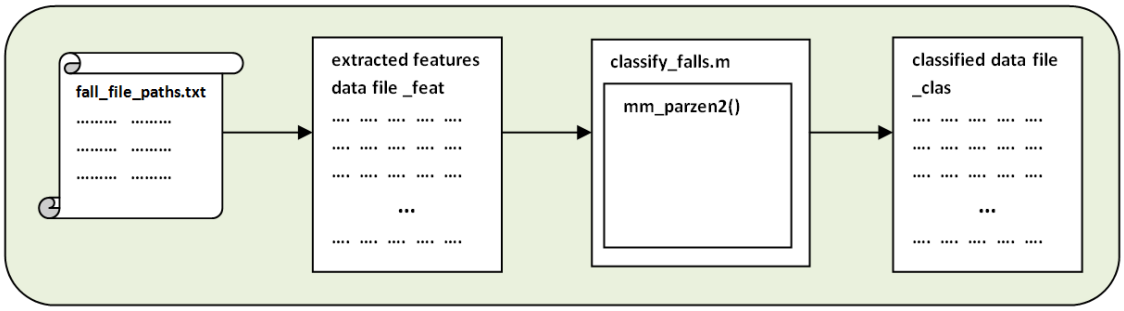

Figure 3.27: Parzen window classification block diagram

The Parzen Window does not require a prolonged training step; all that is necessary is for the data to be separated into training and testing sets. This process was detailed in section 3.5. Once the training data set is defined, the Parzen Window loads the training data that will be used to classify the testing data as is shown in the flowchart in Figure 3.28.

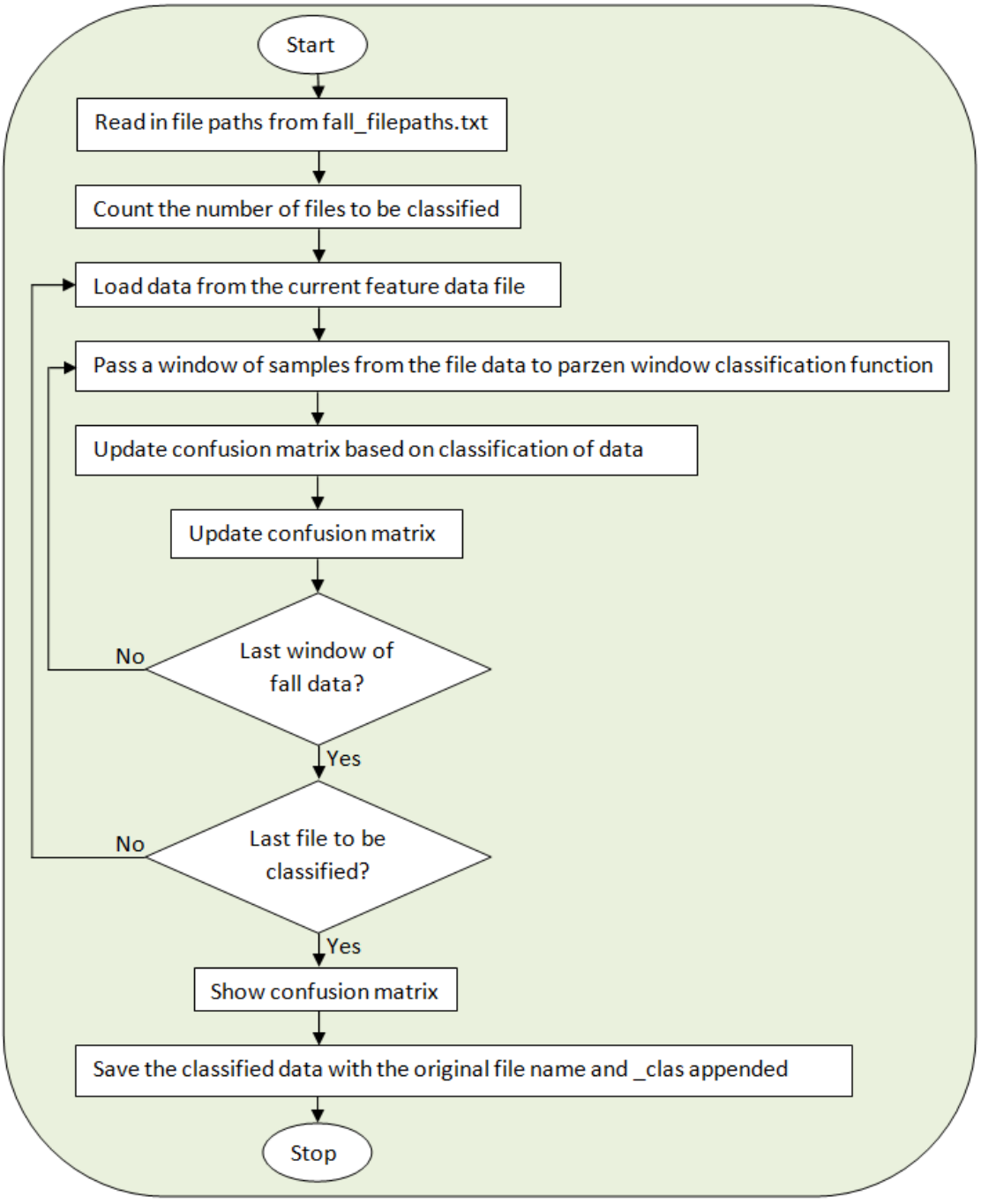

Figure 3.28: Parzen window classification software flowchart

Once the training data set is loaded, an appropriate value for the magnitude of the variance or h parameter is determined. This was done by running the classifier on

some testing data, and using the information reported by the resulting confusion matrix to choose the magnitude of h empirically, through experimentation by varying the h parameter and choosing the value of h that yielded the lowest number of false negatives. The results of this process are show below in Table 3-4.

Table 3-4: h parameter tuning

		___FN___	___FP___		
h	TP	Missed Fall	False Alarm	TN	Accuracy
5	2201	0	16880	7919	37.48
0.5	2201	0	8148	16651	69.82
0.1	2201	0	3865	20935	85.69
0.01	1063	1138	0	24799	95.79
0.05	2201	0	1713	23086	93.66

Initially, h was set high at 5. The result of setting h to 5 yielded no false negatives and a very large number of false positives. From 5, h was lowered one order of magnitude to .5. Lowering h to .5 greatly reduced the number of false positives. Next, an h value of .1 was tested. As shown in Table 3-4, this, once again, improved the accuracy. After that, a value of .01 was tested. This yielded the best accuracy, but increased the number of false negatives, which represent missed falls. Finally, h was set to .05. This yielded the best accuracy while still keeping the number of false negatives at 0.

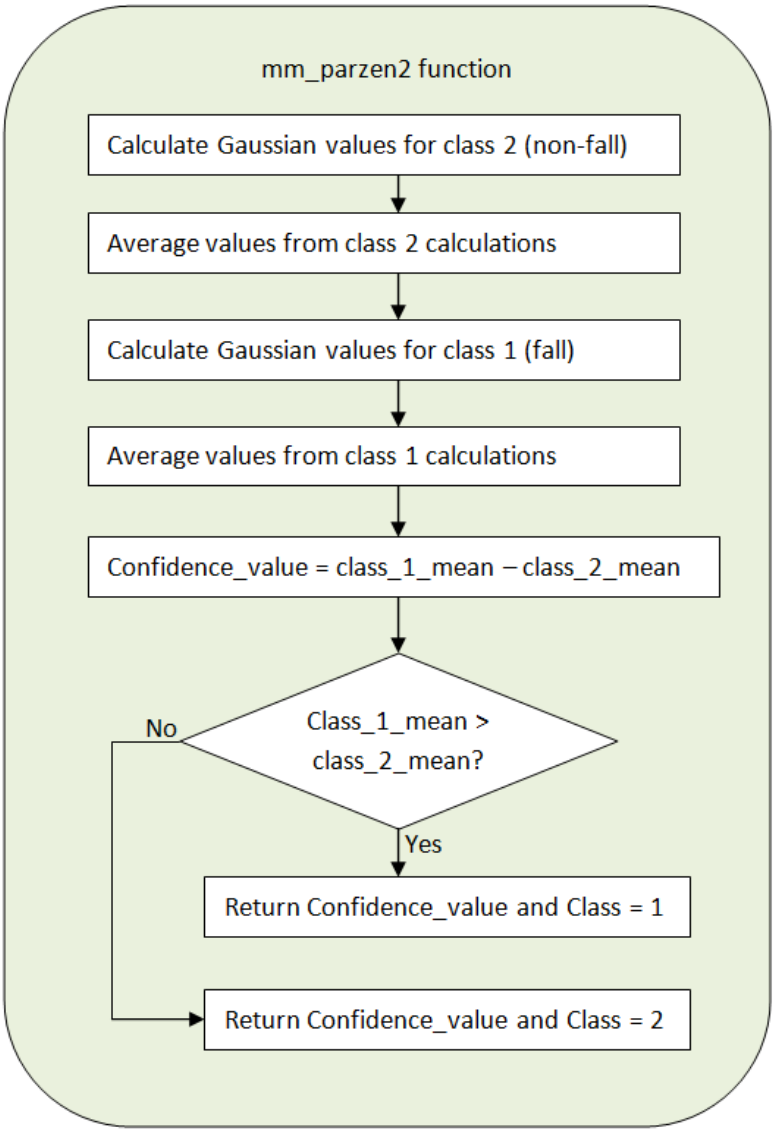

Figure 3.29: Parzen window classification function flowchart

As can be seen above in Figure 3.29, the first step the classification function takes is to calculate the cumulative probability of the Gaussian distribution for each testing sample, centered at each training sample, from the class 2 training data set. The equation for this step is shown below in equation 3-3.

$$\text{C2}parzenVals(j,i) = \frac{1}{h\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{\|\text{x}_\text{j}-\text{x}_\text{i}\|}{\text{h}}\right)^2\right) \begin{matrix}\text{j} = 1,\ldots,\text{length(C2trainData)} \\ \text{i} = 1,\ldots,\text{length(testData)}\end{matrix} \quad \text{3-3}$$

After that, the average of the values generated from the calculations between class 2 training samples and testing samples is calculated.

$$Class2mean\ = mean(C2parzenVals) \quad \text{3-4}$$

Next, the classification function calculates the cumulative probability of the Gaussian distribution for each testing sample centered at each training sample from the class 1 training data set. The equation for this step is shown below in equation 3-5.

$$\text{C1}parzenVals(j,i) = \frac{1}{h\sqrt{2\pi}}\exp\left(-\frac{1}{2}\left(\frac{\|\text{x}_\text{j}-\text{x}_\text{i}\|}{\text{h}}\right)^2\right) \begin{matrix}\text{j} = 1,\ldots,\text{length(C1trainData)} \\ \text{i} = 1,\ldots,\text{length(testData)}\end{matrix} \quad \text{3-5}$$

Once that is complete, the average of the values generated from the calculations between class 1 training samples and testing samples is calculated.

$$Class1mean\ = mean(C1parzenVals) \quad \text{3-6}$$

Next, the confidence value is calculated by subtracting the mean of the cumulative probability calculations between class 2 training data and the testing data from the cumulative probability calculations between class 1 training data and the testing data. Since the bandwidth is set to allow more false positives than false negatives, if the confidence value is below zero, this would more likely result in the classification for the testing data being class 2 or a non-fall and a confidence value would not be necessary and as such all negative confidence values are set to be zero.

$$confidenceValue = \begin{cases} Class1mean - Class2mean, & Class1mean > Class2mean \\ 0, & otherwise \end{cases} \quad 3\text{-}7$$

After the confidence value is calculated, the values of the class 1 mean and class 2 mean are compared. If the class 1 mean is greater than the class 2 mean, then the testing sample is classified as a fall, and if the class 2 mean is greater than the class 1 mean then the testing sample is classified as a non-fall. Once the class label is determined, the 2 class Parzen Window function returns the class label and the confidence values.

3.7 Relevance Vector Machine (RVM)

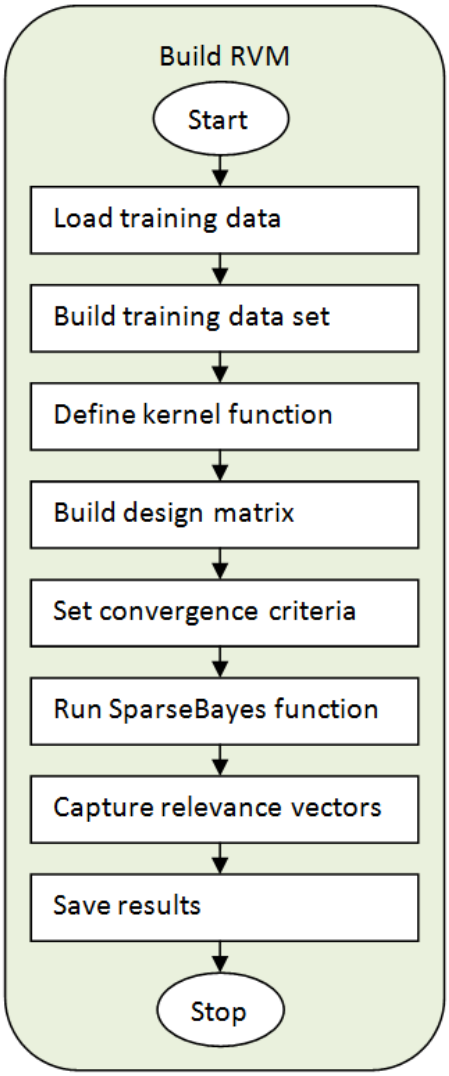

Figure 3.30: RVM training flowchart

An alternative classification method to the Parzen Window classifier is to use the RVM to provide the predictive distribution of individual fall data files, which allows for classification of the fall data. As mentioned in section 3.6, the Parzen Window approach requires a significant amount of processing power and storage space. Alternatively, the training step executed when implementing the RVM, effectively reduces the training data set to a more sparse set of relevance vectors. The reduction of training samples translates to lower computational complexity and requires less storage space. The RVM is a kernel method where a kernel function is used to transform the input data to a feature space mapping.

In contrast to the Parzen Window, classification using the RVM requires a training process where input parameters for classification are found. As shown in Figure 3.30, training of the RVM is started by loading the training data. Next, a training data set is configured by combining the training data with target values that represent the class of the corresponding training data. In this research, class 1 training data, or fall data, is

given a target value of -1 and class 2 training data, which consists of non-fall data, is given a target value of 1. This can be seen below in equation 3-8.

$$[\boldsymbol{x} \quad t] = \begin{bmatrix} class\ 1\ training\ data & -1 \\ class\ 2\ training\ data & 1 \end{bmatrix} \tag{3-8}$$

Next, in the training process, the kernel function $\varphi(x)$ is defined. For this research, since the data set is assumed to be nonlinear, the radial basis function is used.

$$\varphi(\boldsymbol{x}_i, \boldsymbol{x}_j) = e^{\left(\frac{-1}{2\pi^2}\|x_i - x_j\|\right)} \tag{3-9}$$

Once the kernel function is defined, the design matrix $\boldsymbol{\Phi}$ can be built based on this kernel function.

$$\boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{bmatrix} \varphi(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & \varphi(\boldsymbol{x}_1, \boldsymbol{x}_J) \\ \dots & \dots & \dots \\ \varphi(\boldsymbol{x}_I, \boldsymbol{x}_1) & \dots & \varphi(\boldsymbol{x}_I, \boldsymbol{x}_J) \end{bmatrix} \tag{3-10}$$

After the design matrix is initialized, the next step, before running the function that trains the RVM, is to set the convergence criteria. For this research, the SparseBayes function, written by Mike Tipping [19], is used to train the RVM. For the convergence criteria, the SparseBayes function requires the user to set the max iterations and maximum time to run, which in this research, were both set to 50,000. At completion, the SparseBayes function passes back the relevance vectors in the form of indices to the most relevant vectors. To use these data, the relevance vectors are saved to a matrix $\boldsymbol{x}$ and all of the data returned by the SparseBayes function are saved.

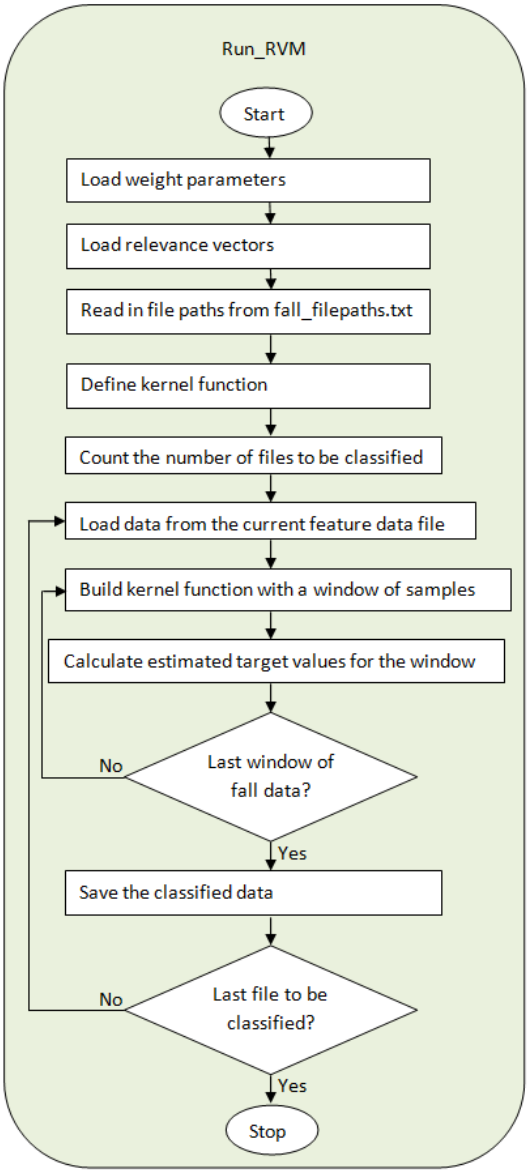

Figure 3.31: RVM testing software flowchart

Once the RVM has been trained, it can be used to classify the fall data. As can be seen in the software flowchart shown in Figure 3.31, the classification process of the RVM begins by loading the weight parameters $\boldsymbol{w}$ and relevance vectors $\boldsymbol{x}$. After the training data are loaded, the feature vector which is to be classified is pointed to when the corresponding file paths are loaded. Next, the kernel function $\varphi(x)$ is defined using equation 3-9. Once the data have been initialized, the main loop of the classification software begins by loading in a testing data file. Once the testing data is loaded, the design matrix can be made with a window of fall data.

$$\boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{x}_j) = \begin{bmatrix} \varphi(\boldsymbol{x}_1, \boldsymbol{x}_1) & \dots & \varphi(\boldsymbol{x}_1, \boldsymbol{x}_J) \\ \dots & \dots & \dots \\ \varphi(\boldsymbol{x}_I, \boldsymbol{x}_1) & \dots & \varphi(\boldsymbol{x}_I, \boldsymbol{x}_J) \end{bmatrix} \quad \begin{matrix} \mathrm{i} = 1, \dots, \mathrm{length(dataWindow)} \\ \mathrm{j} = 1, \dots, \mathrm{length(relevanceVectors)} \end{matrix} \qquad 3\text{-}11$$

Next, the target values are estimated for the window of fall data using the equation below.

$$\boldsymbol{t} = \boldsymbol{\Phi}(\boldsymbol{x}_i, \boldsymbol{x}_j)\boldsymbol{w} \quad \text{3-12}$$

After all of the data windows for each data file have been classified the classification data for each of the individual data files are saved.

The results of the target data calculations are probabilistic classifications which represent the confidence that a window of data belongs under a fall or a non-fall class label. This is a good fit for this research because it allows for post processing of the classification data to improve the accuracy of the classification results.

3.8 Post-processing

As will be discussed in chapter 4, the results from both the Parzen Window and the RVM require post-processing to provide more accurate results. For example, as shown in Figure 3.32, there is a false positive that occurs briefly at around 4500 samples

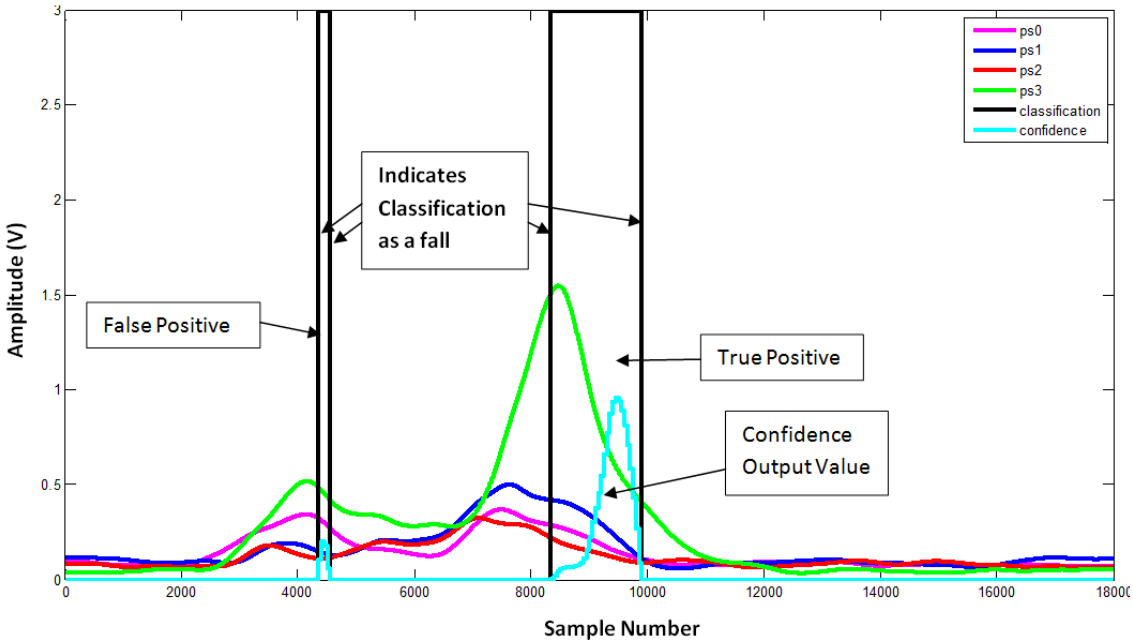

Figure 3.32: Classified data graph

and a true positive that occurs between 8500 and around 10500 samples.

As shown in Figure 3.32, there is more area under the curve of the true positive than there is under the false positive. Since the false positive is an undesirable result, as it represents a situation where caretakers may be unnecessarily alerted to a problem that may not have actually happened, it would be best if this false positive were correctly classified as a non-fall. To solve this issue in post-processing, a mean filter is calculated over the classification signal. This more quickly reduces the peak amplitude of the classification with less area under its curve and will effectively create more separation between false positives and true positives. It also makes determining a threshold value that can be chosen for the maximum amplitude of non-fall data more apparent.

The false positive, shown in Figure 3.32, could be eliminated if a threshold value could be found where, if a classification value is above this threshold, it is considered a true fall, and if it is below this value, then it would be considered a non-fall. To choose the best threshold, all of the falls must be analyzed at each potential threshold value. In determining this threshold value, a receiver operating characteristic (ROC) curve is generated by calculating the true positive rate and false positive rate across a range of threshold values. The ROC signal is shown on a 2 dimensional graph where the horizontal axis is the false positive rate. The calculation for the false positive rate is shown below in equation 3-13 where the false positive rate FP equals the number of false positives divided by the total number of data that are supposed to be negative which is the number of false positives (#FP) and the number of true negatives (#TN).

$$FP = \frac{\#FP}{\#FP + \#TN} \quad \text{3-13}$$

The vertical axis of the ROC curve is the true positive rate. The calculation for the true positive rate is shown below in equation 3-14 where the true positive rate TP equals the number of true positives divided by the total number of data that are supposed to be positive which is the number of true positives (#TP) and the number of false negatives (#FN).

$$TP = \frac{\#TP}{\#TP + \#FN} \quad \text{3-14}$$

To generate the ROC graph, the true positive rate and false positive rate are calculated as the threshold is varied. For this research, the threshold value is varied from 0 to 1 in increments of .1.

The results of the ROC curve for the classification data from the Parzen Window and RVM are shown below in Figure 3.33. The ROC curves shown in Figure 3.33 were generated by calculating confusion matrix data where the results were generated by comparing the classification results from each of the Parzen Window and RVM classifiers to ground truth information over a range of threshold values. The data in Figure 3.33 was also after the mean filter was calculated in the post-processing step.

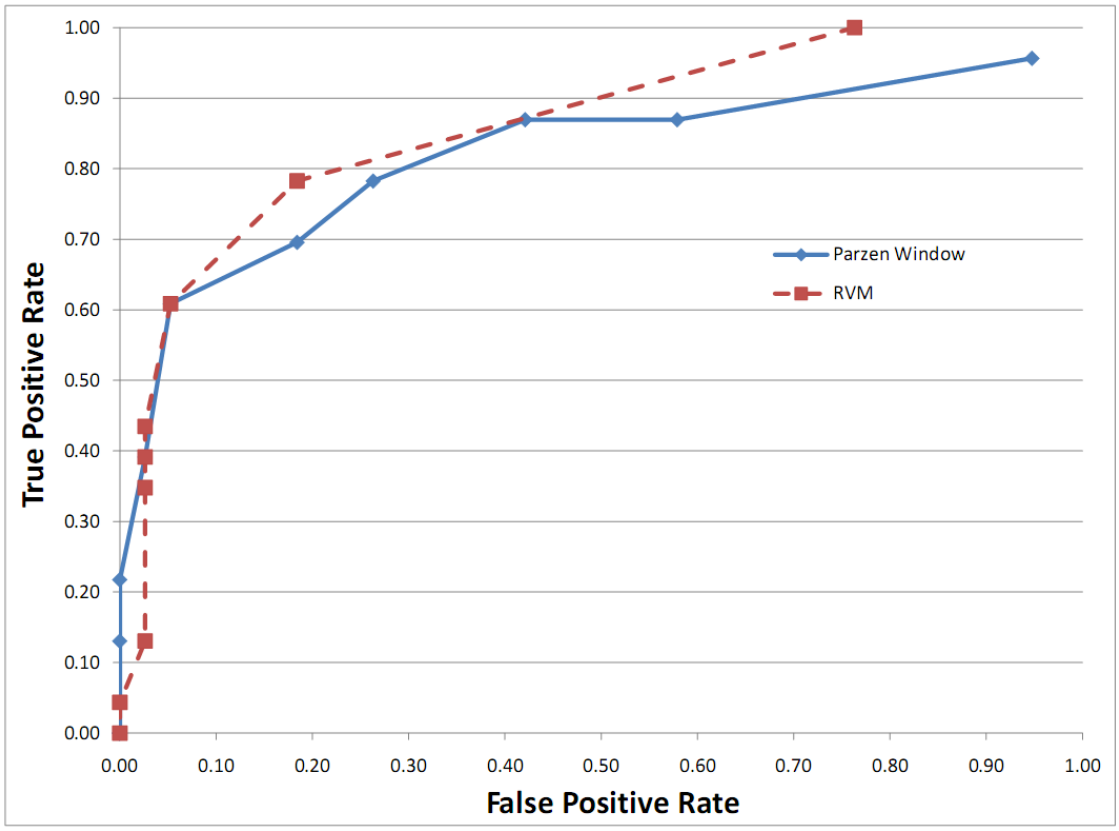

Figure 3.33: ROC curve for the Parzen Window and RVM classification results, after filtering

The ROC curve above was generated from the values in Table 3-5 below. The ROC curves in Figure 3.33 help to visualize the options available for the choice of threshold value which will minimize the false positive rate and maximize the true positive rate. However, looking at the values in the table is most helpful when choosing a threshold value to be used for classification in the final sensing array. Looking at the confusion matrix data in Table 3-5 for the Parzen Window, it can be seen that as the threshold value increases the number of false negatives increases. This is because as the threshold increases, there are more true falls that would have been classified as such but are not because their peak amplitude never reaches the threshold value. This is a

Table 3-5: Confusion Tables for ROC calculations

	Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
Parzen Window	0.0	22	1	36	2	0.96	0.95	39.34
	0.1	20	3	22	16	0.87	0.58	59.02
	0.2	20	3	16	22	0.87	0.42	68.85
	0.3	18	5	10	28	0.78	0.26	75.41
	0.4	16	7	7	31	0.70	0.18	77.05
	0.5	14	9	2	36	0.61	0.05	81.97
	0.6	9	14	1	37	0.39	0.03	75.41
	0.7	5	18	0	38	0.22	0.00	70.49
	0.8	3	20	0	38	0.13	0.00	67.21
	0.9	0	23	0	38	0.00	0.00	62.30
	1.0	0	23	0	38	0.00	0.00	62.30
	1.1	0	23	0	38	0.00	0.00	62.30
RVM	0.0	23	0	29	9	1.00	0.76	52.46
	0.1	18	5	7	31	0.78	0.18	80.33
	0.2	14	9	2	36	0.61	0.05	81.97
	0.3	10	13	1	37	0.43	0.03	77.05
	0.4	9	14	1	37	0.39	0.03	75.41
	0.5	8	15	1	37	0.35	0.03	73.77
	0.6	3	20	1	37	0.13	0.03	65.57
	0.7	1	22	0	38	0.04	0.00	63.93
	0.8	1	22	0	38	0.04	0.00	63.93
	0.9	0	23	0	38	0.00	0.00	62.30
	1.0	0	23	0	38	0.00	0.00	62.30
	1.1	0	23	0	38	0.00	0.00	62.30

negative side effect of increasing the classification threshold value. In contrast, it can be seen that the number of false positives sharply decreases as the threshold value increases. Reducing the number of false positives is the major advantage to implementing a threshold value. As can be seen in Table 3-5 initially the number of false negatives slowly increases when the threshold increases and as the threshold value approaches 1, the number of false negatives sharply increases. Thus, for the sake of keeping the number of missed falls low, a small threshold value should be chosen to minimize the number of false negatives. On the other hand, as the threshold value begins to increase, the number of false positives sharply decreases, and as the threshold

value approaches 1, the number of false positives more slowly approaches 0. From this, it can be seen that a threshold value should be chosen that will minimize the number of false negatives as well as minimize the number of false positives. In this research, for the Parzen Window, a threshold value of 0.3 was chosen because it provides a good balance between a low number of false negatives and false positives. The same trend can be seen in the fall classification data for the RVM, except the numbers for the false negatives and false positives are minimized at a lower threshold of 0.1.

Once an appropriate threshold value has been determined, the data can be classified. Figure 3.34 and Figure 3.35, below, are the post-processed (filtered) graphs of the same data shown in Figure 3.32. In both cases, the false positive was filtered out in the post-processing step.

The post-processing step is the final component in the fall detection process for the vertical array of PIR motion detectors. From this the accuracy of the classifiers can be estimated using the accuracy calculated from the confusion data that was found when generating the ROC curve. The equation for the calculation of the accuracy is shown below. In Chapter 4, the accuracy equation below is used to show the classification accuracy before and after the mean filter is applied to the classification data.

$$accuracy = \frac{\#TP + \#TN}{\#TP + \#TN + \#FP + \#FN} \qquad 3\text{-}15$$

Below Figure 3.34 and Figure 3.35 show the post-processing results from the Parzen Window and the RVM respectively. These results were derived from

implementing the post-processing algorithm on the same data file that was used to generate Figure 3.32 at the beginning of this section. It is interesting to note that after implementing the mean filter and using a threshold value, the false positive can be filtered out.

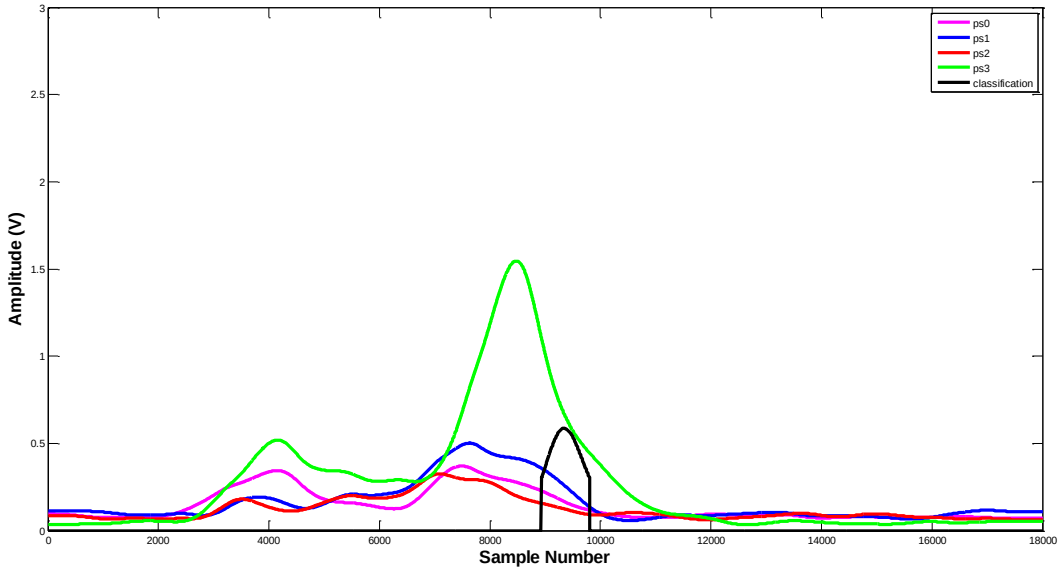

Figure 3.34: Post-processed (filtered) Parzen Window classification data

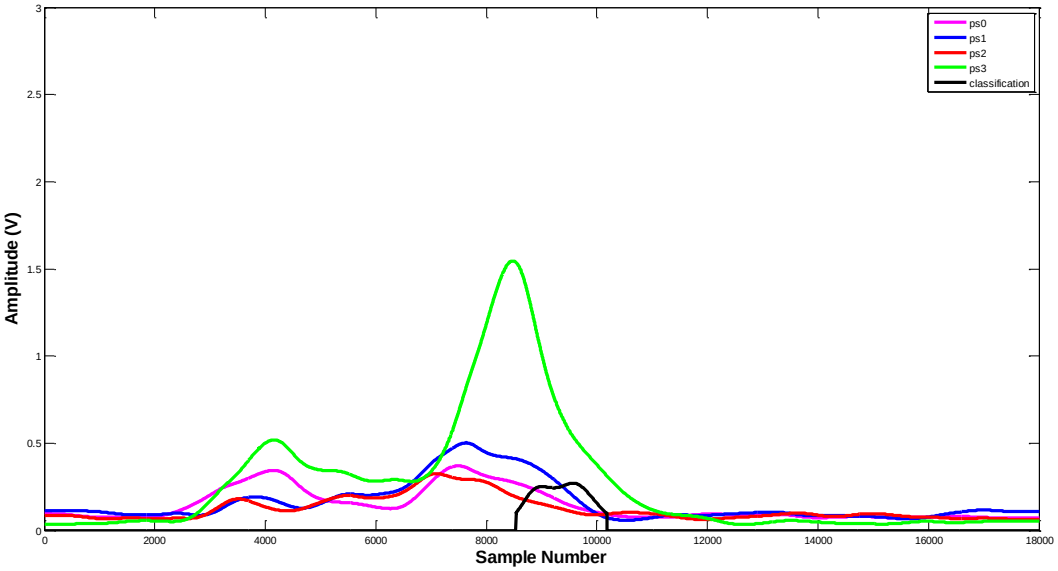

Figure 3.35: Post-processed (filtered) RVM classification data

Chapter 4 - Experimental Results and Analysis

4.1 Stunt Actress Data Set

The data set used in this research was captured in a lab which was set up to mimic an apartment at an assisted living community. The data were collected using the data acquisition system described in section 3.2. Because it would be too much of a risk for injury, it was not possible to have an elderly person perform falls for data collection. Instead, each of the data sets was collected while a stunt actress performed actions defined in a fall/non-fall protocol detailed in Appendix A. In the case of a fall, a mat was placed around 8 feet in-front of the sensing array and the stunt actress performed the fall on the mat. In the case of a non-fall, if there were any props required for the action, they were placed around 8 feet in the center of the field of view (FOV) of the sensing array where the action was performed in that general area. Once the data were collected, all of the data files which contained a fall were manually inspected to identify the approximate sample number where the fall began and ended. Below in Figure 4.1, the sample numbers where the clapping portion ends, the fall data begins, and the fall samples end, are visualized. In this research these three sample points are recorded for each data file.

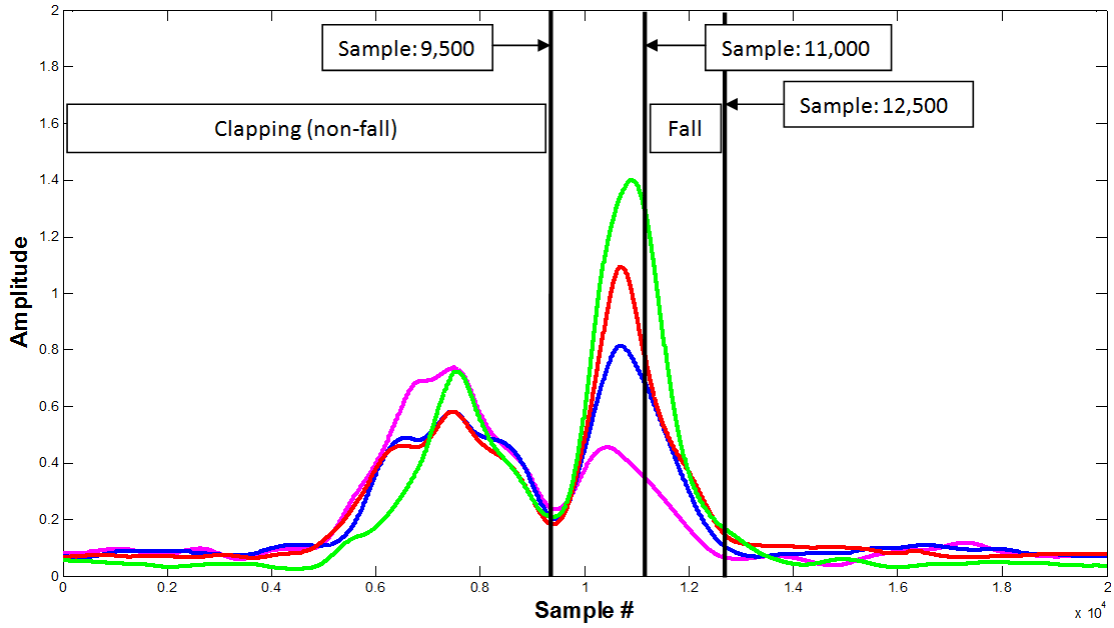

Figure 4.1: Example data file where the sample numbers of the clapping portion and falling portion of the data are identified

After the location of each fall had been identified, the data set was separated into a training set and a testing set. During the data collecting session, each of the falls defined in the fall/non-fall protocol were performed multiple times and thus when separating the data into training and testing sets, the duplicate data runs were separated, with one going to the training set and the other going to the testing set. As seen in Figure 4.2 the data in the fall data files that were part of the testing set were separated by class where the clapping portion of the fall activities were simply considered to be non-fall or class 1 data and the fall data were considered to be fall or class 1 data.

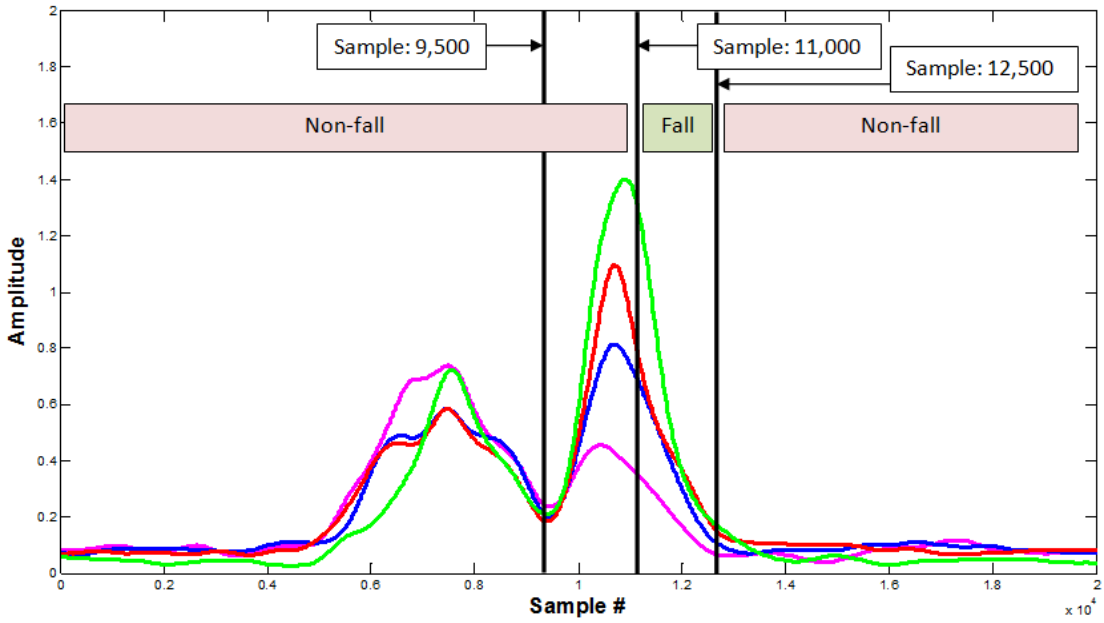

Figure 4.2: Training fall data organization

Alternatively, for experimentation purposes, the testing data were classified with the clapping portion, and without. As shown Figure 4.3, when the data were analyzed with the clapping portion, the clapping activity was considered to be a separate non-fall data run.

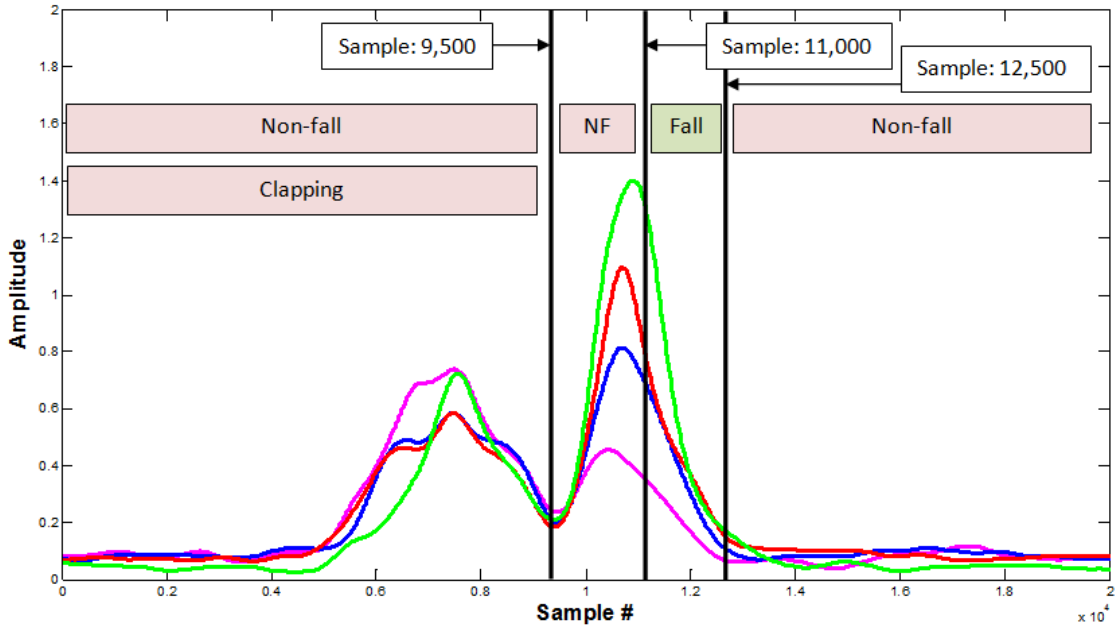

Figure 4.3: Testing data with clapping activity

In experiments where the clapping activity was not considered, it was not included, even with the non-fall data. This is shown in Figure 4.4.

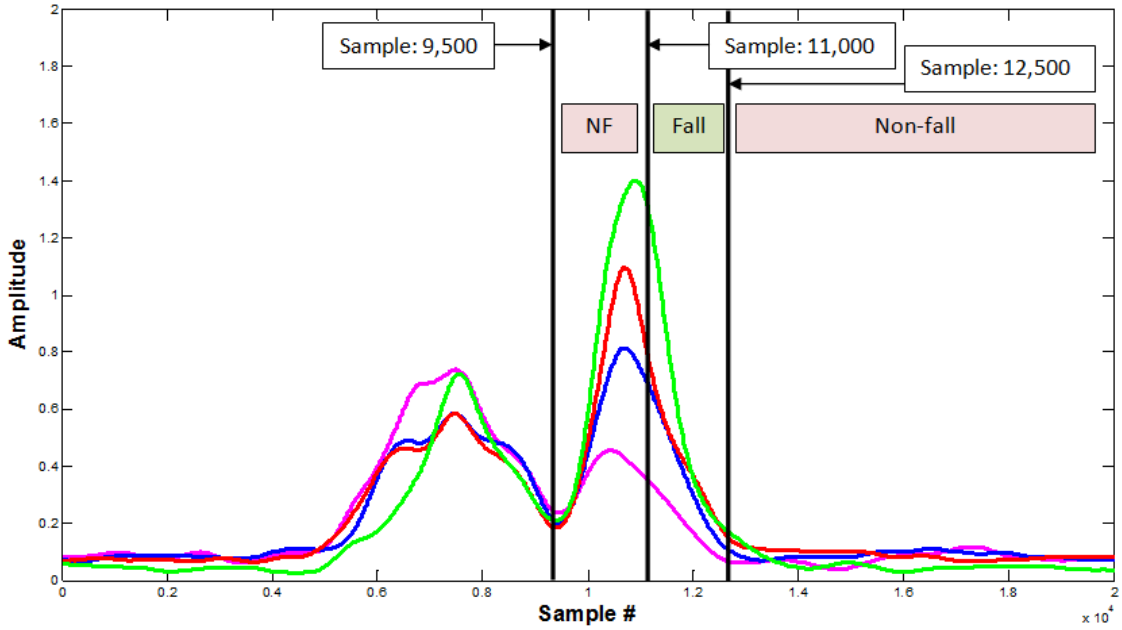

Figure 4.4: Testing data without clapping activity

4.2 Sparsity and Classification Time Results

As discussed in section 2.3, the Parzen Window Classifier uses the entire training data set for classification. This means that the Parzen Window mathematically compares each testing data point to all 6,800 training data points to determine which class it belongs in. Alternatively, the RVM goes through a training process where the least effective training points are pruned. In this research, the training process for the RVM yielded a much more sparse training data set of 1,723. The training data set used by the Parzen Window is nearly 4 times the size the training data set used by the RVM.

As shown in Table 4-1, this enables the RVM to much more quickly classify the 1,341,000 testing data points. The Parzen Window took 8 hours, 55 minutes, and 49 seconds to classify all 38 fall and non-fall data files where the RVM took 2 hours, 27 minutes, and 58 seconds to do the same thing. As shown below, in the next section, the RVM is more accurate in classifying the testing data.

Table 4-1: Classification time

#	File Name	Parzen	RVM	Number of Data Points	Activity Description
1	201004201051.txt	04:08	01:10	10,000	Standing Position Looses Balance Fall Forward
2	201004201053.txt	04:58	01:18	12,000	Standing Position Looses Balance Fall Backward
3	201004201055.txt	07:45	02:03	18,000	Standing Position Looses Balance Fall Backward
4	201004201057.txt	09:05	02:26	22,000	Standing Position Looses Balance Fall Left
5	201004201100.txt	05:21	01:25	13,000	Standing Position Looses Balance Fall Right
6	201004201101.txt	04:57	01:19	12,000	Standing Position Looses Balance Fall Right
7	201004201104.txt	08:12	02:31	20,000	Standing Position Momentary Loss of Consciousness Fall Forward
8	201004201107.txt	05:44	01:31	14,000	Standing Position Momentary Loss of Consciousness Fall Backward
9	201004201109.txt	04:06	01:05	10,000	Standing Position Momentary Loss of Consciousness Fall Left
10	201004201111.txt	06:34	01:43	16,000	Standing Position Momentary Loss of Consciousness Fall Right
11	201004201114.txt	06:01	01:39	15,000	Standing Position Momentary Loss of Consciousness Fall Straight Down
12	201004201138.txt	02:25	00:39	6,000	Tripping and Slipping Trip and Fall Forward
13	201004201139.txt	09:12	02:28	23,000	Tripping and Slipping Trip and Fall Sideways
14	201004201142.txt	10:23	02:47	26,000	Tripping and Slipping Slip and Fall Forward
15	201004201144.txt	11:59	03:14	30,000	Tripping and Slipping Slip and Fall Sideways
16	201004201147.txt	06:49	01:48	17,000	Tripping and Slipping Slip and Fall Backward
17	201004201154.txt	06:48	01:49	17,000	Sitting Falling From a Chair Fall Forward
18	201004201158.txt	08:26	02:17	21,000	Sitting Falling From a Chair Fall Left
19	201004201200.txt	08:26	02:14	21,000	Sitting Falling From a Chair Fall Right
20	201004201203.txt	06:01	01:37	15,000	Sitting Falling From a Chair Sliding Forward Out of Chair
21	201004201206.txt	08:00	02:10	20,000	Sitting Falling From a Chair Sliding Backward Out of Chair
22	201004201210.txt	08:00	02:09	20,000	From Bed or Couch Fall to Side Upper Body Falls First
23	201004201212.txt	05:37	01:42	14,000	From Bed or Couch Fall to Side Hips and Shoulders Fall First
24	201004201310.txt	15:26	04:12	39,000	Standing to Squatting
25	201004201314.txt	24:12	06:34	61,000	Standing to Kneeling
26	201004201315.txt	35:17	09:46	89,000	Standing to Kneeling to Lying on Floor
27	201004201318.txt	16:17	04:35	41,000	Bend Down Plug in Appliance
28	201004201322.txt	21:50	05:59	55,000	Squat to Tie Shoe
29	201004201325.txt	26:40	07:36	67,000	Standing to Sitting Legs Tucked
30	201004201327.txt	29:26	08:01	74,000	Standing to Sitting Legs Extended
31	201004201329.txt	43:16	11:53	109,000	Standing to Situps and Stretches
32	201004201331.txt	30:59	08:51	78,000	Lying to Kneeling to Standing
33	201004201335.txt	35:52	10:38	90,000	Walking Trip But Regain Balance
34	201004201338.txt	12:18	03:35	31,000	Walking Forward and Stop Suddenly
35	201004201340.txt	15:53	04:23	40,000	Walking Forward and Stop Suddenly Turn Around
36	201004201341.txt	20:13	05:30	51,000	Walk to Chair and Sit
37	201004201343.txt	22:39	06:09	57,000	Walk to Chair and Sit Pickup Book
38	201004201344.txt	26:33	07:12	67,000	Walk to Chair and Sit Attempt to Stand
	Total	8:55:49	2:27:58	1,341,000	

4.3 Results Without Filtering Classification Output

As discussed in section 3.8, the post-processing step takes in the classified values for each 50 sample window of data and calculates a mean filter over the classified values to reduce the number of false positives which have a smaller area under the curve than the true positives. To visualize the advantage of calculating the mean filter, it

is helpful to see the confusion matrix data on what the results would be without the mean filter. These results are shown below.

4.3.1 Results Excluding Clapping portion of Fall Data

When the stunt actress data set was collected, each fall or non-fall activity began with the stunt actress bending over to clap near the ground and then performing the activity. This was done to aid the research of a different sensor based on a microphone array. In this research, the clapping portion of the fall activities can be added and removed to demonstrate each classifier's ability to correctly classify data which is a potential false alarm. As data from the clapping portion of the falls are added and removed, it is only added or removed from the testing data set. The training data set does not change. The results of excluding the clapping portion of the fall are analyzed below.

Below in Table 4-2 is the confusion data calculated over a range of threshold values where each data file was analyzed individually and the clapping portion of each data file was removed and not considered. The classification results of each data file containing fall activity, were parsed at each threshold value and if the classification results rose above the threshold value, where a fall was supposed to have happened as indicated by the ground truth, then the data file was considered to be a true positive fall. If no fall was detected, then the data file was counted in the confusion matrix as a false negative. For the non-fall data files, if a fall was detected then the data file was considered to be a false positive and if no fall was detected, then the data file is considered to be a true negative. Since no clap activities are considered, this gives 23

potential true positives (TP) or false negatives (FN), one for each fall activity. There are also 15 potential false positive (FP) or true negative (TN) activities one for each non-fall activity.

Table 4-2:Confusion Table, Parzen Window, 1 Count per Data File, No Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	22	1	15	0	0.96	1.00	57.89
0.1	21	2	14	1	0.91	0.93	57.89
0.2	20	3	14	1	0.87	0.93	55.26
0.3	20	3	13	2	0.87	0.87	57.89
0.4	19	4	11	4	0.83	0.73	60.53
0.5	15	8	9	6	0.65	0.60	55.26
0.6	15	8	8	7	0.65	0.53	57.89
0.7	14	9	7	8	0.61	0.47	57.89
0.8	14	9	6	9	0.61	0.40	60.53
0.9	14	9	3	12	0.61	0.20	68.42
1.0	12	11	1	14	0.52	0.07	68.42
1.1	7	16	0	15	0.30	0.00	57.89

As discussed in section 3.8, choosing the correct threshold means choosing one that will minimize false negatives and false positives which is a balance that should be weighted towards minimizing false negatives because they represent a missed fall. It can be seen in the accuracy column of Table 4-2, that before the classification signal is filtered, and the claps are not considered, the Parzen Window classification accuracy never goes above 68.42% which is a relatively low accuracy to what is seen later in this section. Next, the ROC curve that was generated from the data in Table 4-2 is shown below.

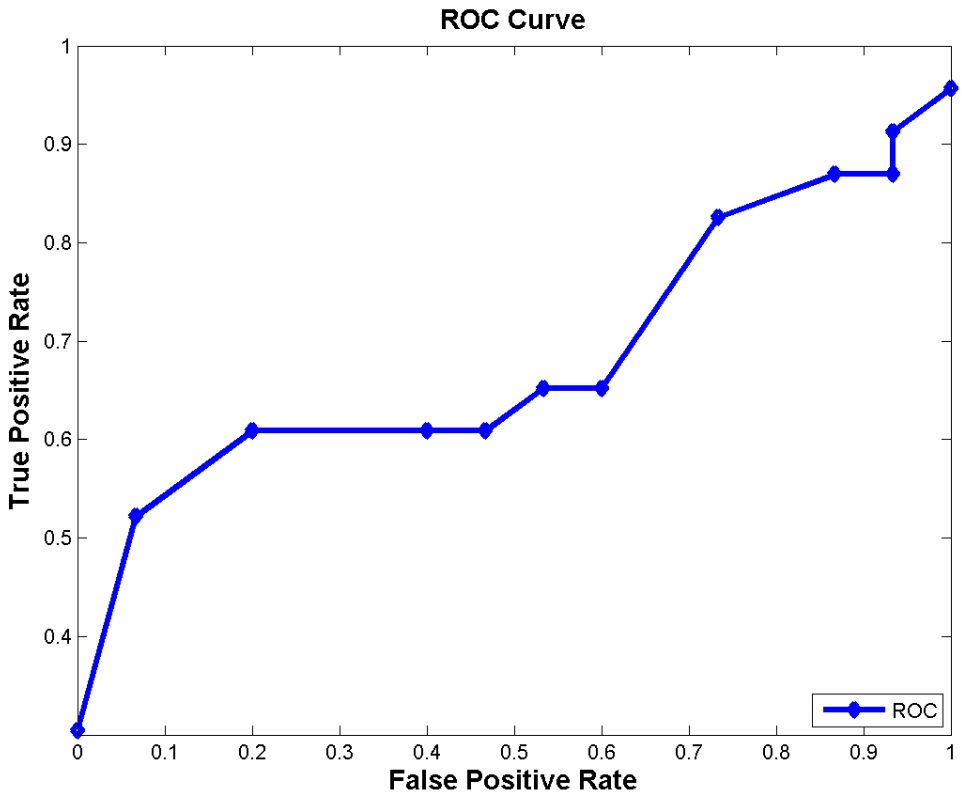

Figure 4.5: ROC Curve, Parzen Window, 1 Count per Data File, No Claps

Table 4-3 below is the confusion table for the same classification as discussed above but with each 50 sample window of classification data being counted in the table. If a window of classification data indicates a fall and it is supposed to be a fall, then that window is considered to be a true positive. If a window is supposed to be classified as a fall and it is not classified as one, then it is considered to be a false negative. If a window of data is classified as a fall and it is supposed to be a non-fall then the false positive count is increased. If a window is classified as a fall and it is supposed to be classified as a fall then that window is considered a true negative.

Table 4-3: Confusion Table, Parzen Window, 1 Count per 50 Sample Window, No Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	573	217	2380	23700	0.73	0.09	90.30
0.1	321	469	347	25700	0.41	0.01	97.00
0.2	248	542	222	25800	0.31	0.01	97.20
0.3	203	587	133	25900	0.26	0.01	97.30
0.4	169	621	85	25900	0.21	0.00	97.40
0.5	141	649	44	26000	0.18	0.00	97.40
0.6	116	674	33	26000	0.15	0.00	97.40
0.7	88	702	21	26000	0.11	0.00	97.30
0.8	49	741	13	26000	0.06	0.00	97.20
0.9	19	771	2	26000	0.02	0.00	97.10
1.0	5	785	0	26000	0.01	0.00	97.10
1.1	2	788	0	26000	0.00	0.00	97.10

Below is the ROC curve that was generated from the data in Table 4-3.

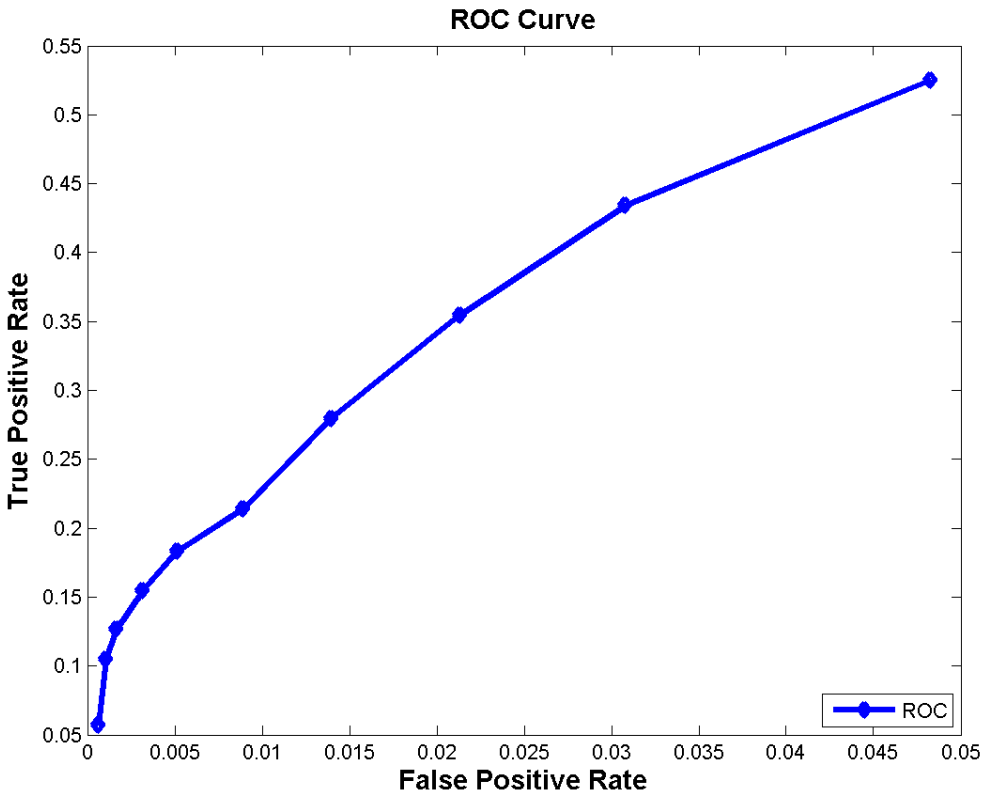

Figure 4.6: ROC Curve, Parzen Window, 1 Count per 50 Sample Window, No Claps

Table 4-4 shows the confusion table for each data file as classified by the RVM, with the clap portion of the fall activities excluded and before the classification values are filtered. Similar to the Parzen Window classification the accuracy does not go above 68.42%.

Table 4-4: Confusion Table, RVM, 1 Count per Data File, No Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	21	2	15	0	0.91	1.00	55.26
0.1	19	4	10	5	0.83	0.67	63.16
0.2	19	4	9	6	0.83	0.60	65.79
0.3	17	6	7	8	0.74	0.47	65.79
0.4	16	7	6	9	0.70	0.40	65.79
0.5	14	9	3	12	0.61	0.20	68.42
0.6	12	11	2	13	0.52	0.13	65.79
0.7	11	12	1	14	0.48	0.07	65.79
0.8	10	13	1	14	0.43	0.07	63.16
0.9	6	17	1	14	0.26	0.07	52.63
1.0	2	21	0	15	0.09	0.00	44.74
1.1	1	22	0	15	0.04	0.00	42.11

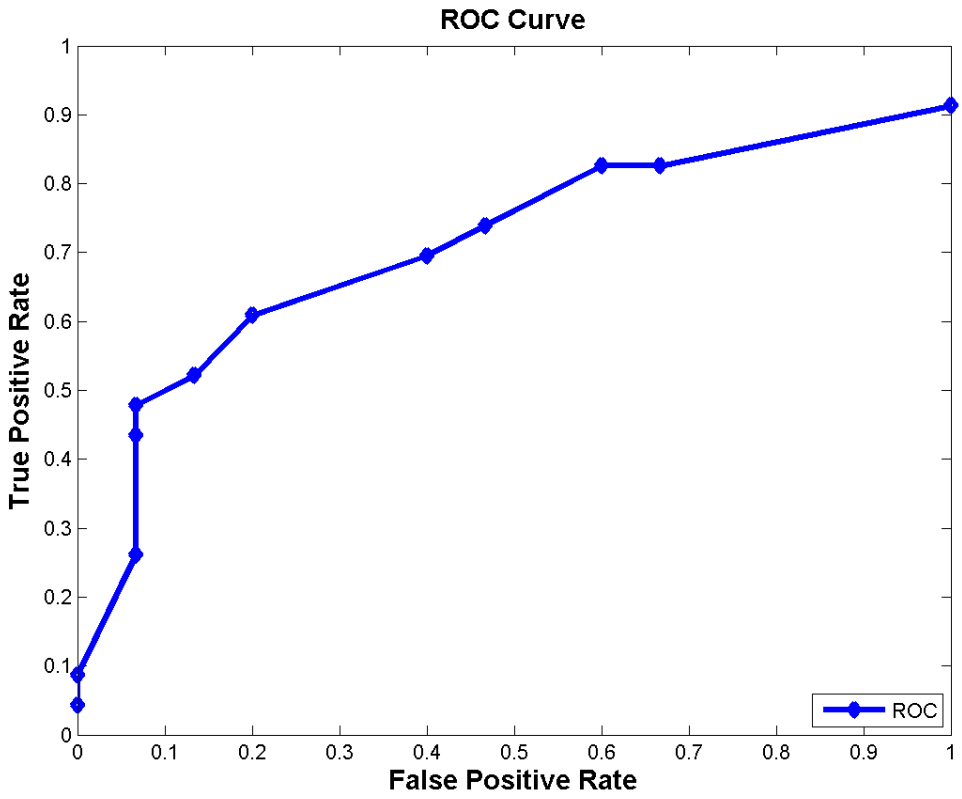

Figure 4.7: ROC curve, RVM, 1 Count per Data File, No Claps

Table 4-5: : Confusion Table , RVM, 1 Count per 50 Sample Window, No Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	573	217	2322	20218	0.73	0.10	89.12
0.1	321	469	333	22207	0.41	0.01	96.56
0.2	248	542	212	22328	0.31	0.01	96.77
0.3	203	587	127	22413	0.26	0.01	96.94
0.4	169	621	80	22460	0.21	0.00	97.00
0.5	141	649	43	22497	0.18	0.00	97.03
0.6	116	674	33	22507	0.15	0.00	96.97
0.7	88	702	21	22519	0.11	0.00	96.90
0.8	49	741	13	22527	0.06	0.00	96.77
0.9	19	771	2	22538	0.02	0.00	96.69
1.0	5	785	0	22540	0.01	0.00	96.64
1.1	2	788	0	22540	0.00	0.00	96.62

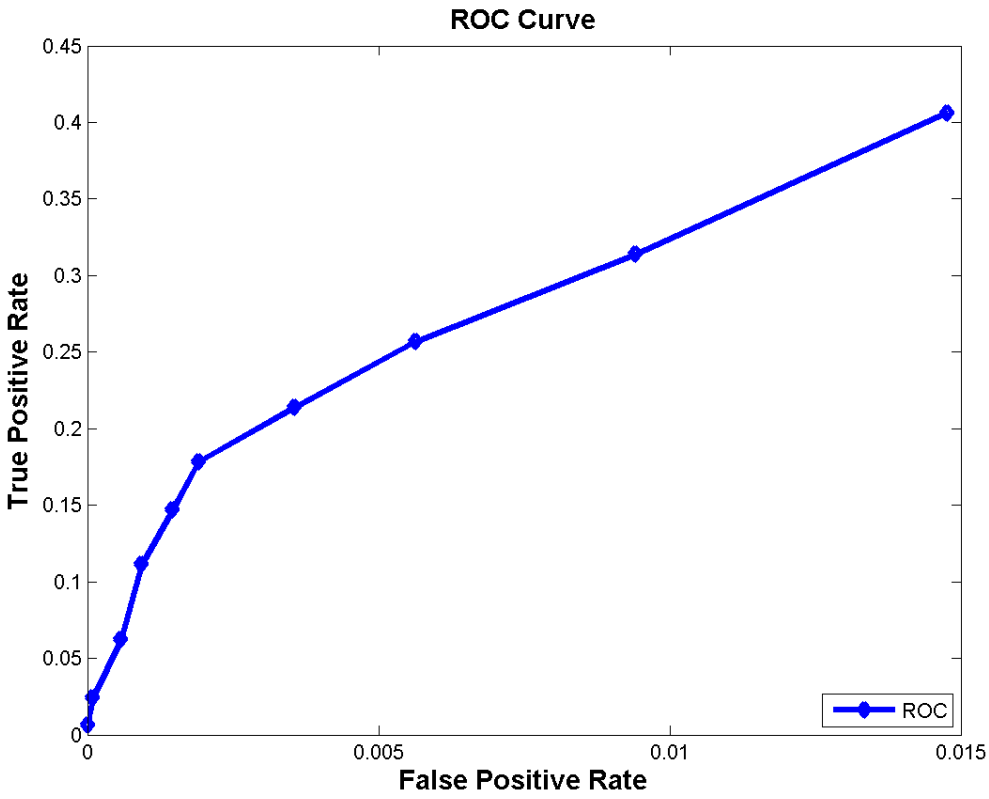

Figure 4.8: ROC Curve, RVM, 1 Count per 50 Sample Window, No Claps

4.3.2 Results Including Clapping portion of Fall Data

In this section, the portion of each fall activity where the stunt actress bends towards the ground and claps is counted as a separate non-fall activity. The addition of clapping activity as a separate non-fall activity is only done in the testing data, and not in the training data. The training data does not change. The results from including the clap activity are improved but are not as good as results seen later in this section.

Table 4-6: Confusion Table, Parzen Window, 1 Count per Data File, Count Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	22	1	36	2	0.96	0.95	39.34
0.1	21	2	32	6	0.91	0.84	44.26
0.2	20	3	24	14	0.87	0.63	55.74
0.3	20	3	22	16	0.87	0.58	59.02
0.4	19	4	19	19	0.83	0.50	62.30
0.5	15	8	15	23	0.65	0.39	62.30
0.6	15	8	13	25	0.65	0.34	65.57
0.7	14	9	11	27	0.61	0.29	67.21
0.8	14	9	9	29	0.61	0.24	70.49
0.9	14	9	4	34	0.61	0.11	78.69
1.0	12	11	2	36	0.52	0.05	78.69
1.1	7	16	0	38	0.30	0.00	73.77

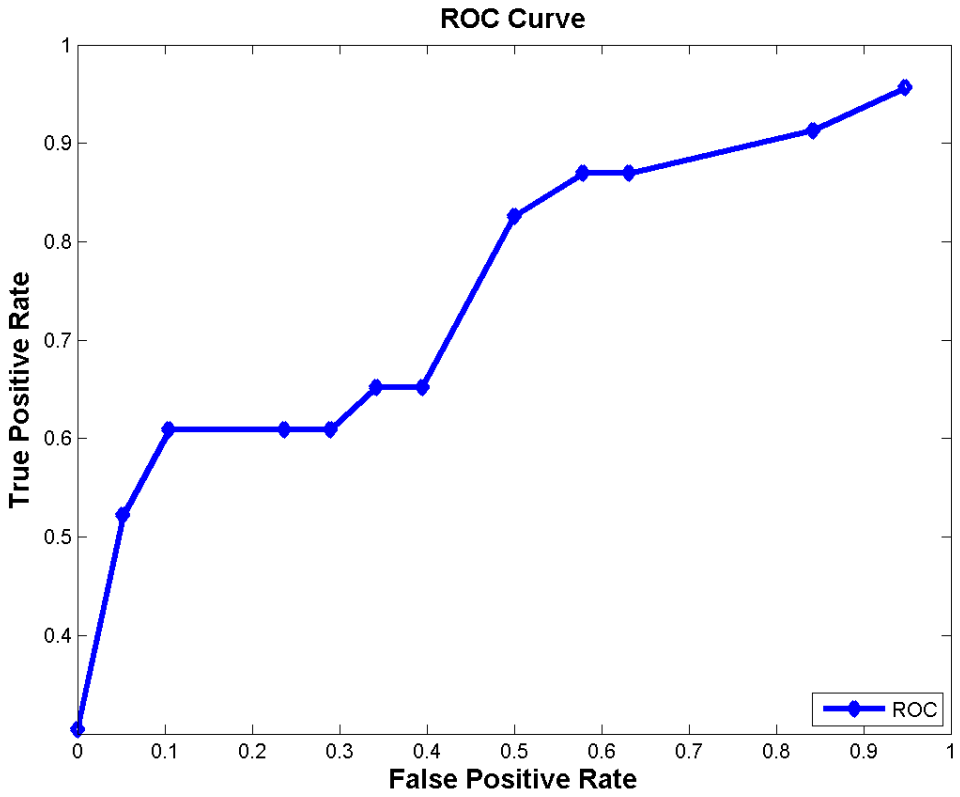

Figure 4.9: ROC Curve, Parzen Window, 1 Count per Data File, Count Claps

Table 4-7: Confusion Table, Parzen Window, 1 Count per 50 Sample Window, Count Claps

		___FN___	___FP___				
Threshold	TP	Missed Fall	False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	662	128	3691	22339	0.84	0.14	85.76
0.1	415	375	1289	24741	0.53	0.05	93.80
0.2	343	447	805	25225	0.43	0.03	95.33
0.3	280	510	563	25467	0.35	0.02	96.00
0.4	221	569	371	25659	0.28	0.01	96.50
0.5	169	621	239	25791	0.21	0.01	96.79
0.6	145	645	142	25888	0.18	0.01	97.07
0.7	122	668	88	25942	0.15	0.00	97.18
0.8	100	690	47	25983	0.13	0.00	97.25
0.9	83	707	30	26000	0.11	0.00	97.25
1.0	45	745	17	26013	0.06	0.00	97.16
1.1	28	762	6	26024	0.04	0.00	97.14

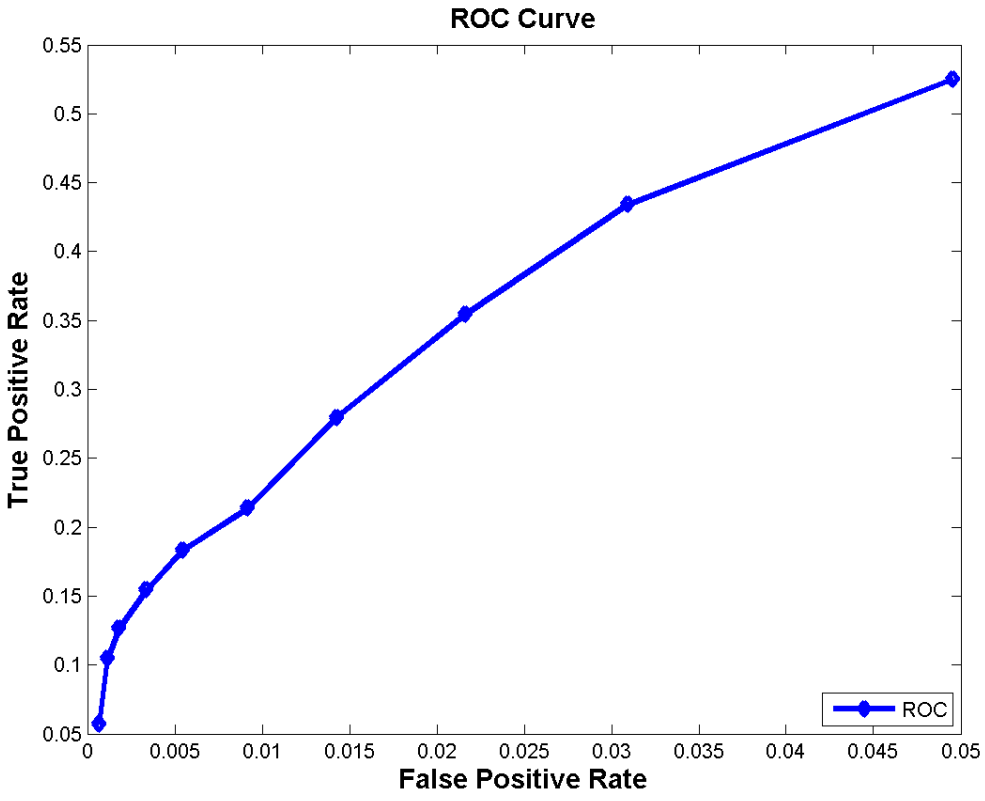

Figure 4.10: ROC Curve, Parzen Window, 1 Count per 50 Sample Window, Count Claps

Table 4-8: Confusion Table, RVM, 1 Count per Data File, Count Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	21	2	28	10	0.91	0.74	50.82
0.1	19	4	15	23	0.83	0.39	68.85
0.2	19	4	12	26	0.83	0.32	73.77
0.3	17	6	9	29	0.74	0.24	75.41
0.4	16	7	8	30	0.70	0.21	75.41
0.5	14	9	4	34	0.61	0.11	78.69
0.6	12	11	2	36	0.52	0.05	78.69
0.7	11	12	1	37	0.48	0.03	78.69
0.8	10	13	1	37	0.43	0.03	77.05
0.9	6	17	1	37	0.26	0.03	70.49
1.0	2	21	0	38	0.09	0.00	65.57
1.1	1	22	0	38	0.04	0.00	63.93

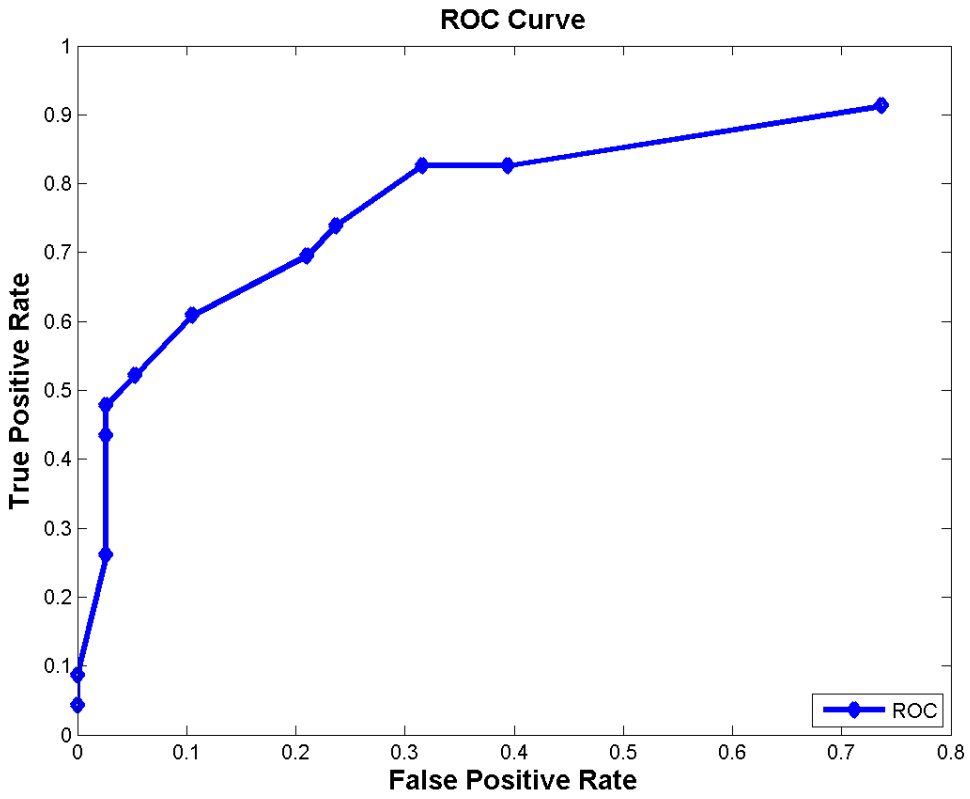

Figure 4.11: ROC Curve, RVM, 1 Count per Data File, Count Claps

Table 4-9: Confusion Table, RVM, 1 Count per 50 Sample Window, Count Claps

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	573	217	2380	23650	0.73	0.09	90.32
0.1	321	469	347	25683	0.41	0.01	96.96
0.2	248	542	222	25808	0.31	0.01	97.15
0.3	203	587	133	25897	0.26	0.01	97.32
0.4	169	621	85	25945	0.21	0.00	97.37
0.5	141	649	44	25986	0.18	0.00	97.42
0.6	116	674	33	25997	0.15	0.00	97.36
0.7	88	702	21	26009	0.11	0.00	97.30
0.8	49	741	13	26017	0.06	0.00	97.19
0.9	19	771	2	26028	0.02	0.00	97.12
1.0	5	785	0	26030	0.01	0.00	97.07
1.1	2	788	0	26030	0.00	0.00	97.06

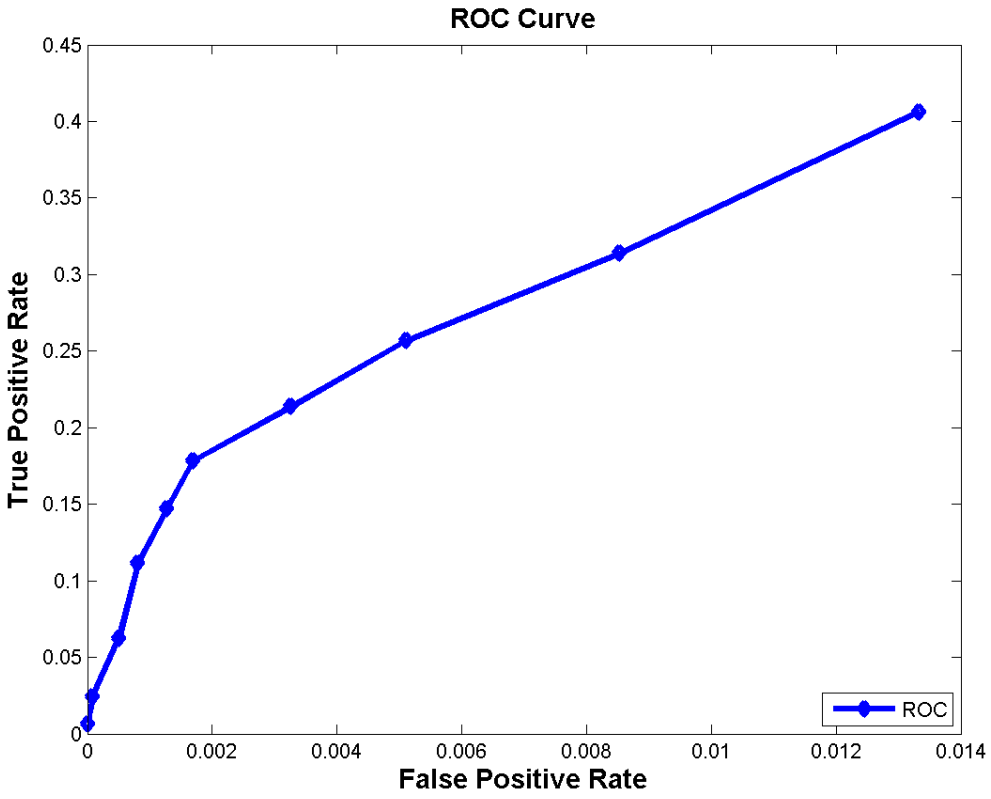

Figure 4.12: ROC Curve, RVM, 1 Count per 50 Sample Window, Count Claps

4.4 Results From Filtering Classification Output

The results below are generated from classification values that have been mean filtered as discussed in section 3.8. As can be seen in the confusion tables and ROC curves below, the results from filtering the classification are considerably better.

4.4.1 Results Excluding Clapping portion of Fall Data

The filtered classification values are considered with and without the clapping activity. The filtered classification values are first considered without the clapping activity below.

Table 4-10: Confusion Table, Parzen Window, 1 Count per Data File, No Claps, filtered

		___FN___	___FP___				
Threshold	TP	Missed Fall	False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	22	1	15	0	0.96	1.00	57.89
0.1	20	3	12	3	0.87	0.80	60.53
0.2	20	3	9	6	0.87	0.60	68.42
0.3	18	5	6	9	0.78	0.40	71.05
0.4	16	7	5	10	0.70	0.33	68.42
0.5	14	9	1	14	0.61	0.07	73.68
0.6	9	14	1	14	0.39	0.07	60.53
0.7	5	18	0	15	0.22	0.00	52.63
0.8	3	20	0	15	0.13	0.00	47.37
0.9	0	23	0	15	0.00	0.00	39.47
1.0	0	23	0	15	0.00	0.00	39.47
1.1	0	23	0	15	0.00	0.00	39.47

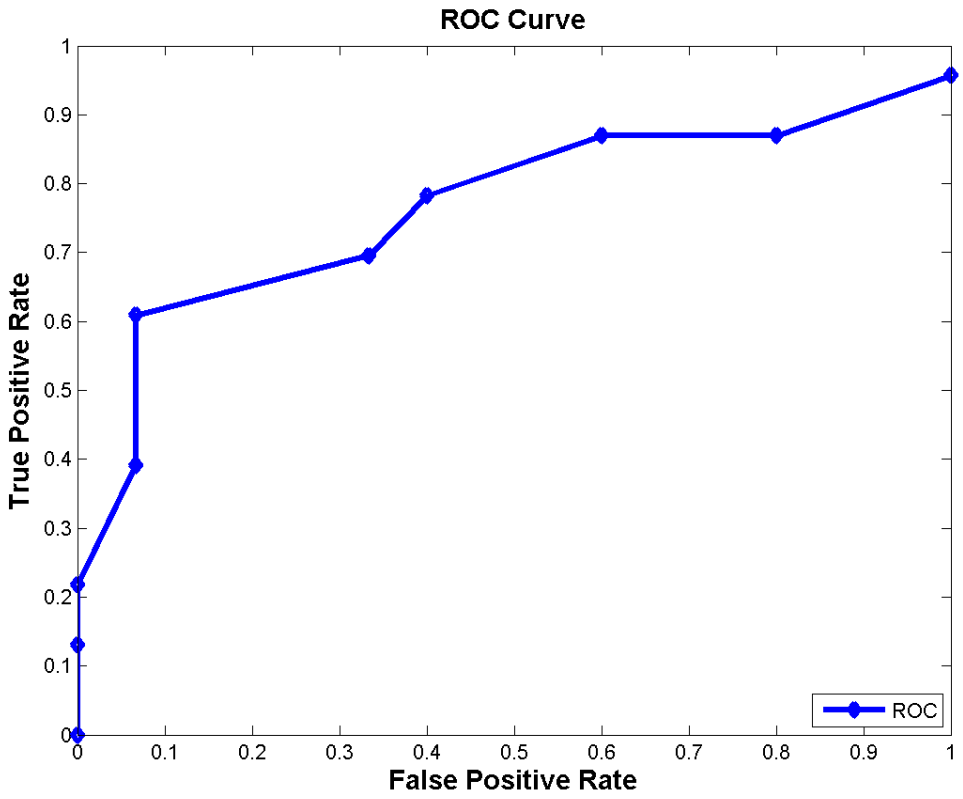

Figure 4.13: ROC Curve, Parzen Window, 1 Count per Data File, No Claps, filtered

Table 4-11: Confusion Table, Parzen Window, 1 Count per 50 Sample Window, No Claps, filtered

		___FN___	___FP___				
Threshold	TP	Missed Fall	False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	770	20	21603	937	0.97	0.96	7.32
0.1	530	260	1305	21235	0.67	0.06	93.29
0.2	409	381	636	21904	0.52	0.03	95.64
0.3	302	488	300	22240	0.38	0.01	96.62
0.4	215	575	98	22442	0.27	0.00	97.12
0.5	151	639	20	22520	0.19	0.00	97.18
0.6	80	710	13	22527	0.10	0.00	96.90
0.7	35	755	2	22538	0.04	0.00	96.76
0.8	12	778	0	22540	0.02	0.00	96.67
0.9	0	790	0	22540	0.00	0.00	96.61
1.0	0	790	0	22540	0.00	0.00	96.61
1.1	0	790	0	22540	0.00	0.00	96.61

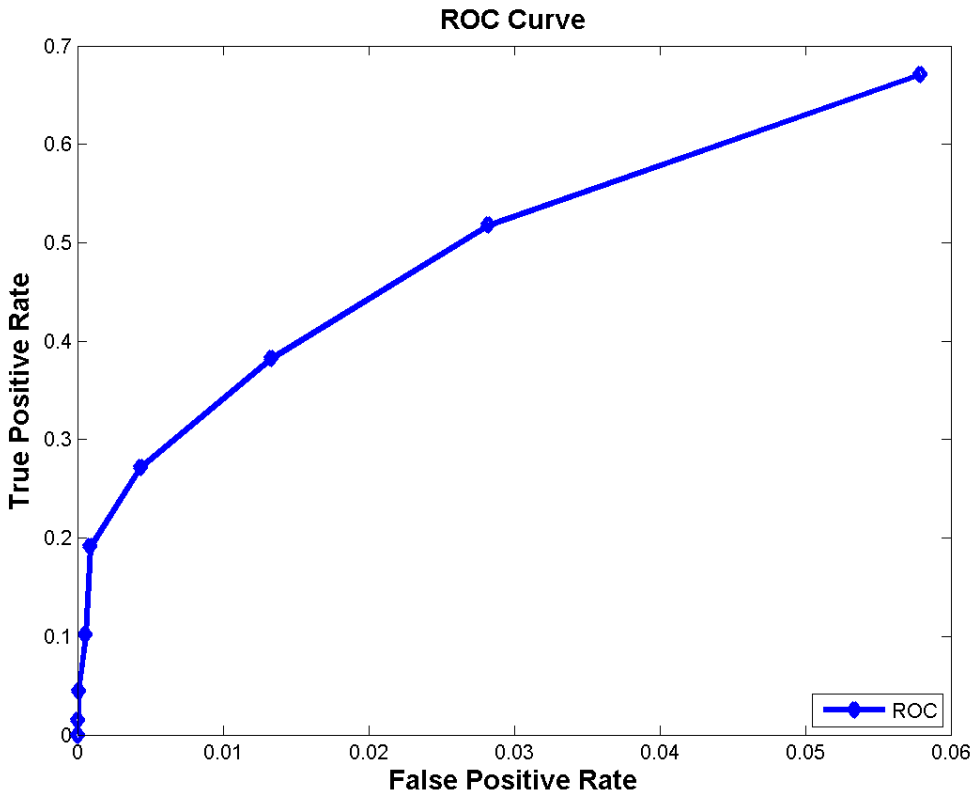

Figure 4.14: ROC Curve, Parzen Window, 1 Count per 50 Sample Window, No Claps, filtered

Table 4-12: Confusion Table, RVM, 1 Count per Data File, No Claps, filtered

		___FN___	___FP___				
Threshold	TP	Missed Fall	False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	23	0	15	0	1.00	1.00	60.53
0.1	18	5	6	9	0.78	0.40	71.05
0.2	14	9	2	13	0.61	0.13	71.05
0.3	10	13	1	14	0.43	0.07	63.16
0.4	9	14	1	14	0.39	0.07	60.53
0.5	8	15	1	14	0.35	0.07	57.89
0.6	3	20	1	14	0.13	0.07	44.74
0.7	1	22	0	15	0.04	0.00	42.11
0.8	1	22	0	15	0.04	0.00	42.11
0.9	0	23	0	15	0.00	0.00	39.47
1.0	0	23	0	15	0.00	0.00	39.47
1.1	0	23	0	15	0.00	0.00	39.47

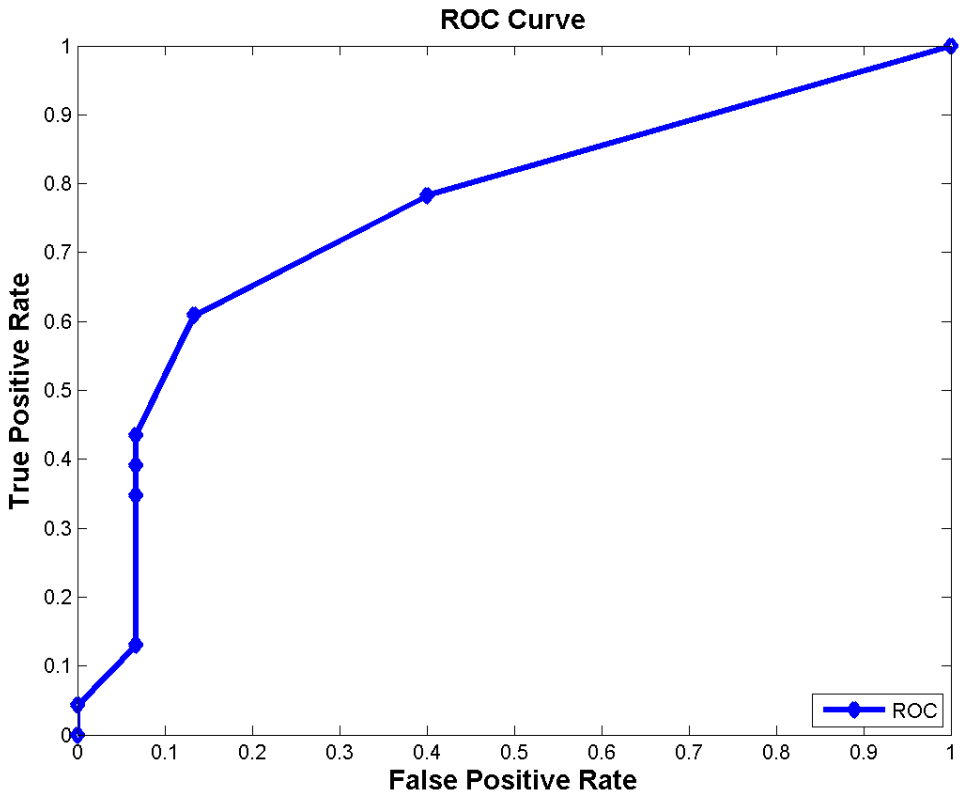

Figure 4.15: ROC Curve, RVM, 1 Count per Data File, No Claps, filtered

Table 4-13: Confusion Table, RVM, 1 Count per 50 Sample Window, No Claps, filtered

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	790	0	21481	1059	1.00	0.95	7.93
0.1	408	382	370	22170	0.52	0.02	96.78
0.2	269	521	123	22417	0.34	0.01	97.24
0.3	180	610	40	22500	0.23	0.00	97.21
0.4	139	651	21	22519	0.18	0.00	97.12
0.5	94	696	14	22526	0.12	0.00	96.96
0.6	37	753	8	22532	0.05	0.00	96.74
0.7	15	775	0	22540	0.02	0.00	96.68
0.8	8	782	0	22540	0.01	0.00	96.65
0.9	0	790	0	22540	0.00	0.00	96.61
1.0	0	790	0	22540	0.00	0.00	96.61
1.1	0	790	0	22540	0.00	0.00	96.61

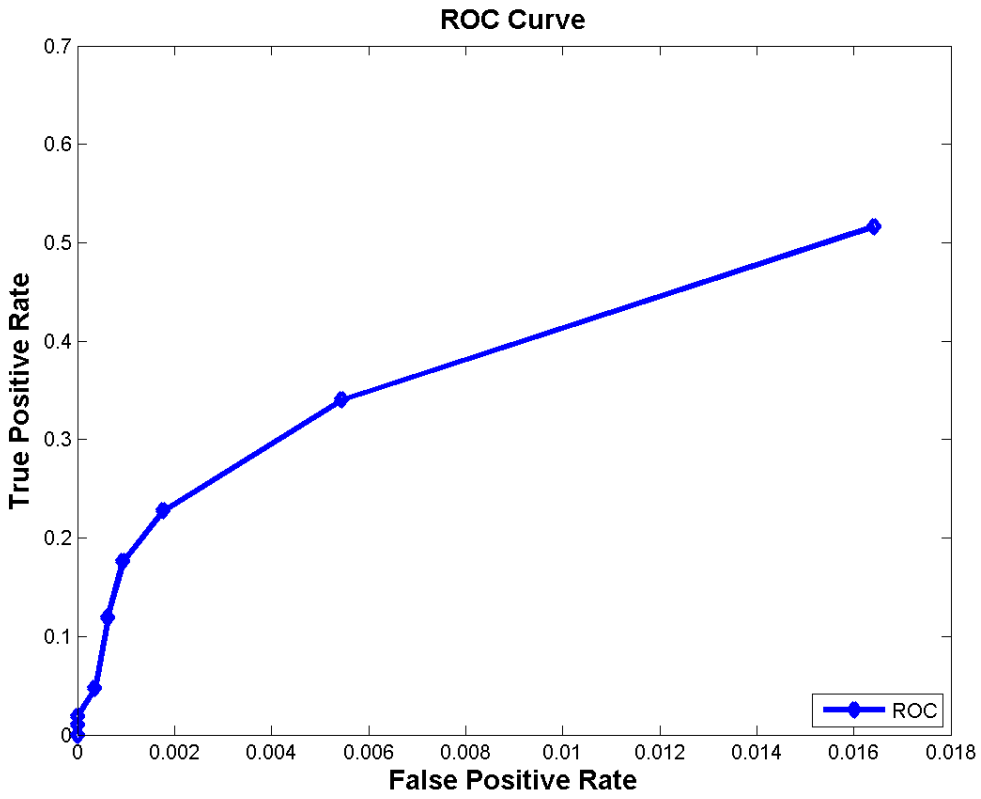

Figure 4.16: ROC Curve, RVM, 1 Count per 50 Sample Window, No Claps, filtered

4.4.2 Results Including Clapping portion of Fall Data

In this section, the final results from filtering the classification data and including the clapping portion of the stunt actress data, are shown. In the Accuracy column of the confusion table, it can be seen below that doing this provides the best results.

Table 4-14: Confusion Table, Parzen Window, 1 Count per Data File, Count Claps, filtered

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	22	1	36	2	0.96	0.95	39.34
0.1	20	3	22	16	0.87	0.58	59.02
0.2	20	3	16	22	0.87	0.42	68.85
0.3	18	5	10	28	0.78	0.26	75.41
0.4	16	7	7	31	0.70	0.18	77.05
0.5	14	9	2	36	0.61	0.05	81.97
0.6	9	14	1	37	0.39	0.03	75.41
0.7	5	18	0	38	0.22	0.00	70.49
0.8	3	20	0	38	0.13	0.00	67.21
0.9	0	23	0	38	0.00	0.00	62.30
1.0	0	23	0	38	0.00	0.00	62.30
1.1	0	23	0	38	0.00	0.00	62.30

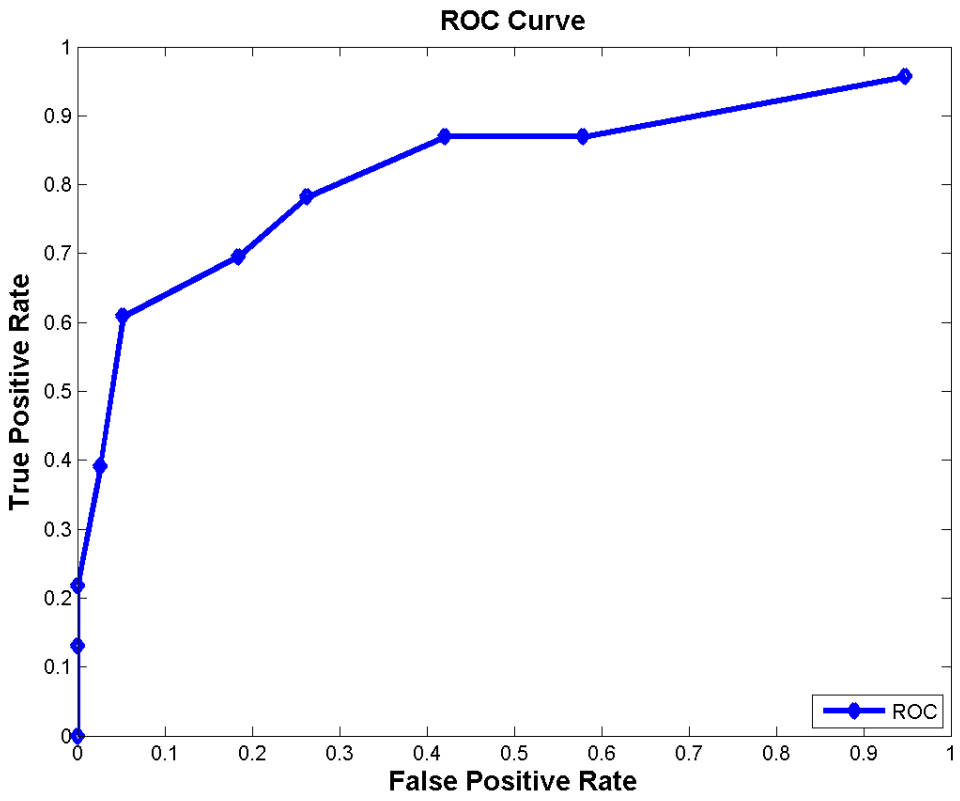

Figure 4.17: ROC Curve, Parzen Window, 1 Count per Data File, Count Claps, filtered

Table 4-15: Parzen Window, 1 Count per 50 Sample Window, Count Claps, filtered

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	770	20	22953	3077	0.97	0.88	14.34
0.1	530	260	1541	24489	0.67	0.06	93.28
0.2	409	381	740	25290	0.52	0.03	95.82
0.3	302	488	347	25683	0.38	0.01	96.89
0.4	215	575	112	25918	0.27	0.00	97.44
0.5	151	639	28	26002	0.19	0.00	97.51
0.6	80	710	13	26017	0.10	0.00	97.30
0.7	35	755	2	26028	0.04	0.00	97.18
0.8	12	778	0	26030	0.02	0.00	97.10
0.9	0	790	0	26030	0.00	0.00	97.05
1.0	0	790	0	26030	0.00	0.00	97.05
1.1	0	790	0	26030	0.00	0.00	97.05

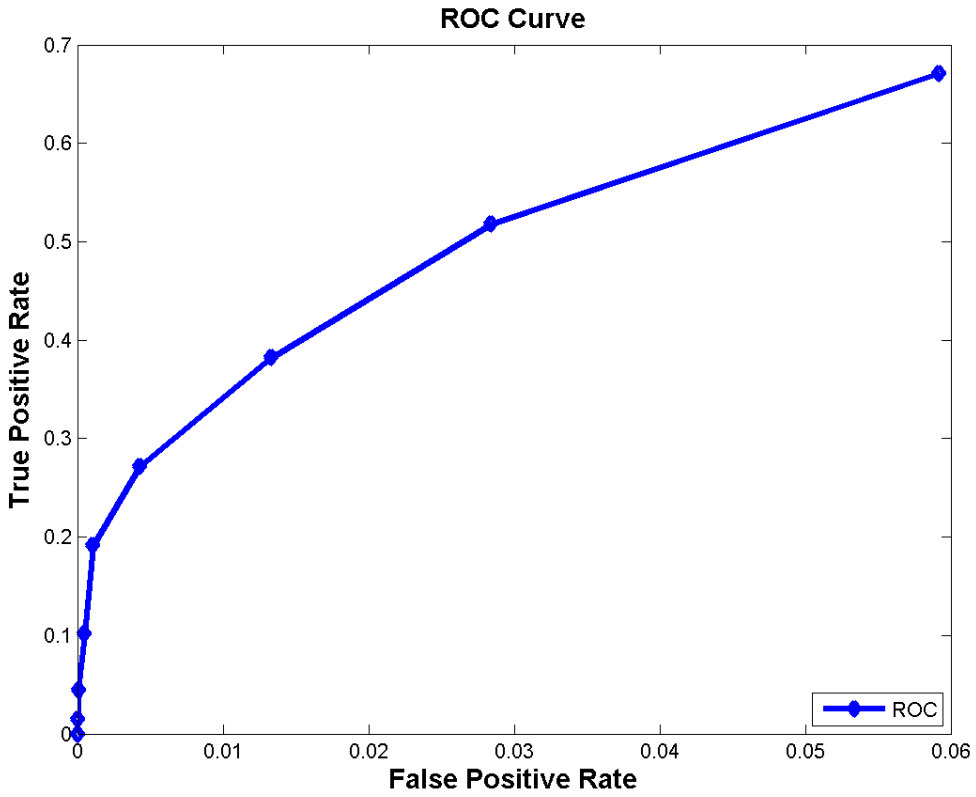

Figure 4.18: ROC Curve, Parzen Window, 1 Count per 50 Sample Window, Count Claps, filtered

Table 4-16: Confusion Table, RVM, 1 Count per Data File, Count Claps, filtered

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	23	0	29	9	1.00	0.76	52.46
0.1	18	5	7	31	0.78	0.18	80.33
0.2	14	9	2	36	0.61	0.05	81.97
0.3	10	13	1	37	0.43	0.03	77.05
0.4	9	14	1	37	0.39	0.03	75.41
0.5	8	15	1	37	0.35	0.03	73.77
0.6	3	20	1	37	0.13	0.03	65.57
0.7	1	22	0	38	0.04	0.00	63.93
0.8	1	22	0	38	0.04	0.00	63.93
0.9	0	23	0	38	0.00	0.00	62.30
1.0	0	23	0	38	0.00	0.00	62.30
1.1	0	23	0	38	0.00	0.00	62.30

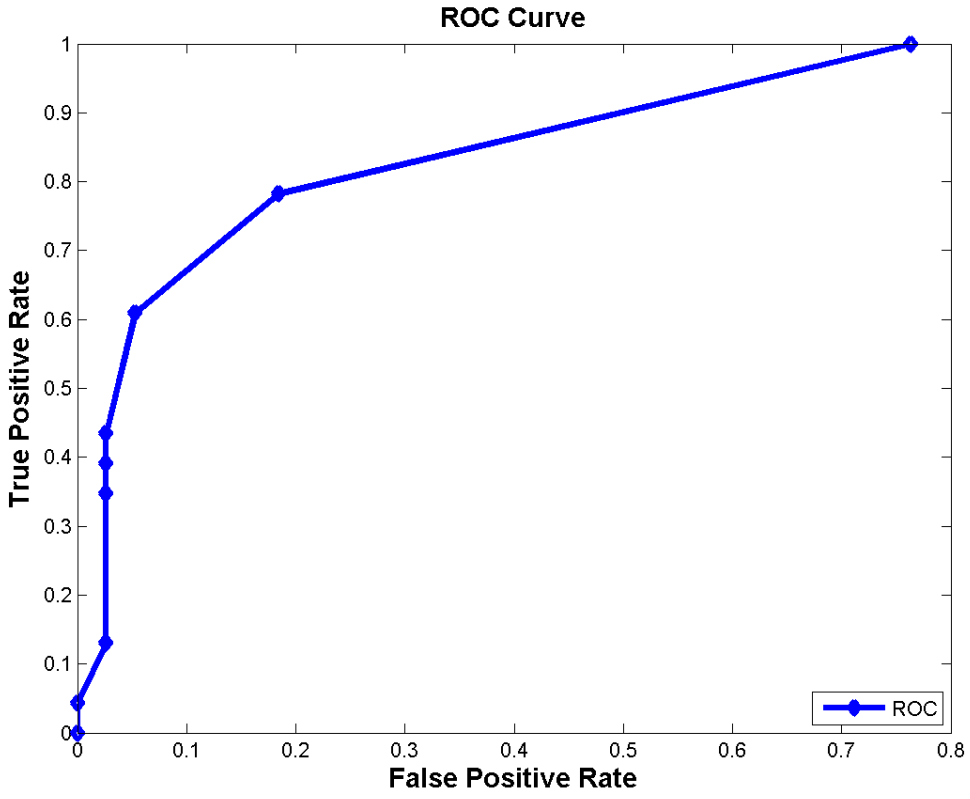

Figure 4.19: ROC Curve, RVM, 1 Count per Data File, Count Claps, filtered

Table 4-17: Confusion Table, RVM, 1 Count per 50 Sample Window, Count Claps, filtered

Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
0.0	790	0	22207	3823	1.00	0.85	17.20
0.1	408	382	382	25648	0.52	0.01	97.15
0.2	269	521	123	25907	0.34	0.00	97.60
0.3	180	610	40	25990	0.23	0.00	97.58
0.4	139	651	21	26009	0.18	0.00	97.49
0.5	94	696	14	26016	0.12	0.00	97.35
0.6	37	753	8	26022	0.05	0.00	97.16
0.7	15	775	0	26030	0.02	0.00	97.11
0.8	8	782	0	26030	0.01	0.00	97.08
0.9	0	790	0	26030	0.00	0.00	97.05
1.0	0	790	0	26030	0.00	0.00	97.05
1.1	0	790	0	26030	0.00	0.00	97.05

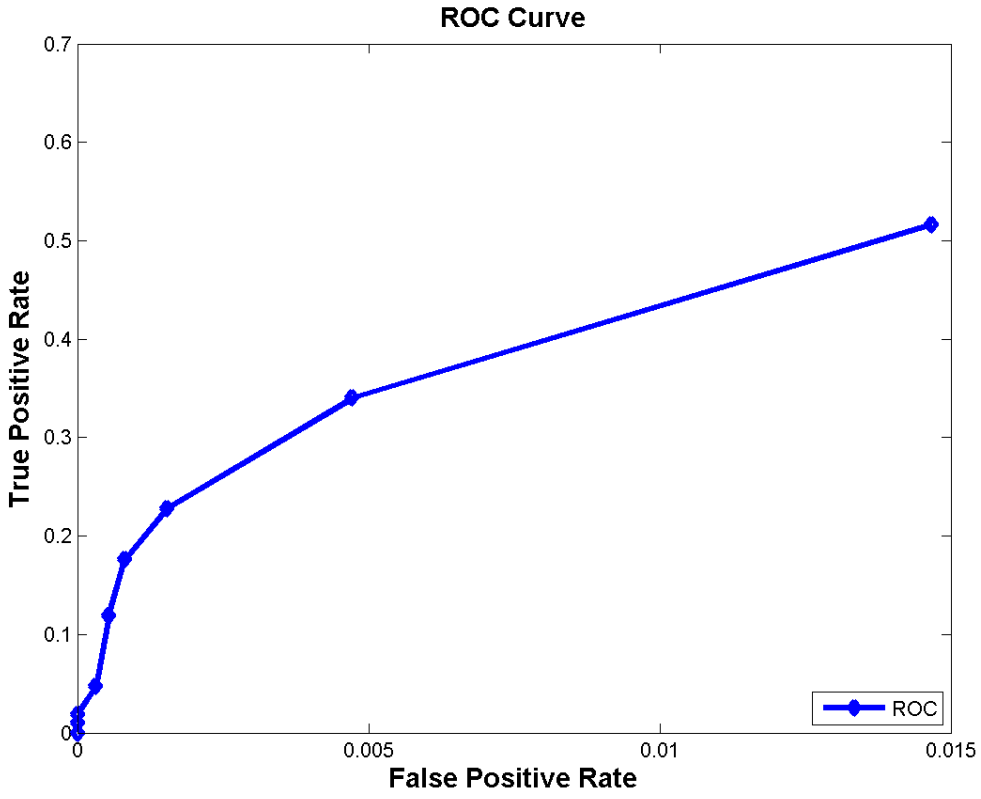

Figure 4.20: ROC Curve RVM, 1 Count per 50 Sample Window, Count Claps, filtered

4.5 Fall Detection Results

After all of the data have been post-processed, it can be determined whether each fall would have been detected or not if the sensing array were placed in a person's home. The fall activities captured in the data files that were processed in this research were detected with varying success. These results are shown below.

4.5.1 Successful Fall Detection

The next two figures show the post-processed classification results from the Parzen Window and RVM for a fall where the stunt actress re-created a fall of a person standing in place then falling. Both classifiers were successful in detecting the fall activity.

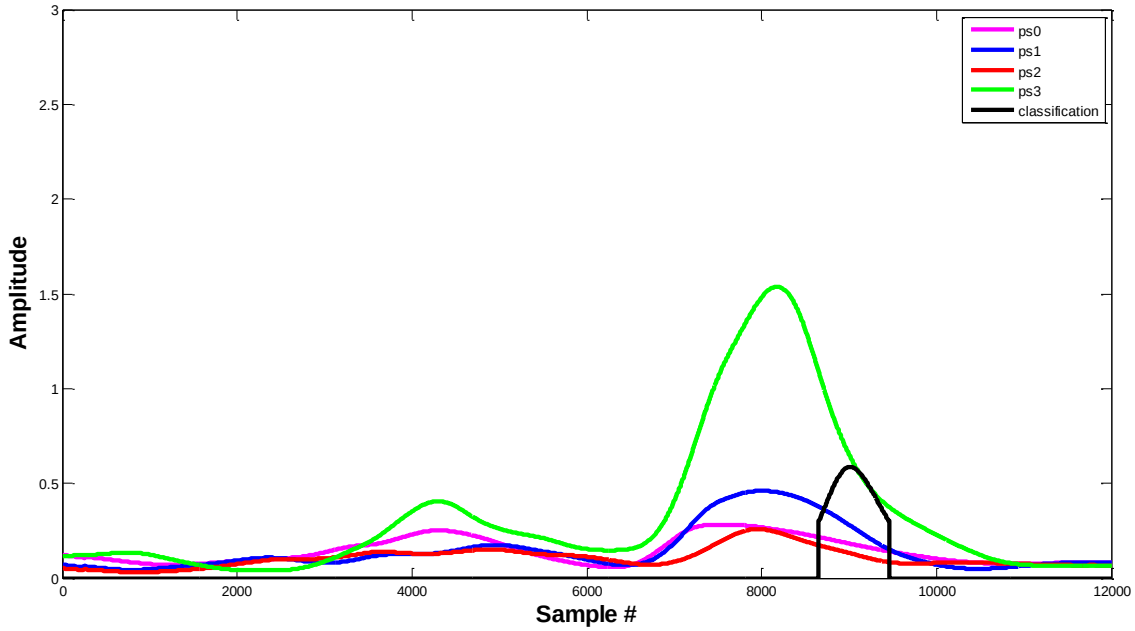

Figure 4.21: Post-processed Classification Parzen 201004201053 Standing Position Looses Balance Fall Backward

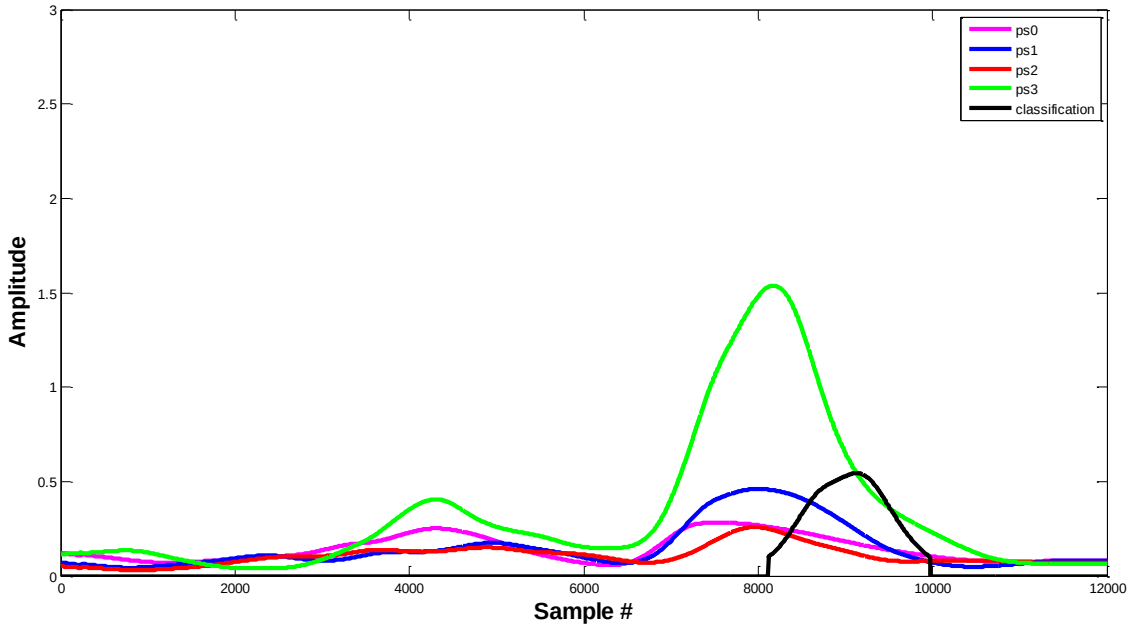

Figure 4.22: Post-processed Classification RVM 201004201053 Standing Position Looses Balance Fall Backward

The data for the next two falls come from the stunt actress depicting a person who is sitting and slides from the chair they were sitting in. Both the RVM and Parzen Window were able to detect this fall activity.

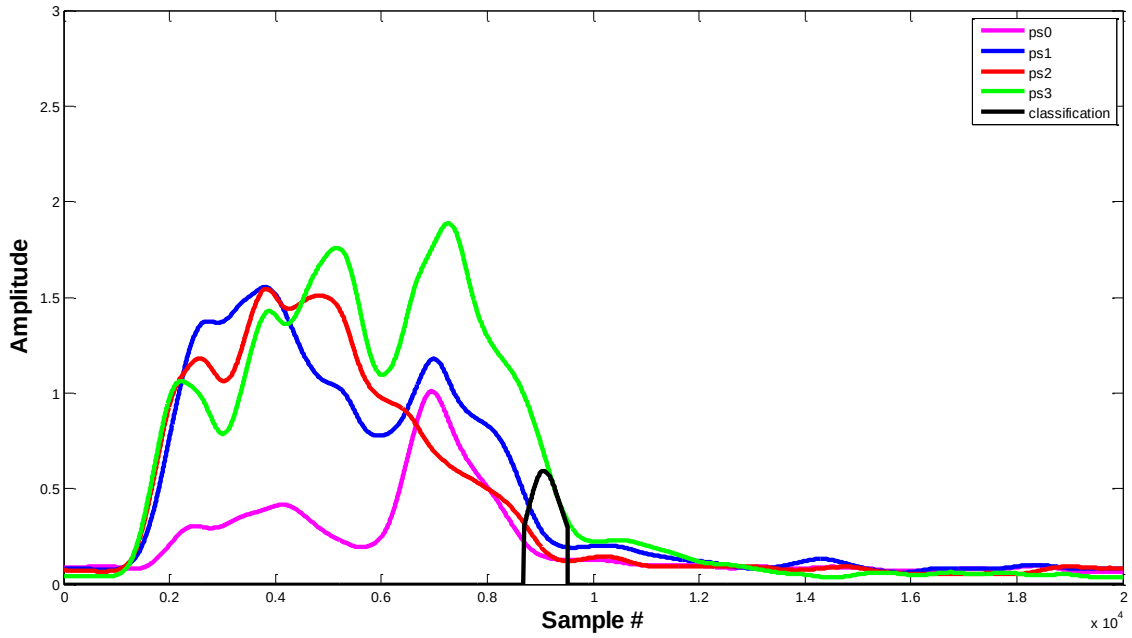

Figure 4.23: Post-processed Classification Parzen 201004201206 Sitting Falling From a Chair Sliding Backward Out of Chair

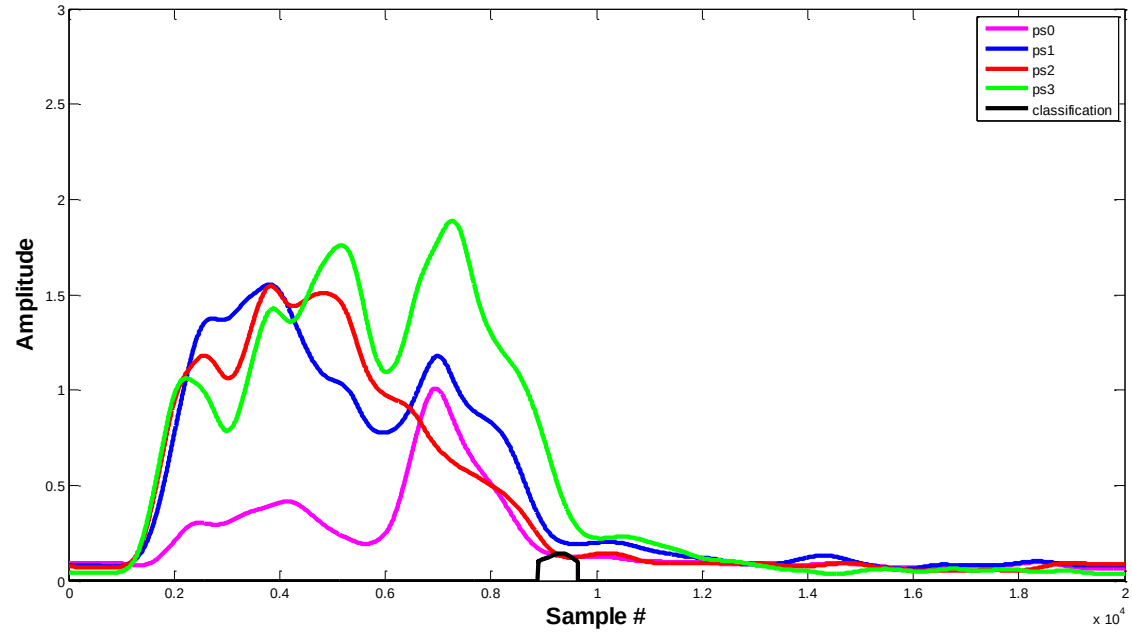

Figure 4.24: Post-processed Classification RVM 201004201206 Sitting Falling From a Chair Sliding Backward Out of Chair

The next four figures show fall activity where the stunt actress falls from a bed or couch. All four of these were detected by each of the classifiers.

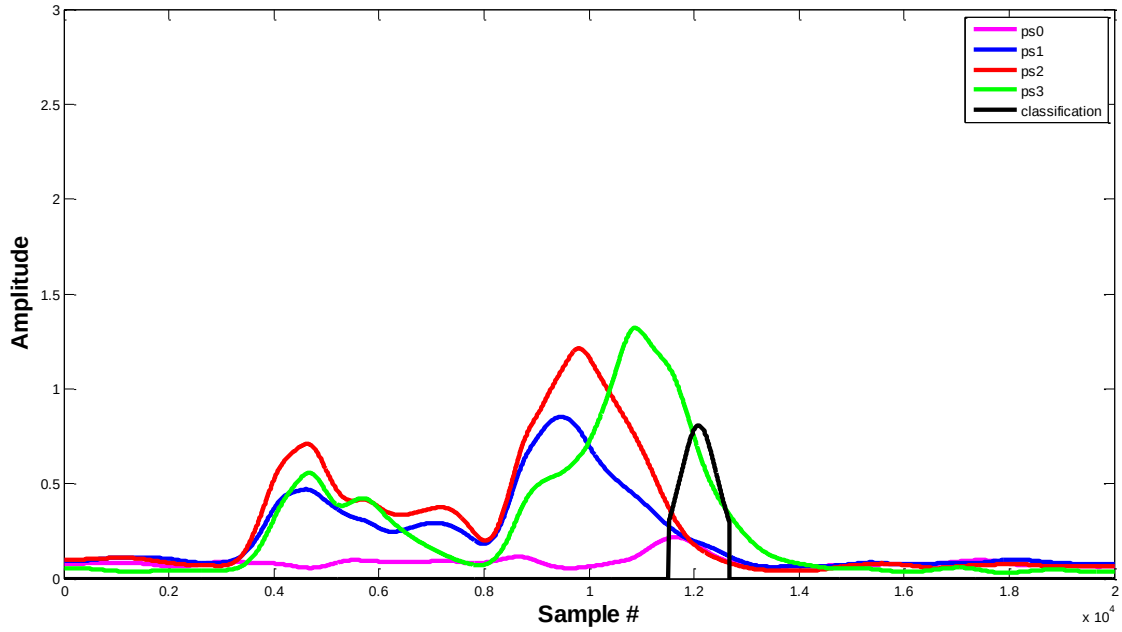

Figure 4.25: Post-processed Classification Parzen 201004201210 From Bed or Couch Fall to Side Upper Body Falls First

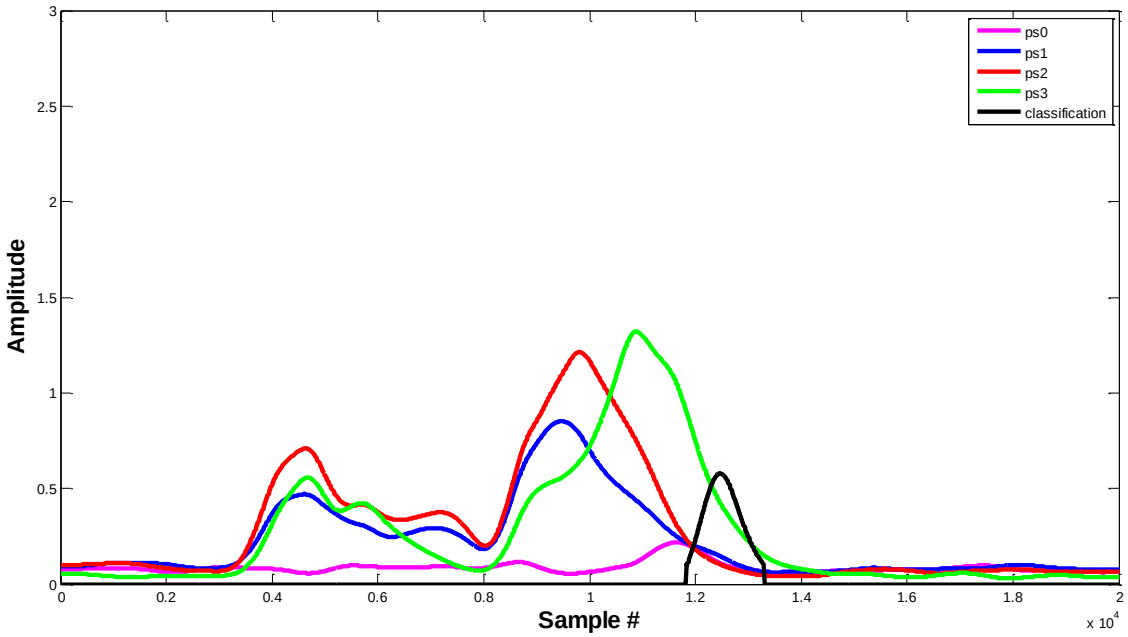

Figure 4.26: Post-processed Classification RVM 201004201210 From Bed or Couch Fall to Side Upper Body Falls First

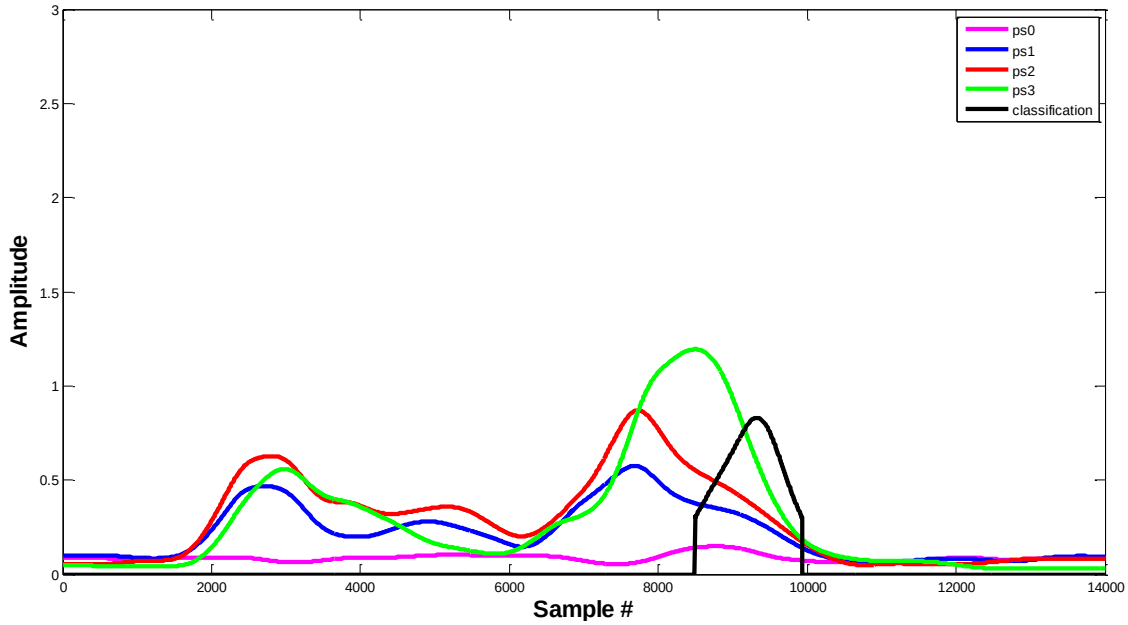

Figure 4.27: Post-processed Classification Parzen 201004201212 From Bed or Couch Fall to Side Hips and Shoulders Fall First

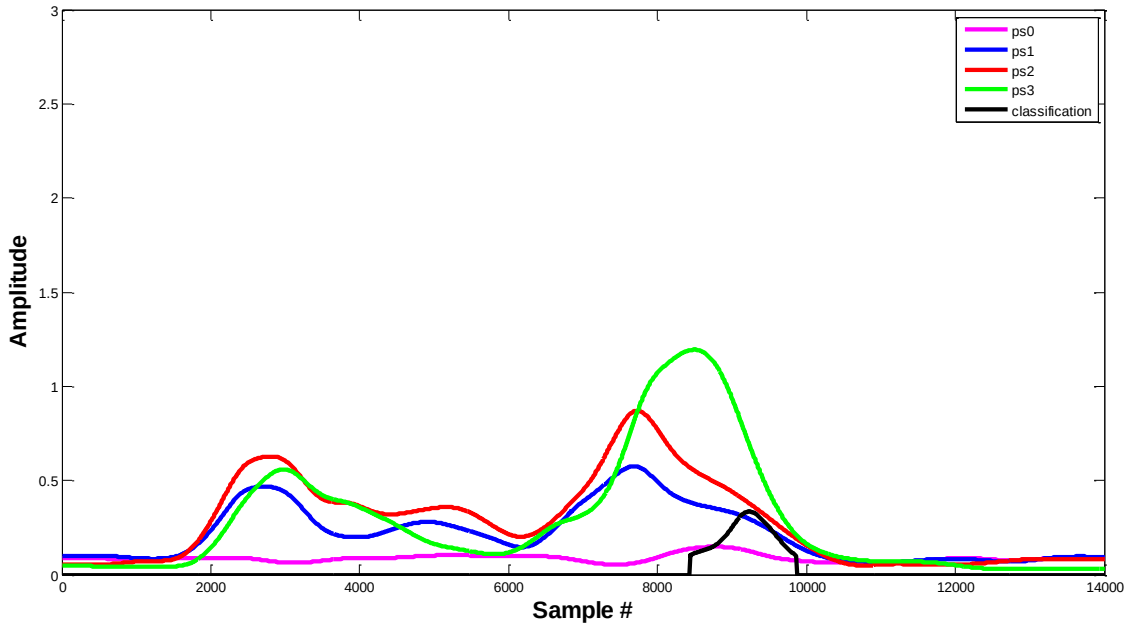

Figure 4.28: Post-processed Classification RVM 201004201212 From Bed or Couch Fall to Side Hips and Shoulders Fall First

Figure 4.21 - Figure 4.28 demonstrate that the sensing array is capable of detecting falls whether a person is standing, sitting, or lying down.

The next two figures are from the false positive protocol where the stunt actress walks to a chair and sits. These figures show that the sensing array is capable of discerning whether a person is sitting in a chair or falling out of it.

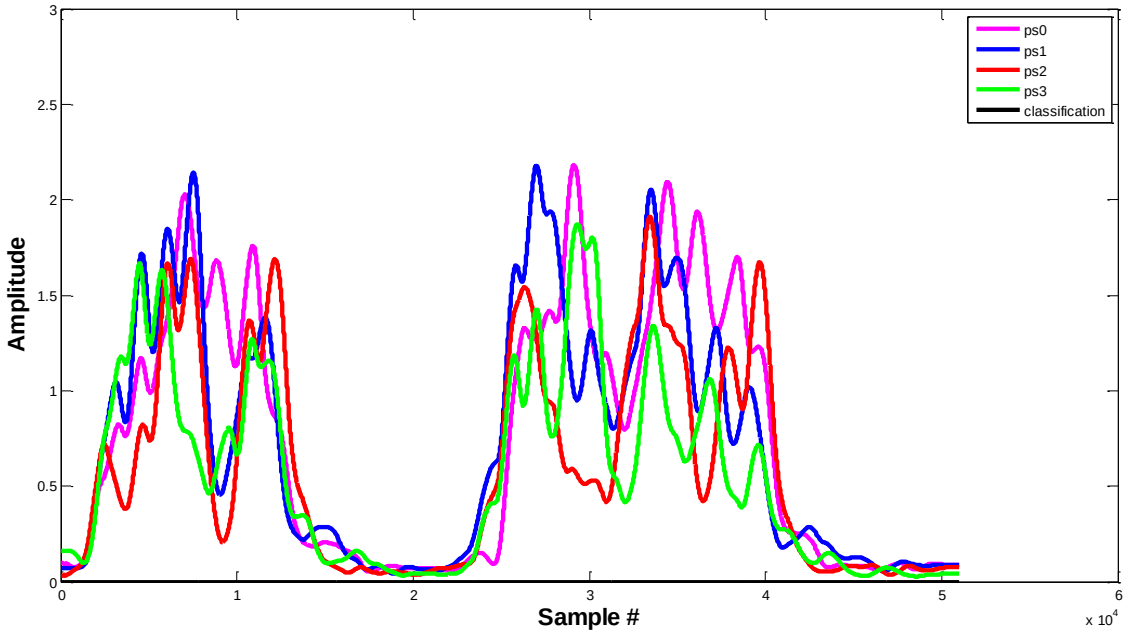

Figure 4.29: Post-processed Classification Parzen 201004201341 False positive 13 Walk to Chair and Sit

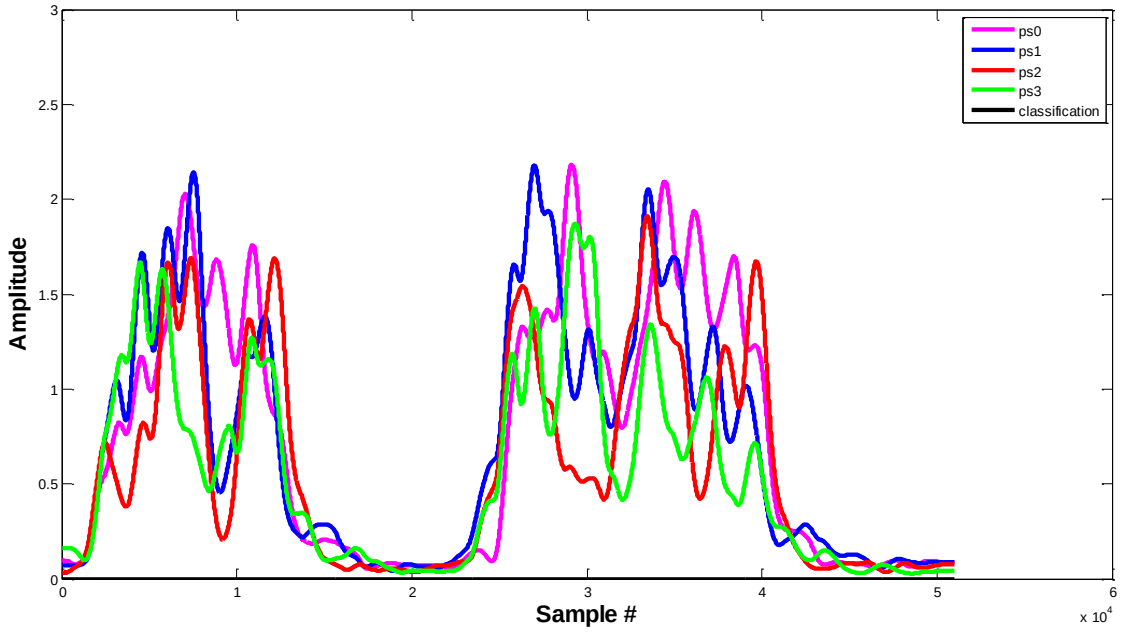

Figure 4.30: Post-processed Classification RVM 201004201341 False positive 13 Walk to Chair and Sit

4.5.2 Failed Fall Detection

The figures in this section show where each of the classifiers either fail to detect a fall, or incorrectly detect a fall for a non-fall activity. The next two figures show fall activity where the Parzen Window was able to detect the fall activity and the RVM failed to detect the fall activity. This fall activity seems like it would be fairly strait forward for the classifiers to detect since the slope resembles a fall. However, the signals are not losing motion in order.

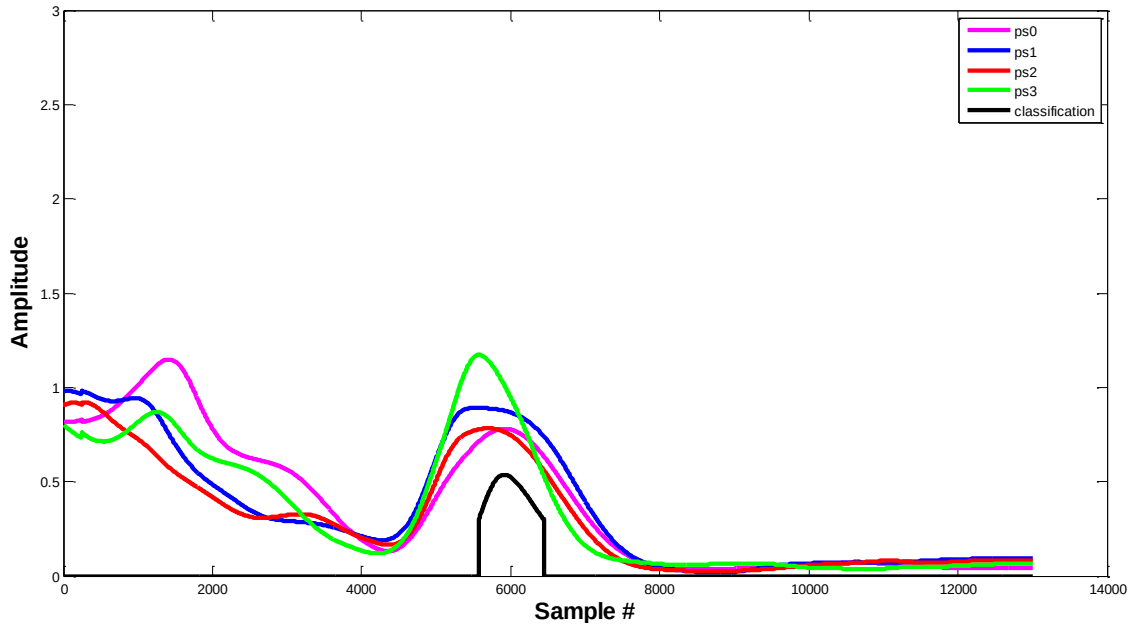

Figure 4.31: Post-processed Classification Parzen 201004201100 Standing Position Looses Balance Fall Right

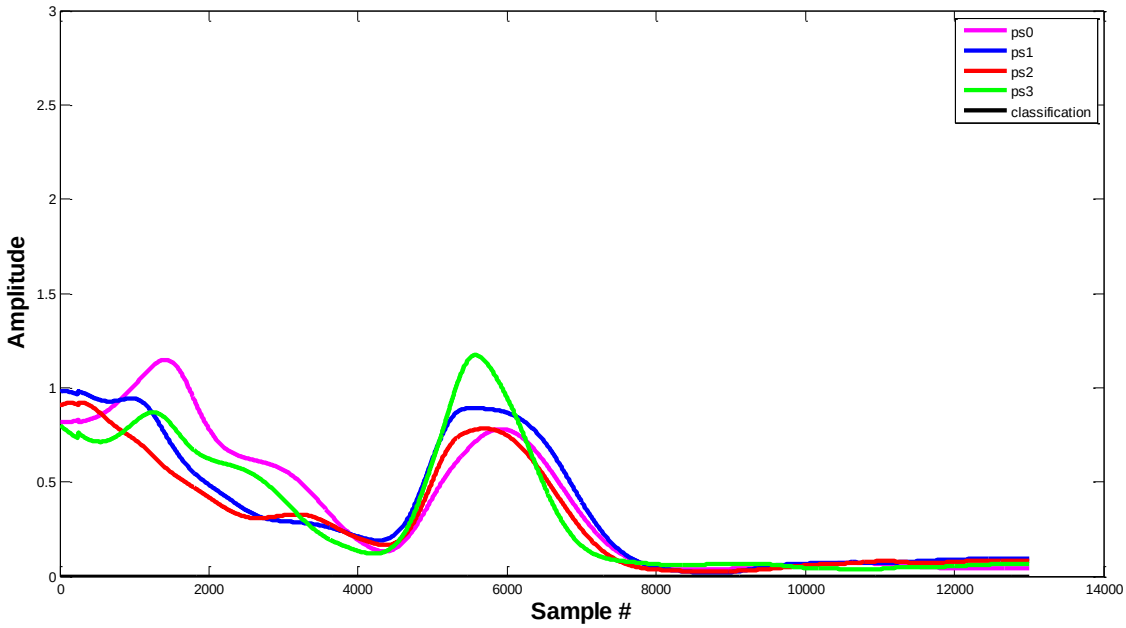

Figure 4.32: Post-processed Classification RVM 201004201100 Standing Position Looses Balance Fall Right

The next two figures show the case in which the stunt actress trips and falls forward. In this case, both classifiers fail to detect the fall activity. This fall data does not look much like a fall but appears to be a very slow fall and this seems to be why the fall activity is not detected.

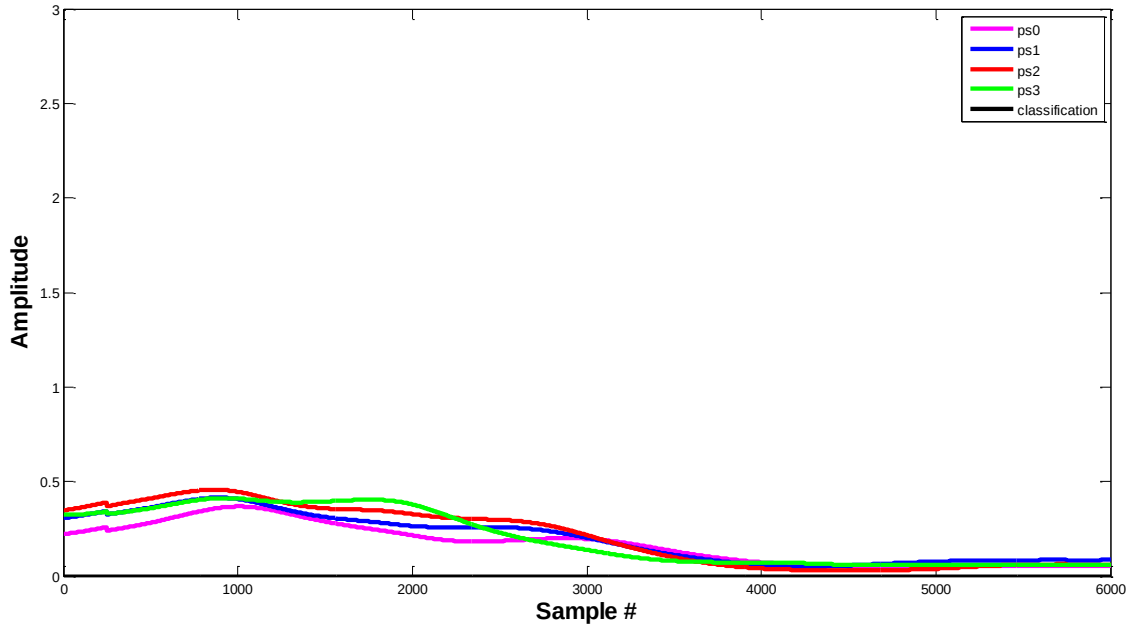

Figure 4.33: Post-processed Classification Parzen 201004201138 Tripping and Slipping Trip and Fall Forward

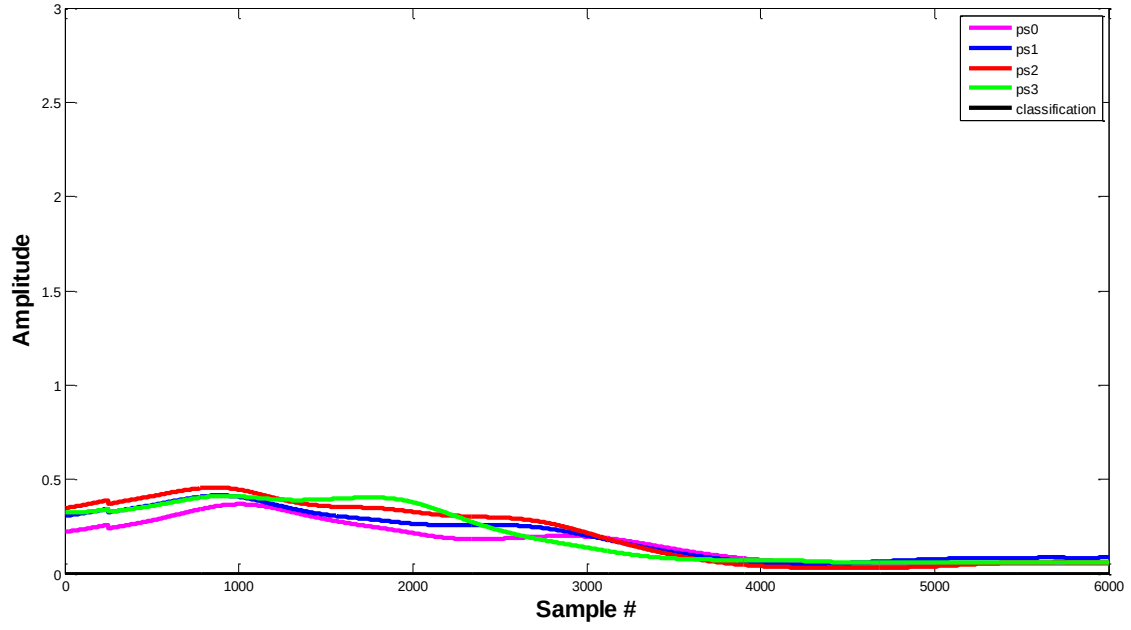

Figure 4.34: Post-processed Classification RVM 201004201138 Tripping and Slipping Trip and Fall Forward

In the next two figures, the RVM is able to detect the fall activity, and the Parzen Window is not. The fall activity in these figures seems detectable. It is possible that the feature values are just out of the detectable range during this fall activity.

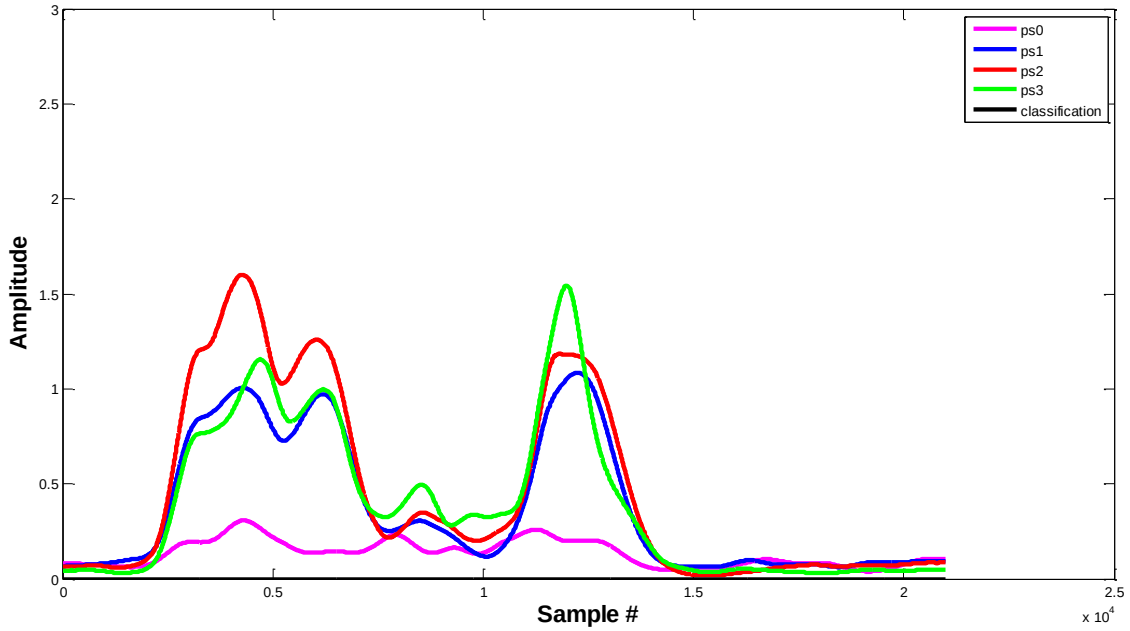

Figure 4.35: Post-processed Classification Parzen 201004201158 Sitting Falling From a Chair Fall Left

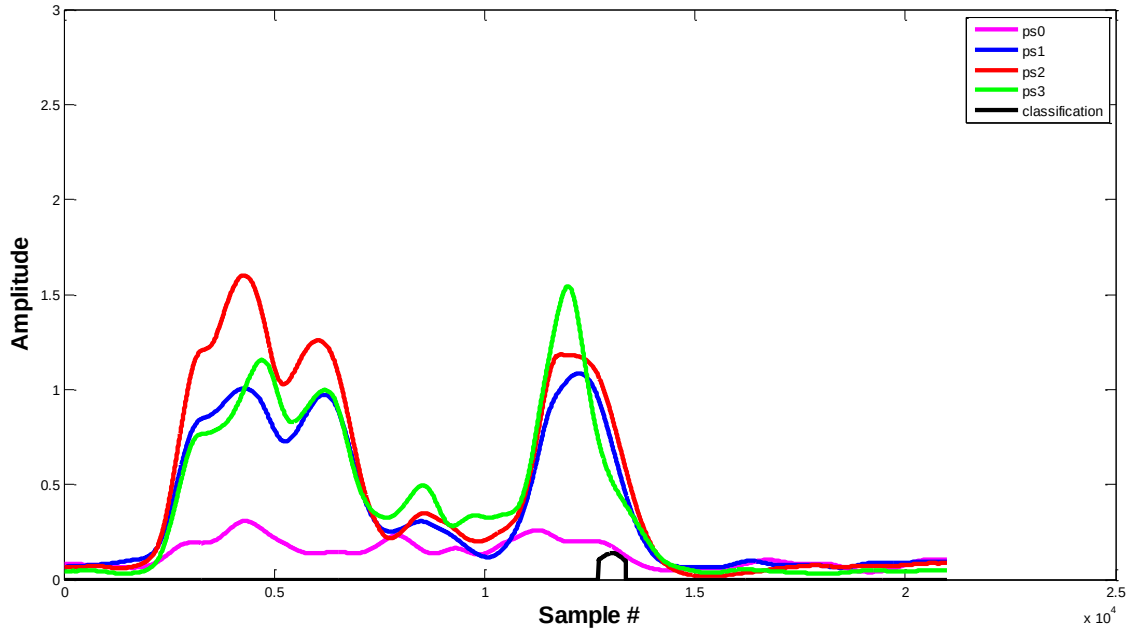

Figure 4.36: Post-processed Classification RVM 201004201158 Sitting Falling From a Chair Fall Left

The last two figures of this section show a non-fall activity where each of the classifiers perform very poorly, detecting several falls within one non-fall data file.

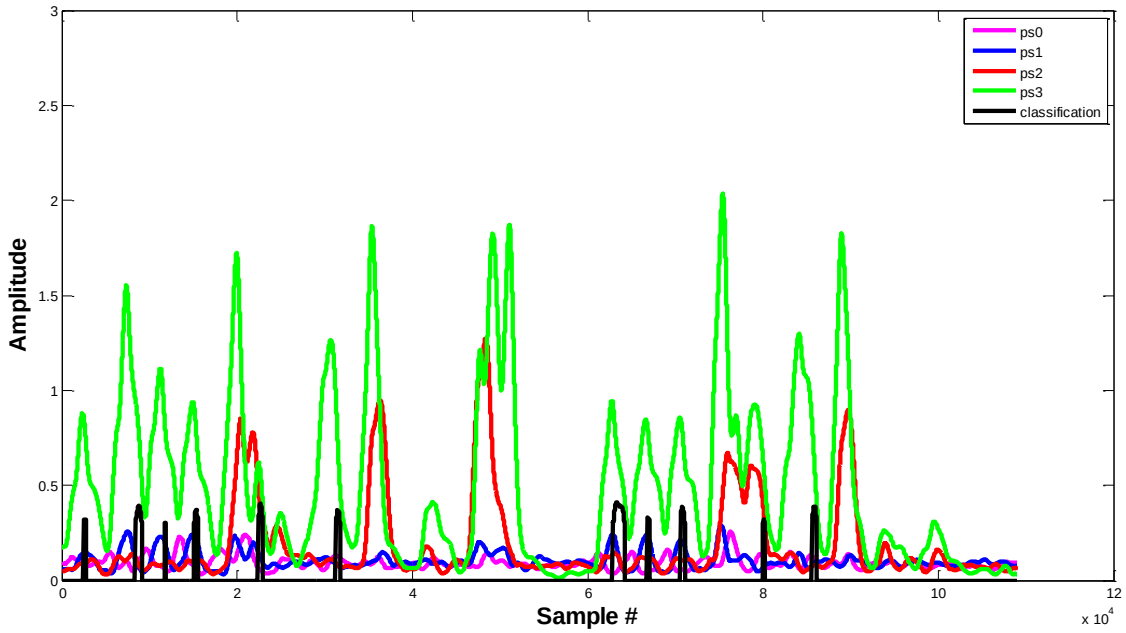

Figure 4.37: Post-processed Classification Parzen 201004201329 False positive 8 Standing to Sit-ups and Stretches

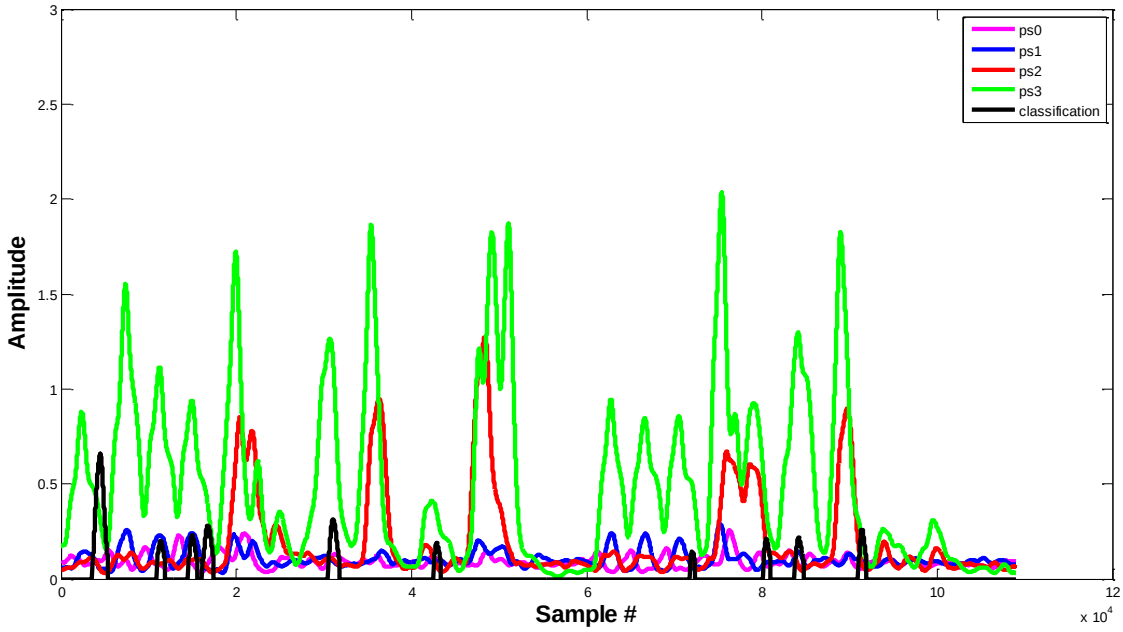

Figure 4.38: Post-processed Classification RVM 201004201329 False positive 8 Standing to Sit-ups and Stretches

This section showed examples in which each classifier was able to successfully and unsuccessfully detect fall activity. It is difficult to identify exactly why the classifiers fail for a given fall activity. However, in general, it seems that more, or better, features and fall data could greatly improve the performance of either of the classifiers used in this research.

Chapter 5 - Discussion

Due to its sparse training data set, the RVM is capable of classifying the entire testing set in 2 hours, 27 minutes, and 58 seconds, where the Parzen Window is only able to do the same thing in 8 hours, 55 minutes, and 49 seconds. Besides being faster, as discussed below, the RVM is also slightly more accurate.

Below in Table 5-1 the results of the Parzen Window classifier are quantified. This table shows that the results of the classifier are dependent on which threshold value is chosen during classification. As discussed in section 3.8, the choice of a threshold value is a balance between minimizing the false negatives or missed falls and minimizing the false positives or instances where someone is alerted of a fall that did not occur. Increasing the threshold value tends to increase the number of false negatives and decreasing the threshold value tends to increase the number of false alarms. From Table 5-1 the choice of a threshold value of 0.3 is the best balance of minimizing both the false positives and false negatives; this yields an accuracy of 75.41%. This is not the highest accuracy value, but as can be seen in the results table, increasing the accuracy means increasing the number of false negatives which represent a fall that goes undetected. Since a fall that goes undetected could mean injury to an elderly resident, and a false positive only means potential inconvenience to care takers, a threshold value should be chosen with a preference towards minimizing false negatives.

Table 5-1: Parzen Window results after filtering

Parzen Window			___FN___	___FP___				
	Threshold	TP	Missed Fall	False Alarm	TN	TP_rate	FP_rate	Accuracy
	0.0	22	1	36	2	0.96	0.95	39.34
	0.1	20	3	22	16	0.87	0.58	59.02
	0.2	20	3	16	22	0.87	0.42	68.85
	0.3	18	5	10	28	0.78	0.26	75.41
	0.4	16	7	7	31	0.70	0.18	77.05
	0.5	14	9	2	36	0.61	0.05	81.97
	0.6	9	14	1	37	0.39	0.03	75.41
	0.7	5	18	0	38	0.22	0.00	70.49
	0.8	3	20	0	38	0.13	0.00	67.21
	0.9	0	23	0	38	0.00	0.00	62.30
	1.0	0	23	0	38	0.00	0.00	62.30
	1.1	0	23	0	38	0.00	0.00	62.30

Below Table 5-2 shows the classification results of the RVM. In the table it can be seen that, as expected, the results of the classifier are dependent on which threshold value is chosen during classification the same way they are with the Parzen Window. From Table 5-2, the choice of a threshold value of 0.1 is the best balance of minimizing both the false positives and false negatives. Choosing 0.1 for the threshold value yields an accuracy of 80.33%. Once again, this is not the highest accuracy value but as can be seen in the results table, increasing the accuracy means increasing the number of false negatives which is not in the best interest of the resident and thus a threshold value of 0.1 is chosen.

Table 5-2: RVM results after filtering

	Threshold	TP	___FN___ Missed Fall	___FP___ False Alarm	TN	TP_rate	FP_rate	Accuracy
RVM	0.0	23	0	29	9	1.00	0.76	52.46
	0.1	18	5	7	31	0.78	0.18	80.33
	0.2	14	9	2	36	0.61	0.05	81.97
	0.3	10	13	1	37	0.43	0.03	77.05
	0.4	9	14	1	37	0.39	0.03	75.41
	0.5	8	15	1	37	0.35	0.03	73.77
	0.6	3	20	1	37	0.13	0.03	65.57
	0.7	1	22	0	38	0.04	0.00	63.93
	0.8	1	22	0	38	0.04	0.00	63.93
	0.9	0	23	0	38	0.00	0.00	62.30
	1.0	0	23	0	38	0.00	0.00	62.30
	1.1	0	23	0	38	0.00	0.00	62.30

Though the results discussed in this chapter show that falls can be detected with a vertical array of PIR motion sensors, there are still improvements that can be made. The training data set does not include a completely diverse representation of the data in the fall protocol; since there is very little data in the training data set to represent the variety of false positive activities that are specified in the fall protocol. In the future, it could be a major advantage to collect enough data files to distribute between both the training and testing data sets.

Also, to improve the performance of fall detection with this sensing array, more classification methods could be explored such as a nearest neighbor. The nearest neighbor is related to the Parzen Window and would likely provide similar classification results but could possibly require less memory and computational complexity.

Chapter 6 - Summary and Conclusion

This research presented the results of a vertical array of passive infrared motion sensors for fall detection. Though this work was successful, future work could be focused on a few areas to likely improve performance. First, since the raw signals already exist in the frequency domain, doing more frequency domain exploration of the raw sensor signals could provide more fruitful results. Also, staying in the frequency domain would likely reveal more features that could be used for classification. Though there are areas of this research that can be improved upon, this research provides some novel contributions. First and foremost, this research provides a tested vertical array of passive infrared motion sensors that are capable of detecting different types of falls. The sensing array explored in this research has the potential to protect the independence of its users by providing a non-wearable fall detection platform that will not leave them feeling as though they are being watched in private areas of the home. This research also successfully demonstrated the use of the Parzen Window and the RVM as a means of identifying falls that occur within the field of view of the sensing array. Although the RVM yielded better results, the Parzen Window was useful in providing preliminary work for the implementation of the RVM.

Bibliography

[1] P. Kannus, et al., "Fall-Induced Injuries and Deaths Among Older Adults," *Jamma*, pp. 1895-1899, 1999.

[2] A. K. Bourke, J. V. O'Brien, and G. M. Lyons, "Evaluation of a threshold-based tri-axial accelerometer fall detection algorithm," *Gait and Posture*, vol. 26, pp. 194-199, 2007.

[3] R. E. Roush, T. A. Teasdale, J. N. Murphy, and M. S. Kirk, "Impact of a Personal Emergency Response System on Hospital Utilization by Community Residing Elders," *Southern Medical Journal*, pp. 917-922, 1995.

[4] A. K. Bourke and G. M. Lyons, "A threshold based fall-detection algorithm using a bi-axial gyroscope sensor," *Medical Engineering and Physics*, vol. 30, pp. 84-90, 2008.

[5] D. Anderson, et al., "Linguistic summarisation of video for fall detection using voxel person," *Computer Vision and Image Understanding*, vol. 113, pp. 80-89, 2009.

[6] D. Anderson, et al., "Modeling Human Activity From Human Voxel Person Using Fuzzy Logic," *IEEE Transactions on Fuzzy Systems*, vol. 17, no. 1, pp. 39-49, Feb. 2009.

[7] M. Popescu, Y. Li, M. Skubic, and M. Rantz, "An Acoustic Fall Detector System that Uses Sound Height Information to Reduce False Alarm Rate," in *30th Annual International IEEE EMBS Conference*, Vancouver, Brittish Columbia, Canada, 2008, pp. 20-24.

[8] A. Sixsmith and N. Johnson, "A Smart Sensor to Detect the Falls of the Elderly," *IEEE Pervasive Computing*, pp. 42-47, Apr. 2004.

[9] T. Banerjee, J. M. Kelller, M. Skubic, and C. Abbott, "Sit-To-Stand Detection Using Fuzzy Clustering Techniques," in *IEEE World Congress on Computational Ineligence (FUZZ-IEEE)*, Barcelona, Spain, 2010.

[10] E. E. Stone, D. Anderson, M. Skubic, and J. M. Keller, "Extracting Footfalls from Voxel Data," in *32nd Annual International Conference of the IEEE EMBS*, Buenos Aires, Argentina, 2010, pp. 1119-1122.

[11] Panasonic. (2009, Oct.) Panasonic Motion Sensor Design Guide. PDF Document.

[12] Panasonic. (2011, Apr.) Panasonic MP Passive Infrared Motion Sensor Datasheet. [Online]. http://pewa.panasonic.com/assets/pcsd/catalog/napion-catalog.pdf

[13] Glolab. (2009, Aug.) How Infrared Motion Detector Components Work. [Online]. http://www.glolab.com/pirparts/infrared.html

[14] G. A. Babich and O. I. Camps, "Weighted Parzen Windows for Pattern Classification," *IEEE Transactions on Pattern Analysis and Machine Intelligence*, vol. 18, no. 5, pp. 567-570, May 1996.

[15] C. M. Bishop, *Pattern Recognition and Machine Learning*. New York, USA: Springer Sciece+Business Media LLC, 2006.

[16] R. O. Duda, P. E. Hart, and D. G. Stork, *Pattern Classification*, 2nd ed. New York: John Wiley & Sons Inc., 2001.

[17] L. Wei, Y. Yongyi, R. M. Nishikawa, M. N. Wernick, and A. Edwards, "Relevance Vector Machine for Automatic Detection of Clustered Microcalcifications," *IEEE Transactions on Medical Imaging*, vol. 24, no. 10, pp. 1278-1285, Oct. 2005.

[18] T. Fletcher. (2010, Oct.) tristanfletcher.co.uk. [Online]. http://www.tristanfletcher.co.uk/RVM%20Explained.pdf

[19] M. E. Tipping, "Sparse Bayesian Learning and the Relevance Vector Machine," *Journal of Machine Learning Research*, vol. 1, pp. 211-214, May 2001.

Appendix A - Fall Test Protocol

Fall observations

There are 21 possible fall directions from standing, sitting, tripping, and lying positions. For the fall detection algorithm development and validation, we will collect data using the fall and fall-risk sensing system for the set of falls listed below. Written criteria for each type of fall guide the stunt actors so that the falls performed strongly resemble the falls of elders as validated in preliminary work.

Standing position

From an initial standing position, the stunt actor will fall as if loosing balance; then as if there is a momentary loss of consciousness.

Criteria for "looses balance falls":

- Leans forward, leans to left, leans to right, or leans back
- Looks down
- When falling, tries to break fall with upper extremities

1. fall forward
2. fall backward
3. fall to the left side
4. fall to the right side

Criteria for "momentary loss of consciousness falls":

- Falls much like a tree, toppling backward or forward or sideways
- Crumples to the floor
- No attempt to break fall with upper extremities

1. fall forward
2. fall backward
3. fall to the left side
4. fall to the right side
5. fall straight down

Tripping and Slipping

From an initial walking position, the stunt actor will trip and fall; then from an initial walking position the actor will slip (as if on water or ice) and fall.

Criteria for walking, then tripping and slipping falls:

- Walks with shortened stride
- Leans forward when walking
- Looks down when walking
- When falling tries to break fall with upper extremities

1. trip and fall forward
2. trip and fall sideways
3. slip and fall forward
4. slip and fall sideways
5. slip and fall backward

Sitting position

From an initial position sitting on a stationary chair (no wheels), the stunt actor will fall from the chair as if loosing balance; then slide forward and backward out of a chair with wheels.

Criteria for falling from the chair:

- Leans until center of gravity changes and falls off the chair
- Attempts to break fall with upper extremities

1. fall forward
2. fall to the left side
3. fall to the right side
4. fall by sliding forward out of the chair as the chair slides back
5. fall by sliding backward out of the chair as it slides back

From Bed or Couch

From a lying position, the stunt actor will roll off the bed or couch in a state of semi-wakefulness or sleep.

Criteria for falls from bed or couch:

- Somewhat awakens, attempts to get up and falls
- Sleeping, attempts to get up, legs get caught in blanket and falls
- Attempts to break fall with upper extremity

1. fall to side, upper body falls first

Criteria for falls from bed or couch, does not awaken:

- Rolls too close to edge of bed or couch, center of gravity changes and rolls off

- No attempt to break fall with upper extremity

1. fall to side, hips and shoulders fall first

Fall Test Safety

An athletic floor mat used for sports training will cover the floor where the falls will occur. The stunt persons will wear joint protection pads during the testing. We will recruit the theater-trained stunt person and make a reference check to assure each person has completed training and has experience in falling for stage productions. Stunt persons will be paid for their services. We will select an athletically fit stunt person to further minimize the risk of injury.

False positive Test Protocol

Fifteen motions appear similar to falls. To ensure that the sensors and algorithms developed for this study avoid recognizing non-falling motions as fall events, we will collect data using the fall and fall-risk sensing system for the following set of activities:

1. From a standing position, the stunt actor will bend at the knees and stoop to a squatting position on the floor.
2. From a standing position, the actor will bend down and kneel on the floor.
3. From a standing position, the actor will bend down and kneel on the floor, wait for two seconds, then lie down on the floor.
4. From a standing position, the actor will bend down to plug an appliance into an electrical outlet close to the floor.
5. From a standing position, the actor will squat to tie a shoe.

6. From a standing position, the actor will sit on the floor with the legs tucked under the body.
7. From a standing position, the actor will sit on the floor with the legs extended from the body.
8. From a lying position on the floor, the actor will perform three sit-ups and some stretches of upper and lower extremities.
9. From a lying position on the floor, the actor will slowly rise to a half kneeling position, then rise to a standing position.
10. From a walking position, the actor will appear to trip but will regain balance and continue walking.
11. From a standing position, the actor will walk forward for three seconds, then stop suddenly.
12. From a standing position, the actor will walk forward for three seconds, then stop suddenly and turn around.
13. From a standing position, the actor will walk to a stationary chair and sit in it.
14. From a sitting position in a chair, the actor will bend over to pick up a book on the floor.
15. From a standing position, the actor will walk to a stationary chair, sit in it, and attempt to stand

Appendix B – Software Manual

Below is a manual for running the software developed for this research. All programs and data are found in the Research directory stored on the kronos server in home\shared\PIR_fall_detection_moore. Within the research directory, all of the MATLAB programs are stored in the Data_Analysis folder and all of the data files and file path text files are stored in the Trial_Data folder.

Preprocessing

Once data has been collected, each of the data files must be preprocessed. This is done by editing the file paths listed in the data_to_preprocess.txt file, which should contain one column of file path strings that link to individual fall data files. After that, the preprocessing.m MATLAB program is run. This program loads the data_to_preprocess.txt file and executes the preprocessing algorithm on each of the data files listed. After each data file is preprocessed a new data file is created and saved within the same directory with the same file name and a "_PP" string appended to the end of the file name.

Feature Extraction

Once all of the data files have been preprocessed, the features described in this research are extracted. To extract features, the extract_features_file_paths.txt should be edited to link to all of the preprocessed data files including the "_PP" string at the end. Once that is done, the extract_features.m file is run. This program extracts features

from the preprocessed data files and then saves the extracted feature data in a file that has a "_feat" string appended to the end of it.

Training Data Set

Once features have been extracted from all of the data files, the data can be separated into testing and training data sets. To build the training data set, the training_file_paths.txt file should point to the feature extracted data files that belong in the training data set. After that, the build_training_set.m program is run. This program organizes the training data in to class 1 and class 2 matrices and saves them in a .mat file. The training data set used in this research was saved as training_data2.mat.

Parzen Window

Once the data has been separated into testing and training data sets, the classification algorithms are run. To run the parzen window algorithm, the fall_file_paths.txt file should contain the file paths of the data files that are to be classified. To run the Parzen Window algorithm on the data, the classify_falls2.m program is run. This program uses the mm_parzen2.m function that is included in the mm_tb folder to execute the parzen window algorithm. Once the data has been run through the Parzen Window algorithm, the resulting data files are saved with a "_clas" string appended to their file name.

RVM

The RVM algorithm is run similarly to the Parzen Window algorithm except it requires a separate training process. To execute the training process, the build_rvm.m

program is run. This program takes the training data as input and returns a file which contains the relevance vectors and weights that were obtained in the training process. Next, the run_rvm.m program loads the file paths of the data that is to be classified as stored in the fall_file_paths.txt file and the relevance vector and weight data. After that, the data is run through the RVM algorithm and saved with a "_rvm" string appended to its file name.

Post-processing (filtering)

After the data is run through the RVM and Parzen Window algorithms, the resulting data is filtered using the post-processing algorithm in the PostProcess.m program which is configured to only filter the data from the previous step. This program uses the classified_file_paths.txt file to find the data files that are to be processed. Once post-processing is complete, the files are saved with a "b" appended to the end of their file name and saved in the same directory as the source data file.

ROC Curves

To help choose a threshold value for classification of the fall data, ROC curves are generated using the Generate_ROC.m program. This program uses ROC_file_paths.txt to find the data files from which the ROC curves are to be generated.

Classification

After a threshold value is chosen, the resulting data is filtered using the post-processing algorithm in the PostProcess.m program, which is configured to only implement the threshold value as to classify the data from the filtering step. This

program uses the classified_file_paths.txt file to find the data files that are to be processed. Once classification is complete, the files are saved with a "_popr" string appended to the end of their file name and saved in the same directory as the source data file.