UNIVERSITY OF OKLAHOMA

GRADUATE COLLEGE

ACCURATE VEHICLE CLASSIFICATION INCLUDING MOTORCYCLES USING PIEZOELECTRIC SENSORS

A THESIS

SUBMITTED TO THE GRADUATE FACULTY

in partial fulfillment of the requirements for the

Degree of

MASTER OF SCIENCE

In

Electrical and Computer Engineering

By

Samer Rajab

Norman, Oklahoma

2012

OU
THESIS
RAJ
cop. 2

ACCURATE VEHICLE CLASSIFICATION INCLUDING MOTORCYCLES USING PIEZOELECTRIC SENSORS

A THESIS APPROVED FOR THE

SCHOOL OF ELECTRICAL AND COMPUTER ENGINEERING

BY

[redacted]

Dr. Hazem Refai, Chair

[redacted]

Dr. Pramode Verma

[redacted]

Dr. Samuel Cheng

ACKNOWLEDGMENTS

The work in this thesis wouldn't have been possible without the continuous support of individuals whom I'd like to take this opportunity to acknowledge.

First my love and appreciation go to my mother, father, Khaled, Reem, Majd and all of my family for their unconditional love and support.

Special thanks go to my advisor Dr. Hazem Refai for his invaluable guidance, patience and supervision. His help and directions have been the major reason for this thesis success.

Dr. Pramode Verma and Dr. Samuel Cheng valuable participation in my master's committee is highly appreciated.

Acknowledgment goes to my friends who have helped me in testing and other aspects associated with work in this thesis; Ahmad, Nathanael, Yahia, Sina and Asaad. My gratitude goes to them and is extended to my friends in U.S. whose support was a major motive for me to work harder and better; Hassan, Maryam, Rami, Samer, Mayssaa, Rima and all others whom I was unsuccessful to mention due to limited space.

Michelle Farabough is acknowledged for her work in editing this thesis.

The University of Oklahoma is very well appreciated and acknowledged for support and embracement

Contents

Abstract

State and federal departments of transportation are charged with classifying vehicles and monitoring mileage traveled. Accurate data reporting enables suitable roadway design for safety and capacity. Vehicle classifier devices currently employ inductive loops, piezoelectric sensors, or some combination of both, to aid in the identification of 13 Federal Highway Administration (FHWA) classifications. However, systems using inductive loops have proven unable to accurately classify motorcycles and record pertinent data. Previous investigations undertaken to overcome this problem have focused on classification techniques utilizing inductive loops signal output, magnetic sensor output with neural networks, or the fusion of several sensor outputs. Most were off-line classification studies with results not directly intended for product development. Vision, infrared, and acoustic classification systems among others have also been explored as possible solutions.

This thesis presents a novel vehicle classification setup that uses a single piezoelectric sensor placed diagonally on the roadway to accurately identify motorcycles from among other vehicles, as well as identify vehicles in the remaining 12 FHWA classifications. An algorithm was formulated and deployed in an embedded system for field testing. Both single-element and multi-element piezoelectric sensors were investigated for use as part of the vehicle classification system.

The piezoelectric sensors and vehicle classification system reported in this thesis were subsequently tested at the University of Oklahoma-Tulsa campus. Various vehicle types traveling at limited vehicle speeds were investigated. The newly developed vehicle classification system demonstrated results that met expectation for accurately identifying motorcycles.

CHAPTER I

1 Introduction

State and federal departments of transportation in the United States monitor vehicle classifications and miles travelled. Accurate reporting of this data is essential for highway and roadway design, ensuring adequate capacity and driver safety. Current vehicle classifier systems use inductive loops, a piezoelectric sensor, or a combination of both to classify vehicles into one of the 13 FHWA classifications. However, systems utilizing inductive loops have been unable to accurately classify motorcycles.

The Federal Highway Administration (FHWA) collects Vehicle Miles Travelled (VMT) for each vehicle classification to predict accident trends, namely risks and fatalities. The National Highway Traffic Safety Administration (NHTSA) is charged with maintaining roadway fatality and injury counts in the United States. In 2005 the agency reported 4,576 fatalities and 87,000 injuries from motorcycles accidents—an increase of 13% and 14%, respectively, since the previous year [1]. Between 1996 and 2005, motorcycle registration was up 61% and motorcycle fatalities increased110%. Both could possibly be the result of a rising trend in motorcycle use following elevated gas prices. Contrary to these figures, motorcycle VMT increased only 8.6% during the same period. This disparity implies a deficiency in currently employed classification systems.

To date, the most popular classification systems detect motor vehicle axles using two piezoelectric sensors and one inductive loop located between the piezoelectric sensors or one loop located before and one between the two sensors. The loop configuration is used to trigger the system and can also be used to measure a vehicle's magnetic length. However, problems with such systems include the following

- A motorcycle might not have sufficient metal to trigger the inductive loop.
- A motorcycle might pass next to instead of on top of the inductive loop, as full-lane sensor coverage is infrequent.

Piezoelectric sensors are of particular significance as they are used in Weight In Motion WIM systems. WIM system weighs vehicles as they move on highway. This task can be done using several methods such as piezoelectric sensors, capacitive plate transducer, strain gauge sensor, fiber brag grating and microwave sensors [2][3]. Though the focus of this work is not implementing an Automatic Vehicle Classification system (AVC), using piezoelectric sensors gives scalability to investigate WIM implementation on the same system.

In this thesis a new classification system is suggested and built to overcome such inherent motorcycle misclassification errors. The computer-based system executes a classification algorithm to process data collected after passing vehicles trigger a road sensor. The algorithm enhances the ability for accurate motorcycle classification by utilizing vehicle features, including vehicle number of tires and length, among others.

1.1 System overview

Typical of most classification systems, the novel system described herein is comprised of three phases:

- Sensing phase: Sensors collect all necessary information and transmit it to the computing system. There the computer executes necessary preprocessing signals, including noise cancellation and amplification when needed, making possible the following phases:
- Feature extraction phase: The computing system processes the signal and extracts necessary distinguishing features for classification, specifically motorcycles—the

target of this research. These include vehicle number of tires, inter axle spacing, velocity, and width.

- Classification phase: The classification algorithm uses input features for classification. Methods investigated and previously detailed in literature are thoroughly investigated in Chapter II. For the system reported in this work, an industry solution using statistically common features for a given classification are used as algorithm inputs.

1.2 Contributions

This thesis presents a novel setup comprised of a single piezoelectric sensor placed diagonally on the road to accurately classify motorcycles, as well as vehicles in other FHWA classes. The author's objective was to formulate an algorithm, and then deploy it in an embedded system. Limited field testing was performed at the University of Oklahoma-Tulsa campus. The final project goal is to interface the vehicle classification system with the Roadside Embedded Extensible Computing Equipment (REECE)—a computing system currently deployed at over 80 sites across the state of Oklahoma.

A single-element piezoelectric sensor is unable to detect vehicle track width which is required for classification process. To accomplish this, an average track width was used. Also a method of using multi-element piezoelectric sensor assembled from an array of smaller piezoelectric sensors was suggested to detect vehicle track width. The multi-element sensor was designed and built to detect vehicle track width as well. As vehicle tires pass the sensor elements, a voltage is triggered and initiates the feature extraction and classification phases. Vehicle track width is calculated using the data acquisition unit (DAQ), which is part of the embedded computing system connected to the sensor elements.

1.3 Organization

This thesis is organized as follows. Chapter II highlights background literature. Previous vehicle classification research and technology currently used by a number of Departments of Transportation (DOT) is described. System development is detailed in Chapter III, including sensor fabrication, DAQ development, and overall system description. Chapter IV offers an extended description of the classification algorithm, as well as the logic behind employing various algorithm features. Chapter V presents field testing results and validation. The thesis concludes and suggested future work is found in Chapter VI.

CHAPTER II

2 Background Search

Automatic vehicle classification systems comprised of a variety of sensing devices has been investigated by both researchers and industry manufacturers for over forty years [20]. The FHWA and DOTs remain interested in ever-evolving systems that provide more accurate classification results. In this chapter several methods and industry products are presented.

2.1 Magnetic sensor and inductive loop based systems

Vehicle types can be classified according to recognized patterns generated by signals harvested after a vehicle triggers either magnetic sensors or inductive loops. In [4] Keawkamnerd et al. use magnetic sensors to measure the earth's magnetic field disturbance. The sensors' small size makes them attractive when compared to the size of inductive loops. Also inductive loops are active devices that need to be excited by a voltage to generate a field of measurement while magnetic sensors are passive devices that measures changes in earth's magnetic field. Thus loops have larger field of measurement, depending on their size, where magnetic sensors have more localized measurement characteristics [5]. Researchers divided vehicles being classified into five groups, selecting an appropriate one based on: magnetic length, average signal energy, and peak number in hill pattern. Magnetic length was used to differentiate buses from the other four types vehicles. Peak number and average energy were used to classify motorcycles, cars, pickups and vans. Although the overall classification success rate using this method was 80.92%, a relatively low number of classification categories were used, indicating that classification rate might suffer significant degradation if a similar algorithm was employed for a greater number of vehicle classes. Inherent problems are the results of an overlap in length between different FHWA categories [6].

Keawkamnerd et al. extended this research in [7], where two Anisotropic Magneto-resistive (AMR) sensors were placed on a road-side mounted board. A microcontroller located on the same board.

Using the onboard microcontroller the vehicle magnetic length and estimated length were calculated. Initially magnetic length was calculated using signal delay between the two sensors using eq 2.1:

$$L = \frac{M}{L_d} \qquad \text{eq. (2.1)}$$

where M is the acquired number of samples during vehicle passage, and L_d is the estimated length.

Next, estimated length was calculated using the number of zero-crossings, as in eq 2.2:

$$N = \frac{\sum_{i=1}^{Nz} S_{T_i}}{N_z} \qquad \text{eq. (2.2)}$$

where T_i is the zero-crossing duration between two sensors; S_{Ti} is the number of samples during Ti; and N_z is number of zero-crossings.

Hill pattern, energy level, and magnetic length were utilized to classify vehicles. When investigating hill pattern the number of peaks of the signal harvested from an inductive loop is used to differentiate vehicle type.

A moving average was computed for the time series of the received signal to smooth it and reduce number of received samples. Differential magnitude and new energy level moving averages were then computed to obtain a rise or fall in energy during vehicle detection. See eq 2.3 and eq 2.4.

$$M_n = \sqrt{Z_n^m - Z_{n-1}^m} \qquad \text{eq. (2.3)}$$

where M_n in differential magnitude between two consecutive moving average values.

$$M_n^m = \frac{M_n + M_{n-1} + M_{n-2} + M_{n-3}}{4} \qquad \text{eq.(2.4)}$$

M_n^m is a four point moving average of the differential magnitude.

Finally, minimum, maximum, and normalized energy levels were computed based on vehicle speed and sampling period.

These parameters were combined in a decision algorithm to distinguish from among four types of vehicles: motorcycle, car, van, and pickup. An overall 81.69% rate in accuracy in classification was observed. Although this demonstrates a marked improvement over previous investigations, again only a few class categories were used.

Earlier research based on the use of two inductive loops for classification was investigated by Pursula and Kosonen in 1989 [8]. In this paper the authors used the output of two inductive loops to calculate the length, speed, and direction of a passing vehicle. These parameters were used in a decision algorithm based on vehicle length.

Two types of detection techniques were presented: 1.) Digital, wherein the received signal was approximated to a digital on/off signal using a specific threshold, and 2.) Analog, wherein the signal edge was approximated with a line to offer consistent time readings or center of gravity method (two centers are used to calculate for velocity).

According Pursula and Konsonen the analog method is more accurate, as the digital method has a higher bias in terms of vehicle lengths recorded.

Ten vehicle types along with several FHWA classifications were combined. Some FHWA classes were split into additional classes. A mean error was shown to be 9%.

Gajda et al. investigated the impact of loop length on vehicle classification in [9]. Loop lengths in the range of 0.25m-4m with a step of 0.25m, as well as a separate 10cm loop used as a reference, were tested. Signals acquired from inductive loops were normalized for velocity and sampling frequency. Several signal characteristics, mainly the magnetic profile, were used as criterion to define vehicle type.

Test vehicles included two types of buses and a passenger vehicle. Testing demonstrated that shorter loops furnish more highly distinguished criterion, and, thus, improved distinction between the three vehicle types. Also, vehicle axles were clearly distinguished in the 10cm loop magnetic profile.

In 1997 Gajda et al. researched the use of one inductive loop to calculate passing vehicle speed [10]. The researchers were able to obtain a correlating parameter between the inductive loop signal and the vehicle speed that was independent of vehicle type. With one inductive loop, passing vehicle speed was measured and compared with results from a two-loop system. With a 10m separation between the two inductive loops, the two loop system demonstrated an inherent error in response to vehicle acceleration or deceleration occurring between the two loops. Results have shown a velocity calculation with a maximum RMS error of 2.5Kmph. However, the authors did not provide information on classification type or classification rate accuracy.

Sokra studied data fusion techniques using parameters collected from an inductive loop and piezoelectric sensor [11]. Parameters from these sensors were used to distinguish vehicle classes. Typically, axle spacing and number utilized with piezoelectric sensor systems. The inductive loop provides information to calculate vehicle magnetic length and vehicle profile.

Magnetic profile parameters include mean value, mean square value, standard deviation, maximum value, moment of third order, and central moment of third order. These, along with the others such as vehicle length, speed, and number of axles, were used in membership functions based on fuzzy logic.

More than one parameter was employed to determine an acceptable distinction between classes. Gaussian shape and a triangular shaped membership function aided in classification, as demonstrated below.

$$\mu_{ij}(x_i) = \begin{cases} 1 - \frac{|xi - m_i|}{2\sigma_i} & for \;\; |xi - m_i| < 2\sigma_i \\ 0 & otherwise \end{cases} \qquad \text{eq. (2.5)}$$

$$\mu_{ij}(x_i) = e^{\frac{(x_i - m_i)^2}{2\sigma_i^2}} \qquad \text{eq. (2.6)}$$

where x_i is the ith element in the feature vector; m_i is the mean; and σ_i is the standard deviation.

Fusion of the parameters was accomplished with the following functions.

$$f_{(AND)j} = \cap_{i=1}^{N} \mu_{ij}(x_i) = min(\mu_{1j} \ldots\ldots \mu_{Nj}) \qquad \text{eq. (2.7)}$$

$$f_{(AND)j} = \cup_{i=1}^{N} \mu_{ij}(x_i) = max(\mu_{1j} \ldots\ldots \mu_{Nj}) \qquad \text{eq. (2.8)}$$

$$f_{(power)j} = \frac{1}{N} \sum_{i=1}^{N} \mu_{ij}(x_i) \qquad \text{eq. (2.9)}$$

$$f_j = \frac{\prod_{i=1}^{N} \mu_{ij}}{\prod_{i=1}^{N} \mu_{ij} + \prod_{i=1}^{N}(1-\mu_{ij})} \qquad \text{eq. (2.10)}$$

$$f_{(weight)j} = \left(\frac{1}{K^N - 1}\right)\left(\frac{-1+K^N G_j}{1+G_j}\right) \qquad \text{eq. (2.11)}$$

$$\text{Where: } G_j = \prod_{i=1}^{N} \frac{1+(K-1)\mu_{ij}}{K-(K-1)\mu_{ij}} \; for \; 1 < K < \infty \qquad \text{eq. (2.12)}$$

The researchers evaluated the results based on classification of four, one-axles vehicle classes; cars, vans, lorries, and buses. Classification rates ranged from 0.68 to 0.92 with the triangular membership function and from 0.68 to 0.94 for Gaussian membership. Improved performance was obtained using the Gaussian membership function.

In 2003, Sun et al. researched a system with two inductive loops and an Inductive Classifying Artificial Network (ICAN) [12]. ICAN is a self-organized feature map (SOFM) employing inductive loop output from classification in a manner similar to the number of neurons. The researchers selected SOFM for several reasons, including unsupervised learning which enables the network to retrain and cluster itself and that network weights are directly related to classification templates. Distance between neurons corresponded to frequency of class occurrence.

Distances between the input vector and neural network weights are calculated at each iteration, leading to one winning neuron, whose weight will be updated. Weights of neurons within the same neighborhood, Nc, winning neuron will also be updated, as seen in eq 2.13.

$$w_i(K+1) = \begin{cases} w_i(k) + \alpha(k)[x(k) - w_i(k)] \\ w_i(k), \qquad\qquad inot\ inNc(k) \end{cases}, \quad i\epsilon Nc(k) \qquad \text{eq. (2.13)}$$

where w_i is the ith weight, and α is the learning rate.

The researchers' test set consisted of 300 carefully selected vehicles in four categories. Initial test results indicated 77% accuracy in classification—a moderately acceptable rate due to misclassification of vehicles that were listed in an adjacent class. When passenger cars change lane, inductive signature output decreases. In this way, pickup and SUV classification is problematic. Their scheme was updated to include four categories by merging some vehicle types. Classification rate increased to 81% accuracy.

The scheme was then expanded to a more complicated system with nine categories. A classification rate of 71% accuracy was achieved. Limiting classes to seven categories resulted in an 87% and an 82% accuracy rate for two data sets. Output classification rate was low compared to similar research, which can be explained by the fact that the focus was primarily to differentiate between two axle vehicles. As such, five of the seven categories in the scheme were two axle vehicles.

Meta and Cinsdikici in 2010 used a single inductive loop to contribute to vehicle classification research [13]. Their classification system was comprised of a 2X1 single loop, inductive loop detector, validation camera, and computer. Their contributions included signal preprocessing, data set reduction, and appropriate training set selection for their neural network vehicle classification system.

The researchers chose five classification categories:

- Class1: car, SUV
- Class2: minibus, van
- Class3: pickup, truck
- Class4: bus, articulated bus
- Class5: motorcycle

First in their procedure, unwanted noise and unexpected ripples were cleaned from the collected inductive loop raw signal, using DFT methods where frequency filters smoothed the signal. The sampled signal was then transformed to frequency domain using DFT. Next, the transformed signal was shifted to zero frequency at the center. The noise signal, R^{-}, was defined using the following equations.

$$R = \{X_{shifted}(i)|_{i=1}^{N}\} \qquad \text{eq. (2.14)}$$

$$R = \left\{ X_{shifted}(i) \Big|_{i=\left(\frac{N}{2}\right)-\beta}^{\left(\frac{N}{2}\right)+\beta} \right\} \qquad \text{eq. (2.15)}$$

$$R^{-} = R - R^{+} \qquad \text{eq. (2.16)}$$

where β was defined as the distance from the DC component, α is the noise coefficient, which was set at 0.06 and R is the shifted signal set. After noise cancellation the signal was converted to time domain for further processing.

In previous methods the signal was down sampled to reduce size for processing and applicability to neural network (NN). This caused a loss in signal feature. Meta and Cinsdikici implemented a newer approach using principal component analysis to capture the signal's main feature without need for a large data set. After subtracting the mean of each inductive loop signal sets the figures were added as columns of a single matrix called signal matrix. Decomposition of the signal covariance matrix captured eigenvalues and eigenvectors of the signal matrix, which served as feature vector. Multiple tests were performed on several feature set sizes, namely 4, 8, 16, and 32. Superior performance was demonstrated by16.

The researchers included the local maximum as another parameter to increase distinction among classes. This was particularly useful, as vehicle classes have different chassis formation and heights, which in turn affects inductive loop output. L_{max} was added to the Principal Component Analysis (PCA) to constitute the feature vector. A data set of 1000 vehicles was used to train the neural network. A data set of 1330 vehicles was used for testing. Multiple feature extraction methods where investigated with the nearest neighbor technique for100 tests. These included PCA, Linear Discriminant Analysis (LDA), Independent Component Analysis (ICA), PCA-LDA, and PCA-ICA. Superior results were obtained using PCA with NN. The classification rate for the 100 experiments was 91.13%.

A three three-layered Back Propagation Neural Network was used. To select the number of nodes in the hidden layer, four values were tested_ 10, 20, 30, and 40. Optimal node number relative to cognition rate and computational complexity was 20.

The configuration offered an overall classification accuracy rate of 93.53%. The researchers added L_{max} count to the NN input to improve the algorithm's classification rate. Overall recognition rate was improved to 94.21% accurate.

2.2 Vision Based Systems

Vision-based classification system research has recently enjoyed increased popularity due to advances in image processing systems and techniques, making the systems easier and more efficient. Installing traffic-monitoring cameras is often less disruptive, expensive, and easier than installing in-pavement sensors, which typically involves scoring the roadway and damaging the surface. Nonetheless, vision-based systems remain limited for vehicle classification. Lens maintenance and camera operation in heavy weather conditions, e.g., rain and fog, are significant challenges. Also those systems suffer from several effects such as shadows and illumination variations [14].

Because these systems lack relative significance to the research presented herein, this section offered limited attention to these systems.

In 2002, Gupte et al. investigated a system using gray-scale images to detect, track, and classify vehicle [14]. The approach employed a single, overhead camera monitoring several traffic lanes and was comprised of the following phases: segmentation, feature extraction, vehicle identification, tracking, and vehicle classification.

In segmentation, passing vehicles foreground was separated from the background. After identifying and removing the foreground pixels, background revisions were made using adaptive background update to save both the background and the current frame in a weighted average.

During the tracking phase a binary mask separated vehicles from the background in the current frame and tracked movement. Inherent problems included region splitting or combining, region sudden appearance, and region sudden disappearance. These were somewhat alleviated by building an association graph between consequent frames, and then using the graph for decision-making on regions in the current frame.

Calibration is required to extract vehicle features, such as length and width, among others. To do so, traffic lane standard characteristics, e.g., stripes, were used with a calibration user interface tool. Using stripes was advantageous because there is a standard separation distance between each.

Throughout the vehicle identification phase typical vehicle sizes were used to set maximum and minimum thresholds. These were then used to identify whether or not a region actually represented a vehicle.

Vehicle parameters were updated as the vehicle moved through the scene from one frame to another, using the aforementioned association graph. The researchers classified vehicles into two major categories: cars and non-cars. Cars included passenger vehicles and non-cars for all larger vehicle types. A sample of 20 minutes of highway traffic passed without mention of actual vehicle number passed. The system demonstrated a 90% accuracy rate for tracking and a 70% accuracy rate for classification.

In 2009, Shaoqing et al. used three overhead, mounted cameras tilted 60 degrees to classify vehicles on a multi-lane highway [15]. Their proposed system classified three categories (cars, trucks, and buses) through three stages. Rough classification is initiated by extracting the license plate region from the first camera. To do this, a vertical and horizontal region representing the highway was compared to regions where the vehicle is expected to arrive. In rough classification cars and non-cars were distinguished based on license plate color according to the standard license plate colors in China.

Next feature extraction stage employed five discriminatory features, namely total number of regions; total number of colors; window size (big widow), number of edges of vehicle's top, and low gray region of top. The researchers specifically chose these to differentiate trucks and buses through statistical search. Of note is that the parameters chosen represent a limited set of vehicle classification.

The vehicle classification stage compared passing vehicle features to stored features. The researchers competed two training sets: one hour between 7 and 8a.m. A total of 438 cars, 174 trucks, and 58 buses were accurately classified at a rate of 95%, 90.8%, and 86%, respectively. A second test was performed between 3 and 4p.m. A total of 413 cars, 183 trucks, and 27 buses were accurately classified at a rate of 94.1%, 88.5%, and 81.5%, respectively.

2.3 Piezoelectric based system

In 2008, Zhang et al. used an electrical resistance strain gauge sensor based on piezoresistive material to perform axle detection for vehicle classification [16]. Typically these sensors monitor pavement status, especially on bridges. The researchers assumed that the sensors can also be used for both pavement monitoring and vehicle classification.

Although five in-pavement sensors were installed, only three were used as a result of high correlation between sensors output. Each vehicle axle produced its own peak on the output of each individual sensor. The researchers aimed to classify vehicles into five distinct categories: small trucks, medium trucks, buses/large trucks, 3-axle trucks, and combination trucks.

A preliminary test was first conducted to perform feature extraction and confirm measurement repeatability. Two-axle vehicle at different velocities were examined. Maximum wheelbase measurement error was 4.1%, which can be explained by the vehicle's acceleration/deceleration when passing over the combination of sensors. Of note is that the sensors covered a distance of 4.1m, with a spacing of 2.05m between consecutive sensors.

A final vehicle classification was accomplished with support vector classification SVM—a machine learning pattern recognition/classification technique. SVM is a binary linear classifier in which input data is classified into one of two classes. The researchers immigrated the problem to multi-class SVM via two separate methods: One Against All (OAA) and One Against One (OAO). A data set from 602 highway vehicles was collected and used for training 50%, validation (control set) 25%, and test 25%.

Two data fusion techniques were proposed to combine information from the three different sensors:

- Centralized fusion extracted features by combining sensor outputs, and then applying a SVM method in a centralized manner.
- Distributed fusion extracted features from each sensor output individually, and then independently applied them to the SVM algorithm. This process required that three independent decisions are combined for classification.

This system rated 96.4% accuracy using OAO method and a distributed data fusion scheme.

In 2011 Bajwa el al. developed a wireless sensor classification system based on an axle detection technique using vibration sensors and detection sensors [17]. Vibration sensors detected each individual vehicles axle as a result of pavement structure vibration. Although four accelerometers were implanted in the pavement, data from only three was gathered due to the failure of the fourth.

A magnetometer served as a detection sensor excited by the magnetic field changes of a passing vehicle. A limited number of sensors was installed at fixed distance from one another, and then used to calculate vehicle speed based on arrival times at each of the two sensors.

To mitigate the effect of acoustic noise on the input, a third-order LP Butterworth filter was used with a cutoff frequency of 50Hz.

The vibration sensor output is normalized to maintain the signal below 1 given no axle is detected. The signal is then squared to obtain the power, which further increases the signal to noise ratio and is, then, smoothed by a moving average.

The researchers tested a sample of 53 trucks and reported results for accurate axle count. Accuracy ranged from 86.8% to 90.6% for axle count when using a single sensor; 100% accuracy for axle count was achieved when sensors were combined. The researchers did not provide test results for actual vehicle classification.

2.4 Current technologies

Many DOTs use AVC to count and classify vehicle types travelling their highway systems. This fact is particularly important for several reasons—most importantly because VMT aids in capacity planning for new highways based on the highways currently used. Data is also used for pavement design. Big truck counts are of higher significance than smaller vehicles for pavement design, as they greatly impact pavement degradation. Real-time AVC data can also be used for enforcement and toll collection.

Current technology for DOT classification and vehicle count includes ever-popular axle-spacing techniques made from AVC products. These systems have proven reliable and practical. Either piezoelectric sensors with inductive loops or road tubes are commonly used.

In 2010, 23 states reported motorcycles as part of both permanent and temporary vehicle counts. The most frequently used technologies those employing axle spacing with piezoelectric sensors and inductive loops or road tubes. Other technologies included video, radar, and signature-based inductive loops [18].

2.5 Axle spacing vehicle classification systems

As previously mentioned, axle spacing vehicle classification technologies are frequently employed by DOTs . Technological maturity and extensive testing afford these agencies a thorough understanding of the procedure and inherent errors.

2.5.1 Road tubes

Road tubes are fashioned from rubber and terminated on one side by a plug to prohibit air leakage. The other side is connected to a vehicle detector. As a vehicle passes over the road tube, an air pulse travels through the tube towards the detector. The detector is equipped with a sensor that detects the pulse which is usually a piezoelectric film sensor, air switches or loop [19].

In this way, vehicle classification is accomplished using two road tubes installed within a given distance, as shown in figure 2.1. Passing vehicles will impact the first road tube with their front axle at time “t”, and then impact the second road tube using the same axle at time "t+Δt". Knowing the time difference Δt and the distance between the two road tubes, vehicle speed can be calculated.

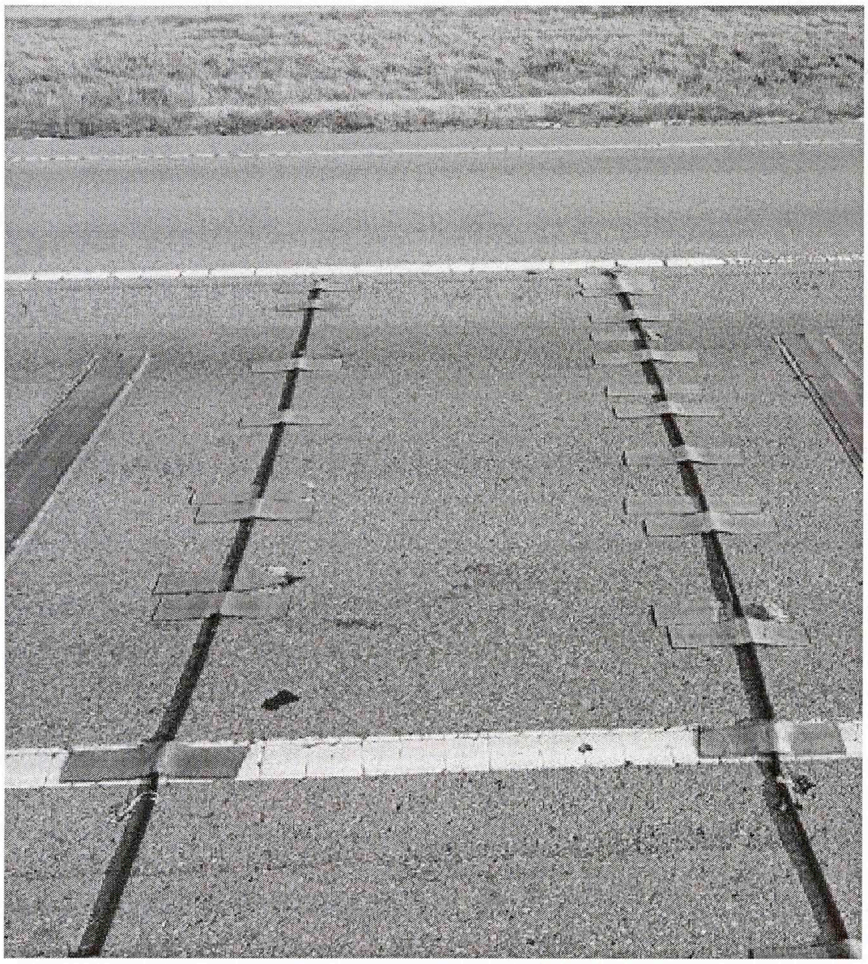


Figure 2-1 Pneumatic road tubes installed on Oklahoma highway 69 for temporary classification testing

Axle spacing can be found using vehicle speed and the time at which the consequent axles impact the road tubes. These parameters can then be used to classify the vehicle based on a preset margins determined by axle spacing.

Vehicle classification based on road tubes is highly appealing for various reasons, including low complexity, inexpensive cost, and ease of installation. Because road tubes are fixed over the road, there is no need to cut the pavement. However, these systems have several problems, as well, including a tendency to miss motorcycles tire due to low sensitivity [18]. For this, road tubes often under classify Class1 vehicles. Distinguishing adjacent axles in close proximity to one another on vehicles traveling at high speeds is challenging due to limited detection frequency of the detector. Axle over-count can occur, too, especially if a road tube wasn't affixed properly to the pavement. In this case, a heavy axle hitting the tube can bounce the road tube at such a force that a fake pulse is interpreted as a new axle. Other axle spacing problems, e.g., vehicle combining or splitting, are common, as well [19]. Road tube technology is typically used in temporary and low-density traffic conditions. Because road tubes are usually affixed on top the pavement, tubes will become loose after frequently being passed by vehicles.

2.5.2 Piezoelectric axle sensors with and without a loop

A piezoelectric axle sensor is comprised of piezoelectric material that produces an electric charge when pressured and is translated to voltage that, in turn, is detected by an ADC [20]. Two full-lane or half-lane piezoelectric sensors are often installed with a predetermined distance separating them. When a vehicle axle impacts a piezo-axle sensor, a voltage pulse is produced and then detected, marking arrival time of the axle, as shown in figure 2.2. When a vehicle triggers both sensors, speed and inter-axle distance are calculated using a procedure similar to the one described in the previous subsection for road tubes.

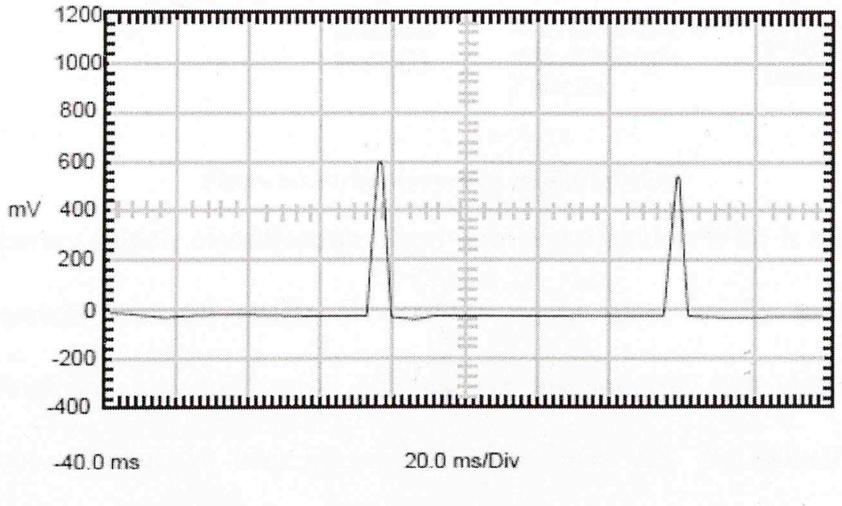


Figure 2-2 Pulses produced by axles hitting a piezoelectric sensor [22]

Two types of vehicle classification and Weigh in Motion (WIM) are achieved using piezoelectric axle sensor technology:

- Permanent vehicle classification: Sensors are embedded in the pavement with either one inductive loop located in-between the sensors or one inductive loop located in-between and one located before the combination in the upstream side of traffic. See figure 2.3. Embedding the sensor combination under the pavement provides protection and will extend their lifetime. Inductive loops are made of two or more rounds of metal through which current passes, generating an electromagnetic field and triggering vehicle detection. When a vehicle passes over the inductive loop, its metal disturbs the earth electromagnetic field and changes the inductive loop inductance [23]. When installed at a predetermined distance, two piezoelectric sensors can be used to measure vehicle speed.

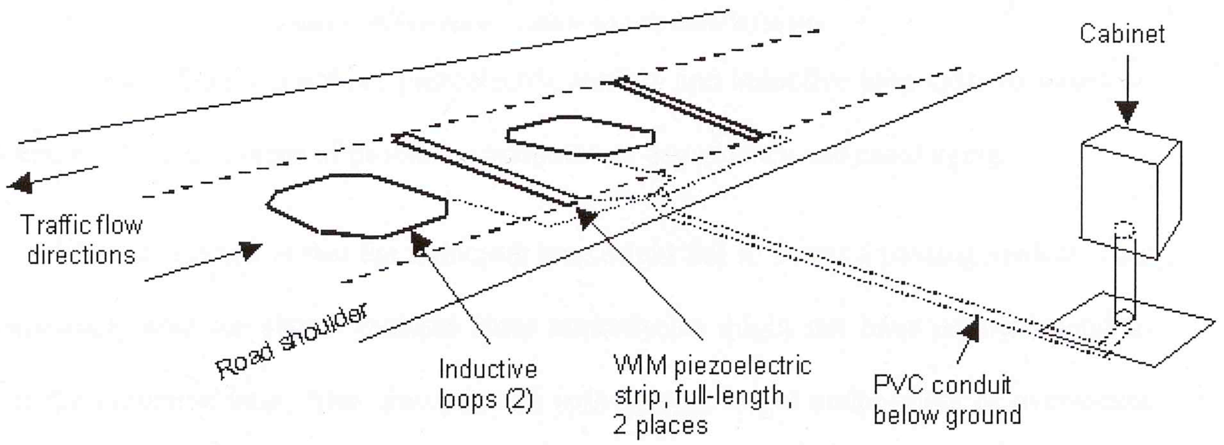


Figure 2-3 Permanent vehicle classification [24]

- Temporary vehicle classification: Short-term classification/WIM is sometimes desired for certain seasonal traffic abnormality or to asses traffic load on a certain road/highway for short period of time. For this purpose, two piezoelectric sensors without as inductive loop are utilized. See figure 2.3. An inductive loop in this configuration is clearly unfeasible due its large size and stringent installation requirements. Two sensors are flashed over the pavement surface and then connected to a road side detector/classifier. Although temporary vehicle classification installation is faster and less expensive installation than permanent vehicle classification—no need to cut the pavement to embed the sensor, sensors are damaged more quickly and installation cannot be maintained for a long period of time.

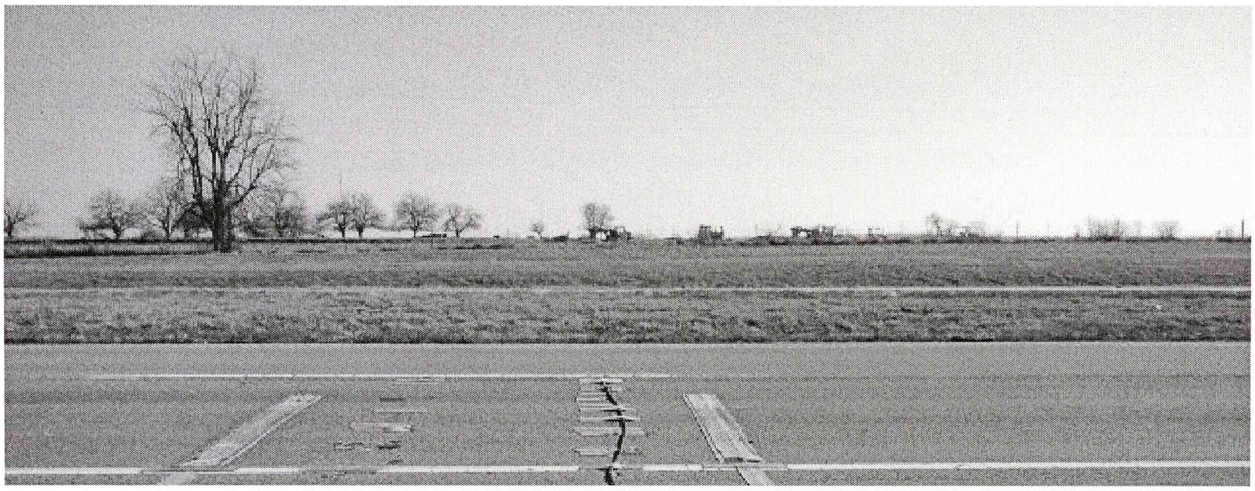


Figure 2-4 Temporary vehicle classification/WIM site

Although highly practical, piezoelectric sensors and inductive loop systems based on axle spacing have a number of problems: temperature dependence and rapid aging.

Of most concern is that the inductive loop might fail to detect a passing vehicle. This is particularly true for class1 vehicles since motorcycles might not have enough metal to trigger the inductive loop. Also, piezoelectric axle sensors might under-count or over-count the number of vehicle axles, especially as the sensors age. This will result in inaccurate classification. Piezoelectric sensors have a tendency to shift output with temperature, resulting in under-detection or over-detection of axles. Classification and detection algorithms are accountable for a number of mistakes, including vehicle splitting and combining. System calculations rely on the maximum axle spacing set in the algorithm and the level of traffic congestion. There might also be an overlap for axle spacing in vehicles belonging to different classes but with the same number of axles [6].

CHAPTER III

3 System development

This chapter details the development of the proposed vehicle classification system. The discussion includes system hardware, software, and a comprehensive system description. Hardware described includes sensors that were developed during each step and subsequent testing. Other hardware described includes data acquisition devices, computing devices, and cellular data connection. Software details include programs used for both development and testing. A complete system overview synthesizes all elements of this research.

3.1 System hardware general overview

System hardware is described in this section, including sensors and devices fabricated to build and test the vehicle classification system. Of utmost importance are the piezoelectric sensors. In this thesis both single element piezo-sensor and multi-element piezoelectric "all-lane" sensors are required. These are used to fabricate single-element and multi-element classification systems, which will be further described in the following subsections and chapters. Computing hardware will be described, and testing devices will be listed and explained. An illustration of validation systems will be provided.

3.2 Piezoelectric sensors

Piezoelectricity was discovered by Pierre Curie in 1880. In this phenomenon, electric charge accumulates in a piezoelectric material subsequent to its mechanical dimensions changing as a result of external mechanical force. These materials also change in dimension when placed in an electrical field. Examples of piezo materials include certain crystals, ceramics, and polymers.

Piezoelectric sensors are pressure-operated sensors that produce an electric charge in response to alteration in their dimensions. New advances in polymer technology enable the use of Poly-vinylidene Fluoride (PVDF) film for manufacturing piezoelectric cable sensors [25].

The purpose of this research is to investigate several piezoelectric sensors with varying features, including flexibility, size, durability against heavy weight, and appropriate output. This thesis will investigate two different layouts of piezoelectric sensors to develop our motorcycle classifier, namely single element and multi-element sensors. Single element sensors will be used for a single element vehicle classification system where a sensor made of one element will be placed diagonally for full lane traffic coverage. A multi-element sensor will be used for the development of a multi-element vehicle classification system where a sensor fabricated from multiple independent piezoelectric elements is placed diagonally for full lane traffic coverage.

Several market products were investigated. These include the following.

3.2.1 Piezo-ceramic sheets

Piezo-ceramic sheet sensors are manufactured by PIEZO SYSTEMS INC. and provide ample length that allows for flexible design of a multi-element sensor. See figure 3.1. However, this sensor is problematic in that the disc is extremely fragile and prone to easy breakage, making its use very inconvenient for traffic application since the sensor will be subject to heavy loads, e.g. trucks.

The sensors are also difficult to interface and harvest signal output, which might cause irregularities among different elements if interfacing is not carefully executed. Of note is that Piezo-ceramic sheet sensors are highly sensitive and easily excited by very slight fingertip pressure. With these considerations it is clear this sensor is not appropriate for single element development without an expensive an elaborate enclosure design.

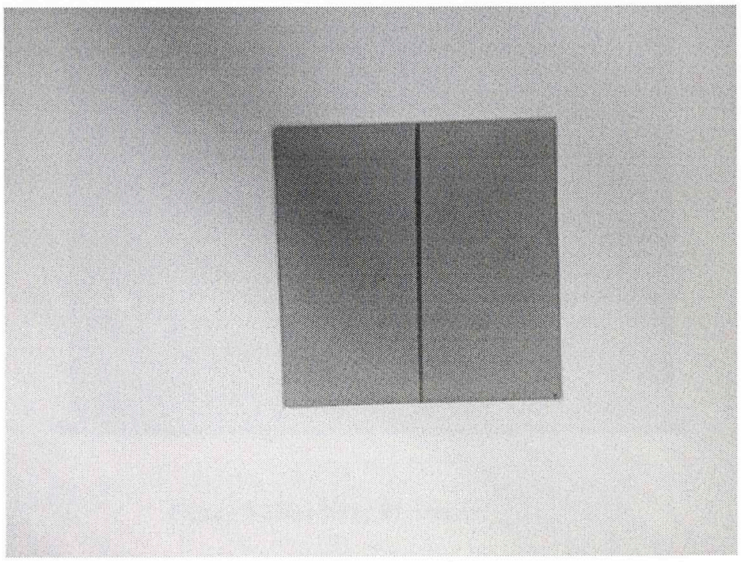


Figure 3-1Piezo-ceramic sheet

3.2.2 Roadtrax BL traffic sensor (short cut)

Roadtrax BL sensors are manufactured by Measurement Specialties and designed specifically for traffic use. See figure 3.2. As such, their features are highly desirable for traffic counting, classification, and WIM application. The sensor can withstand heavy weight and delivers clear output pulses when triggered by vehicle axles or tires. However, the sensor is relatively expensive, and the shortest length available is 6' [26]. Smaller lengths are available by special order, although there is an additional cost. While the price increases for multi-element application, the sensor's features make it a perfect candidate for single element application. This sensor is available by manufacturer at length of 6'6" which can cover an entire traffic lane when placed diagonally.

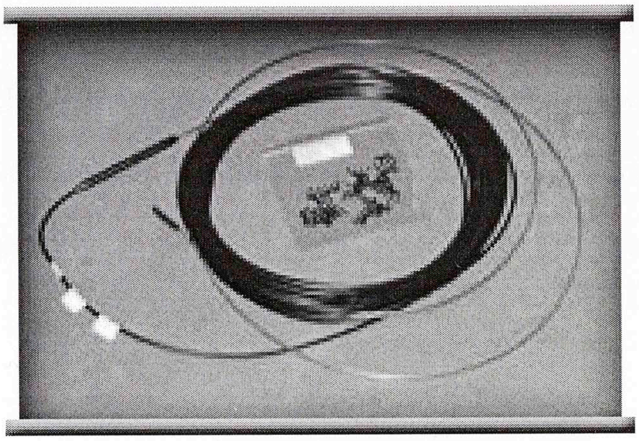


Figure 3-2Roadtrax BL sensor

3.2.3 Piezo-film sensor

Piezo-film sensors are sensitive piezoelectric elements used for various applications, such as vibration detection and voice conversion to electrical signal. Often the sensors are fragile and difficult to protect in roadway conditions. Their use as a multi-element sensor can be a challenge as a result of their interfacing schemes.

3.2.4 Piezoelectric polymer coaxial cable

The piezo-cable affords an extremely flexible solution. The sensor is manufactured by Measurement Specialties and can easily be formed to any desired length. The cable can then interface as a regular coaxial cable, as shown in figure (3). According to the manufacturer, the sensor has multiple applications that include vehicle classification and counting [27].

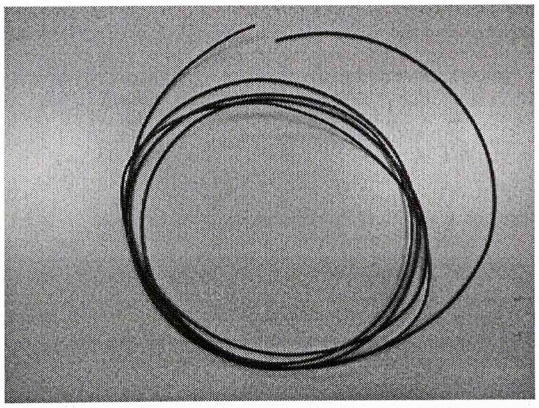


Figure 3-3piezoelectric polymer coaxial cable

Figure 3.4 illustrates the four layers of the cable: 1.) Stranded center core that acts as a conductor of the electrical output signal; 2.) Piezoelectric film tape that serves as the actual piezoelectric material; 3.) Copper braid; and 4.) Dielectric outer jacket, as illustrated in figure 3.4.

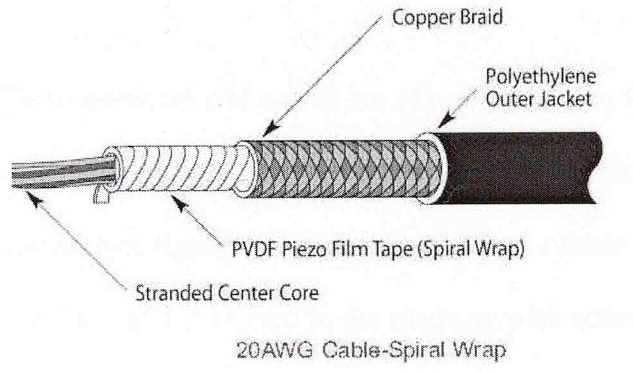


Figure 3-4Piezoelectric polymer coaxial cable layers [20]

The aforementioned features combined with a relatively inexpensive cost aided in the decision to select the cable for further consideration and testing for the development of the multi-element vehicle classification system.

3.3 Multi-element sensor fabrication

A standard 12' lane width must be entirely covered by the multi-element sensor to detect all passing tires and render a vehicle classification. By reducing the sensor to traffic direction angle (θ) allows additional separation between pulses from tires belonging to the same axle. However either the number of elements or the element length must be increased for full lane coverage.

A fixed element length of ~11.5" with 0.5" separation between consecutive sensors was chosen for development and testing. This measurement satisfies the need to provide appropriate vehicle width resolution. Thus, reducing θ ensures improved resolution, although additional elements— meaning more input channels to the data acquisition system—are required. Having a separation gap between sensors ensures adequate space for wire channeling and to reduce the probability of multiple sensors being impacted at the same time. With this in mind, it was determined that the angle should be set in a range of 45 to 50 degrees. Sixteen sensor elements are, then, required to cover the entire lane for 45°, and 15 are required for 50°.

Protection of the piezo-elements and cables are of major concern for preliminary field testing. To shield the cables and sensor, they were placed inside pocket tape specifically designed for traffic application. See figure 3.5. This is heavy duty rubber has a 4-inch pocket to encase both sensors and cables, and is affixed to the roadway with adhesive.

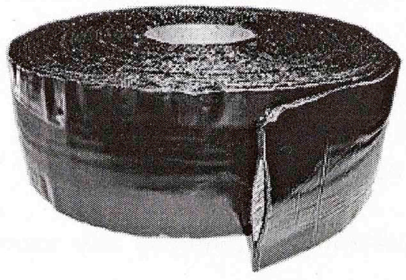


Figure 3-5 Heavy duty pocket tape.

To ensure reusability, an 8-foot aluminum/rubber road plate was used to affix the road pocket tape and the inserted multi-element sensors and connecting cables, as shown in figure 3.6. It is possible for each road plate to hold up to eight piezo-elements. Two plates are required for full lane coverage. Two parallel rows of pocket tape used—one containing piezo-elements and another containing cable that connect sensor-elements to the DAQ.

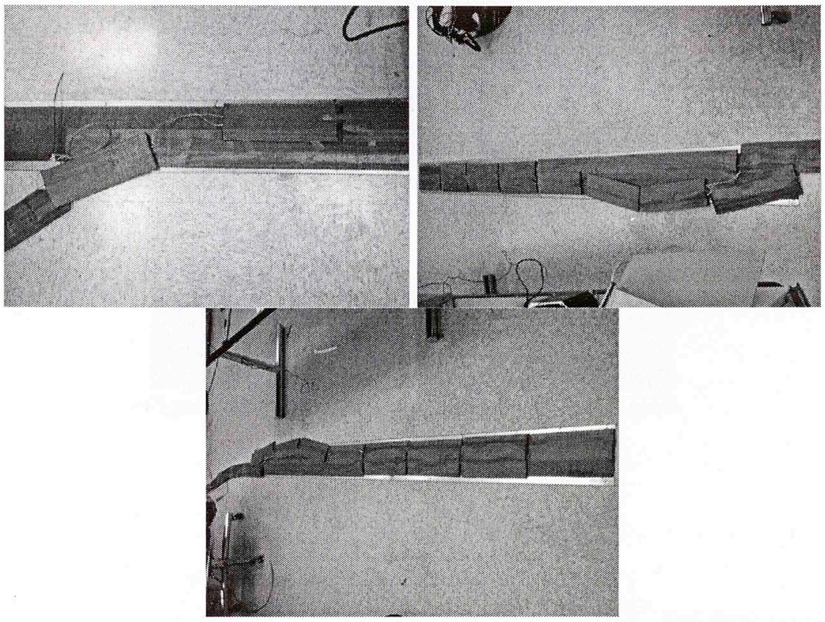


Figure 3-6 Multi-element sensor packaging procedure.

3.4 Temperature sensor

This section describes the integration of a temperature sensor with the classifier system. Piezoelectric sensors have the tendency to change their output signal relative to temperature. A temperature sensor was acquired and interfaced with the REECE device to monitor road surface temperature and account of any changes affecting piezoelectric sensor outputs, causing vehicle misclassifications.

Roadway surface temperatures in close proximity to where sensors are deployed should be measured. We chose the temperature probe 108 manufactured by Campbell. See figure 3.7. The temperature will be logged and then compared to output of the system in question. The four connecting wires which are excitation voltage, signal HI, signal return and ground. See figure 3.8.

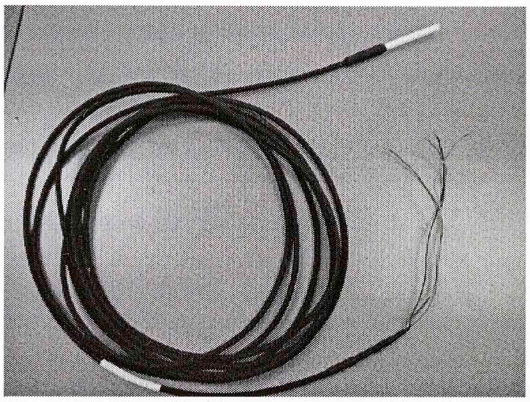


Figure 3-7 Piezoelectric polymer coaxial cable layers

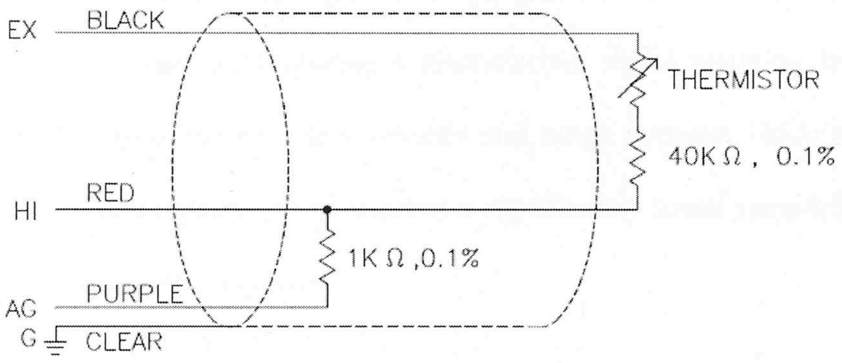


Figure 3-8 Temperature probe wiring scheme [28]

A continuous excitation voltage must be supplied to the probe through a wire. The voltage can then be measured from HI wire and a surface temperature can be computed using two equation that relate the voltage measurement and excitation voltage [28]. The measured voltage Vs and excitation voltage are related through the following equation:

$$\frac{Vs}{Vx} = \frac{1000}{Rs+40000+1000} \qquad \text{eq. (3.1)}$$

where Rs is the thermistor resistor, and 1k and 40k are fixed resistors. Rs can then be applied to the Steinhart-Hart equation 3.2 to calculate the temperature in Celsius:

$$T = \frac{1}{A+B(LnRs)+C(LnRs)^3} - 273.15 \qquad \text{eq. (3.2)}$$

where A, B and C are coefficients related to the thermistor: A = 8.271111e-4, B = 2.08802e-2, and C = 8.0592e-8.

3.5 Testing and development data acquisition units

This section describes the external data acquisition units (DAQs), which acquire voltage signals coming from the sensors. Understanding the behavior of sensor signals and their characteristics is essential to the development process, as the signals must be interpreted and distinguishable according to vehicle classification.

DAQs have a variety of specifications enabling different capabilities, e.g. sampling rate, input range, number of channels, and DAQ mode (differential or single ended). The most important among these is sampling rate. Sampling rate takes into consideration acquired signal frequency. For example, acquiring a piezoelectric signal resulting from a passing vehicle is dependent upon the vehicle's velocity and range between 1kS/s and 10ks. The signal acquired from temperature probe requires a significantly lower sampling rate because temperature changes are relatively slow.

Three DAQs were used in this project.

3.5.1 National Instruments NI-9215:

The National Instruments NI-9215 DAQ has four inputs, as shown in figure 3.9, and can

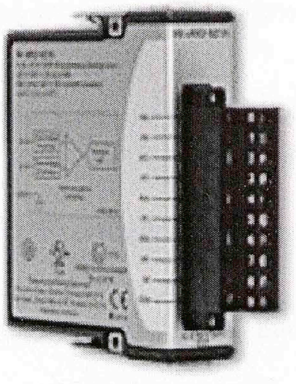


Figure 3-9 NI-9215 data acquisition unit [29]

Simultaneously perform sampling rate up to 100KS/s on the four analog inputs. Its resolution is 16bit, and maximum voltage range is -10v to 10v with maximum accuracy of 0.003v [29].

3.5.2 National Instruments NI-9205

This National Instruments NI-9205 DAQ has 32 single ended (SE) or 16 differential inputs, as shown in figure 3.10, and can perform a maximum of 250KS/s sampling rate

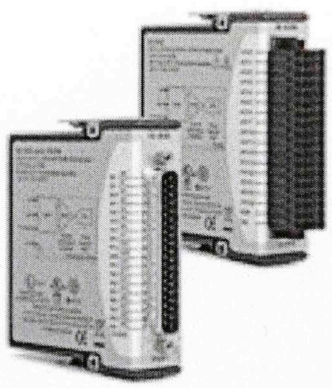


Figure 3-10 NI-9215 data acquisition unit [30]

divided among a number of active input channels. Its resolution is 16bit, and maximum voltage range is -10v to 10v with a maximum accuracy of 6220μv [30].

3.5.3 REECE Helios embedded DAQ:

The REECE DAQ will be further detailed in the following section.

3.6 Roadside Extensible Embedded Computing Equipment (REESE)-Helios

In this section the Diamond Systems Helios computing system interface and related capabilities pertinent to this thesis are briefly explained.

The RRECE device was initially developed as part of a project funded by the Oklahoma Transportation Center (OTC) in 2005-2006. The goal was to develop wireless access to Oklahoma Department Of Transportation (ODOT) traffic data collection sites, enabling automatic vehicle classification (AVC) and weight-in-motion (WIM). The device was developed by Dr. Refai research team at the University of Oklahoma. The REECE has the capability to access a cellular network using either a 3G or 4G wireless modem, and then transfer data back and forth to servers in real time. Diamond Systems Prometheus served as the basis for the computing system, and development advanced to incorporate the new generation of Helios.

Helios is an embedded system designed by Diamond Systems. See figure 3.11. The compact, embedded computer is able to run a number of operating systems, including Linux.

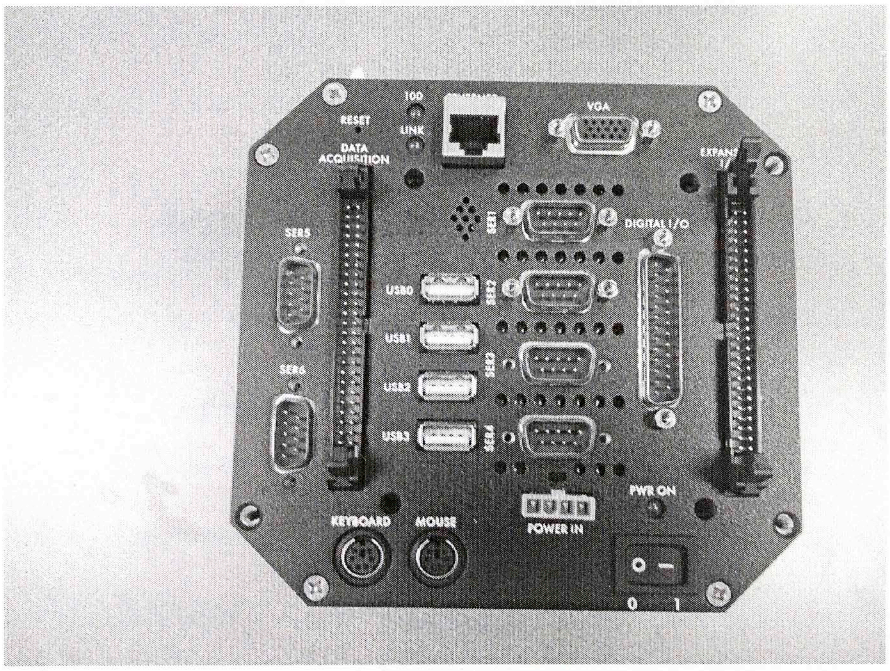
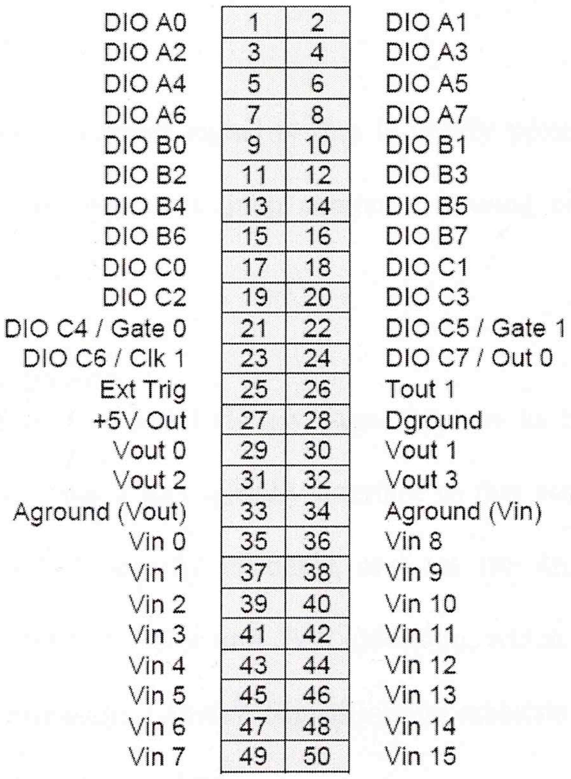


Figure 3-11 Helios computing system

Helios has an 800 vortex86DX CPU and is equipped with six VGA ports; PS/2 mouse and keyboard ports; RS-232 ports; four USB ports; 40 digital programmable I/O lines; four analog outputs; and Ethernet port with 10/100 Ethernet circuit integrated into the processor. Most important among Helios I/O ports is the 16 analog inputs, further detailed below [31].

Helios includes a built-in DAQ unit that can operate with 16 single ended (SE) channels or 8 differential channels. Helios DAQ scans input channels sequentially and has a maximum sampling rate of 250kS/s, although this rate is divided among input channels when more than one channel is used. For example; when using 16 SE input channels, the rate drops to 15.625kS/s.

Signal input is connected to Helios DAQ through a 50-pin male header on the I/O module The header is comprised of all 16 analog input channels; four analog outputs; grounds; voltage out; and digital I/Os [32]. See figure 3.12.



DIO A0	1	2	DIO A1
DIO A2	3	4	DIO A3
DIO A4	5	6	DIO A5
DIO A6	7	8	DIO A7
DIO B0	9	10	DIO B1
DIO B2	11	12	DIO B3
DIO B4	13	14	DIO B5
DIO B6	15	16	DIO B7
DIO C0	17	18	DIO C1
DIO C2	19	20	DIO C3
DIO C4 / Gate 0	21	22	DIO C5 / Gate 1
DIO C6 / Clk 1	23	24	DIO C7 / Out 0
Ext Trig	25	26	Tout 1
+5V Out	27	28	Dground
Vout 0	29	30	Vout 1
Vout 2	31	32	Vout 3
Aground (Vout)	33	34	Aground (Vin)
Vin 0	35	36	Vin 8
Vin 1	37	38	Vin 9
Vin 2	39	40	Vin 10
Vin 3	41	42	Vin 11
Vin 4	43	44	Vin 12
Vin 5	45	46	Vin 13
Vin 6	47	48	Vin 14
Vin 7	49	50	Vin 15

Figure 3-12 DAQ I/O connector

The aforementioned characteristics make the REECE device Helios an appealing platform for the development of a vehicle classification embedded system. It is currently deployed at 80 ODOT data collection sites, making the deployment of the single/multi-element classifier into these sites inexpensive and feasible

3.7 Cellular broadband modem

Either a 3G or 4G broadband modem can be used to connect the REECE device to the Internet, allowing remote access and real-time operations. This is desirable when performing real-time per vehicle reporting. Modems for this project were acquired from Sprint cellular service provider. Drivers and chat files associated with the cellular base station must be installed on the REECE to ensure Internet connectivity.

3.8 Software used through development

This section provides a general overview of software used throughout system development and testing. The classification algorithm will be described in further detail in Chapter IV.

3.8.1 Testing software

Software used during sensors signal testing is briefly presented in this subsection. Software was primarily executed on Laptop computer running either Windows Vista or Windows 7.

3.8.1.1 Labview SignalExpress:

National Instrument provided Labview SignalExpress to operate their proprietary DAQs. The software has a user-friendly interface so that users can easily choose the preferred DAQ, select specific channels, and set the sampling rate. Also, data collected can be saved to either to a .lvm extension, which is Labview specific file type, or to a .txt extension. Labview SignalExpress maintains a continuous graphical plot of signals as they are acquired.

3.8.1.2 Matlab:

Matlab is used for several purposes: analyze signals collected from various sensors; validate DAQ accurate operation; study piezoelectric sensors signal to design the classification algorithm thresholds accurately; monitor temperature sensor output; and perform correlation studies between different vehicles signals.

3.8.2 Software used in the on REECE coding process

A number of software programs aided in the development of the proposed vehicle classification system. These ranged from operating system and programs on the REECE device itself to programs used on a PC to support the development process. A brief description of programs and tools are offered below.

3.8.2.1 REECE device Operating System (OS):

The Linux operating system (OS) was selected for the REECE device because it is an embedded system, supporting society and its stability. Kernel version 2.6.37 was used. A special build was done for REECE device applications using "buildroot" software. This build includes tools and drivers needed for REECE device to operate properly. Examples of these include the built-in DAQ driver, cellular broadband modem driver, and chat file used by broadband modem to communicate with the cellular base-station. The REECE OS contains files and programs developed for REECE device connectivity from the server and VPN to server settings. Please refer to Appendix A for additional information regarding the VPN network and REECE network setup.

3.8.2.2 VirtualBox:

A host OS is needed to contain C compiler that is compatible with REECE OS because the Linux OS was specifically built for REECE. Thus, VirtualBox serves as a tool with Linux Debian OS so that C compiler for REECE OS is contained.

VirtualBox is cross-platform software used to emulate a computing environment for an operating system. The program reserves hardware resources running Virtual Machines (VM). VirtualBox can execute several VMs on a single host OS. The software can support several versions of Windows, Linux, Mac OS X, and Solaris and also 32-bit and 64-bit Oss [33].

3.8.2.3 Putty:

Putty client software provides the capability for SSH, Telnet, Serial, and TCP connections. This tool is extremely useful for remotely accessing and controlling Linux machines; REECE device and Linux server.

3.9 Overall system description

This section offers a description for both single- and multi-element systems. The layout, logic, and configuration of each system are detailed below.

3.9.1 Single element system

The single element vehicle classification system is comprised of one piezoelectric sensor, a DAQ, and the embedded computing system. See figure 3.13.

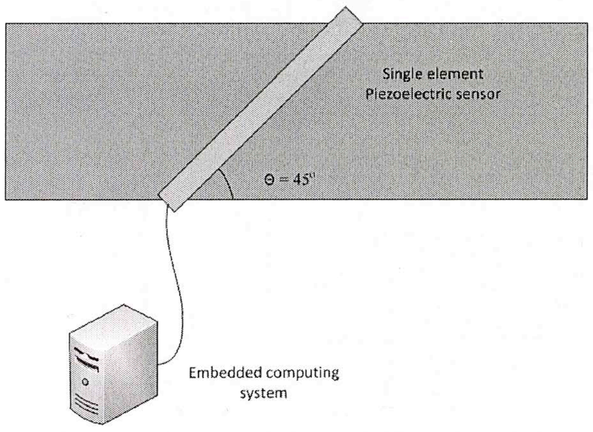


Figure 3-13 Single element vehicle classification system overview

It is imperative for the piezoelectric sensor to be flexible, available in various sizes, withstand heavy weight, and provides appropriate output. As previously mentioned, the sensor must cover the entire traffic lane diagonally at a certain degree, namely 45^{o} for this thesis. As such, length constraints must be met. Roadtrax BL manufactured by Measurement Specialties was selected since the material was designed for sensing traffic. The piezoelectric sensor is diagonally placed across a lane of traffic, ensuring that each tire of a passing vehicle produces a pulse on the piezoelectric sensor output. Output signals from the piezoelectric sensor are analyzed through a feature extraction algorithm where pulses from each passing vehicle are detected and then parameters are computed.

Data is then sent to classification phase of the algorithm to aid in vehicle classification. In this way, motorcycle Class1 vehicles can accurately classified, as they're the only class with two tires. For the remaining 13 FHWA classes, vehicle velocity is required to distinguish among classes with the same number of tires. Tire spacing is required to set thresholds between different classes with the same number of tires. Vehicle track width is needed to calculate vehicle velocity. See figure 3.14.

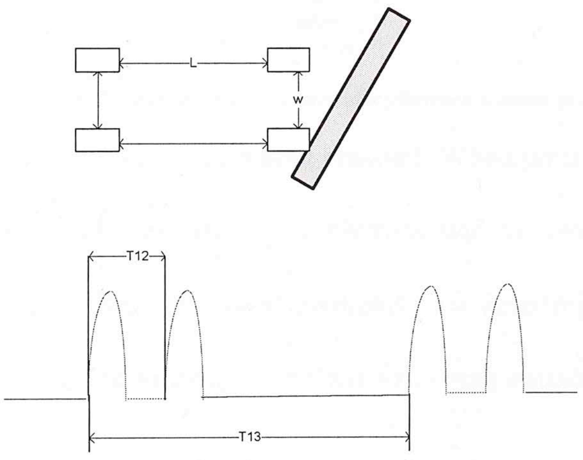


Figure 3-14 Example of piezo-sensor expected output when triggered by a passenger vehicle

Using vehicle track width w, velocity V and axle spacing L can be determined using the following equations:

$$V = [w * \cot(\theta)] / T12 \qquad \text{eq. (3.1)}$$

$$L = V * T13 \qquad \text{eq. (3.2)}$$

where: θ is angle between traffic direction and sensor;

T12 is time duration between beginning of first pulse and second pulse; and

T13 is time duration between beginning of first pulse and third pulse.

3.9.2 Multi-element system

The multi-element vehicle classification system is comprised of several piezoelectric sensor elements, multi-channel DAQ, and a computing system. See figure 3.15.

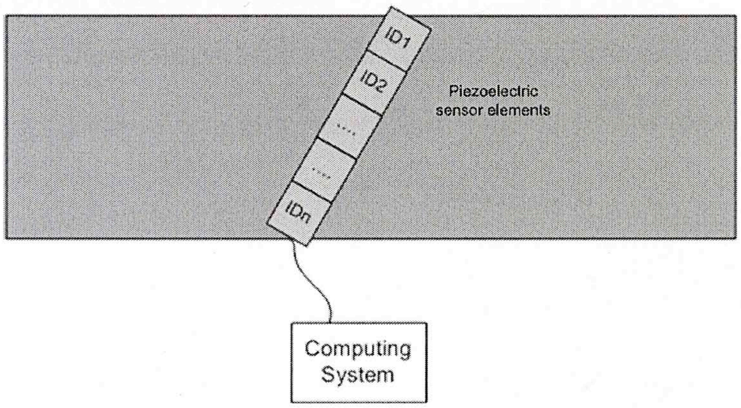


Figure 3-15 Single element vehicle classification system overview

Each element is connected to an output channel. When pressured, an element provides a higher output than adjacent elements. Each element has its own ID numbered from 1 to 16—each represents one of the 16 sensor elements. To determine passing vehicle width, assuming sensor elements n and m where m>n, the following equations are used.

$$m - n = w' \qquad \text{eq. (3.3)}$$

where w' is the hypotenuse of a right triangle in which the cathetus across from angle θ is the vehicle width w.

Angle θ represents the angle created between the piezoelectric sensor and direction of the traffic. Vehicle width can then be simply calculated by:

$$w = w' \,.\, \sin(\theta) \qquad \text{eq. (3.4)}$$

Width measurement resolution is sin(θ) feet, since the separation between every two adjacent sensor elements is one foot. This width is used to calculate vehicle velocity and axle spacing, as described in the previous section. The multi-element system requires more complex computing than the single element system because signals from 16 channels are simultaneously acquired. Voltage level of each channel must also be investigated to differentiate sensor elements that are triggered from others. For these reasons, the computer system processing power must be increased.

CHAPTER IV

4 Program architecture and vehicle classification algorithm

This chapter describes the vehicle classification algorithm and program architecture implemented on the REECE device. First a general description of software architecture will be provided, followed by additional detail for each module/phase. The difference between single- and multi-element systems will be discussed.

4.1 Overall program architecture

This section provides a detailed description of program architecture developed for this project. The architecture can be defined as the path taken from the commencement of raw data acquisition by DAQ to the decision stage of the algorithm's classification phase. The vehicle classification algorithm can be separated into two phases: feature extraction phase and classification phase.

The architecture for a single-element vehicle classification system is nearly the same as the architecture for a multi-element vehicle classification, with few exceptions. Algorithm differences between single- and multi-element systems will be explained in subsection of this chapter. Program development was implemented for REECE using C. C is a programing language of choice for embedded systems. It has flexible structure with various functions [34]. Figure 4.1 illustrates the overall software architecture.

The program is comprised of three primary modules: data acquisition; socket server and initial processing; and feature extraction and classification.

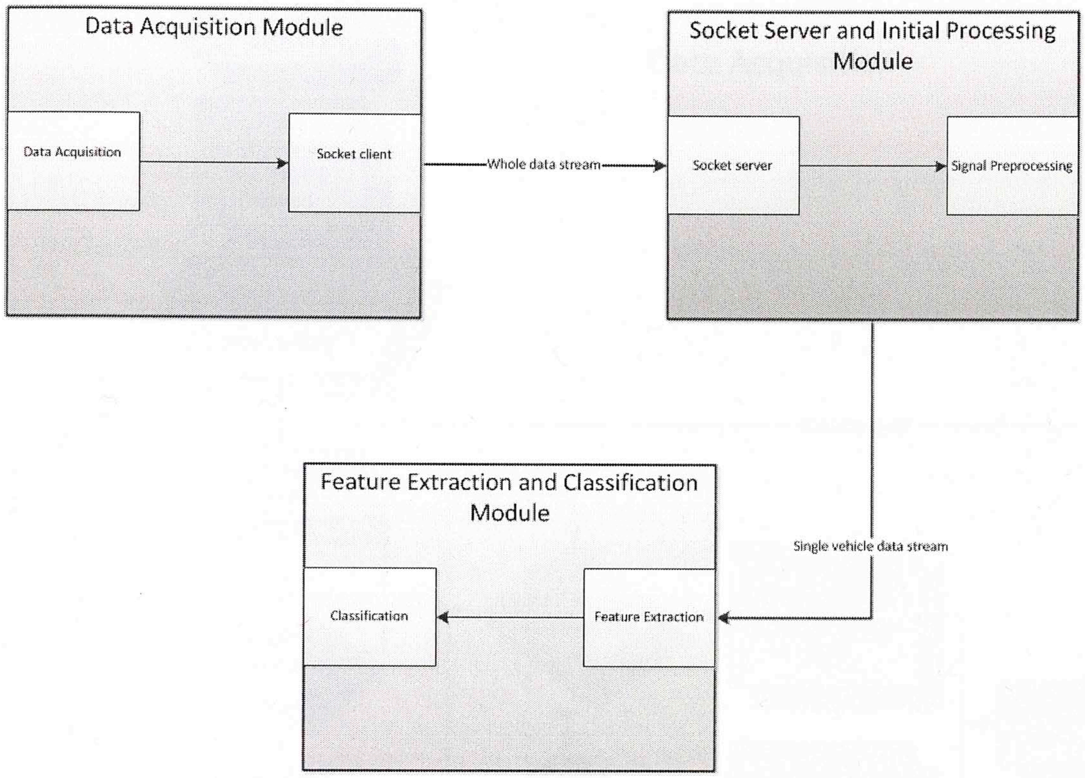
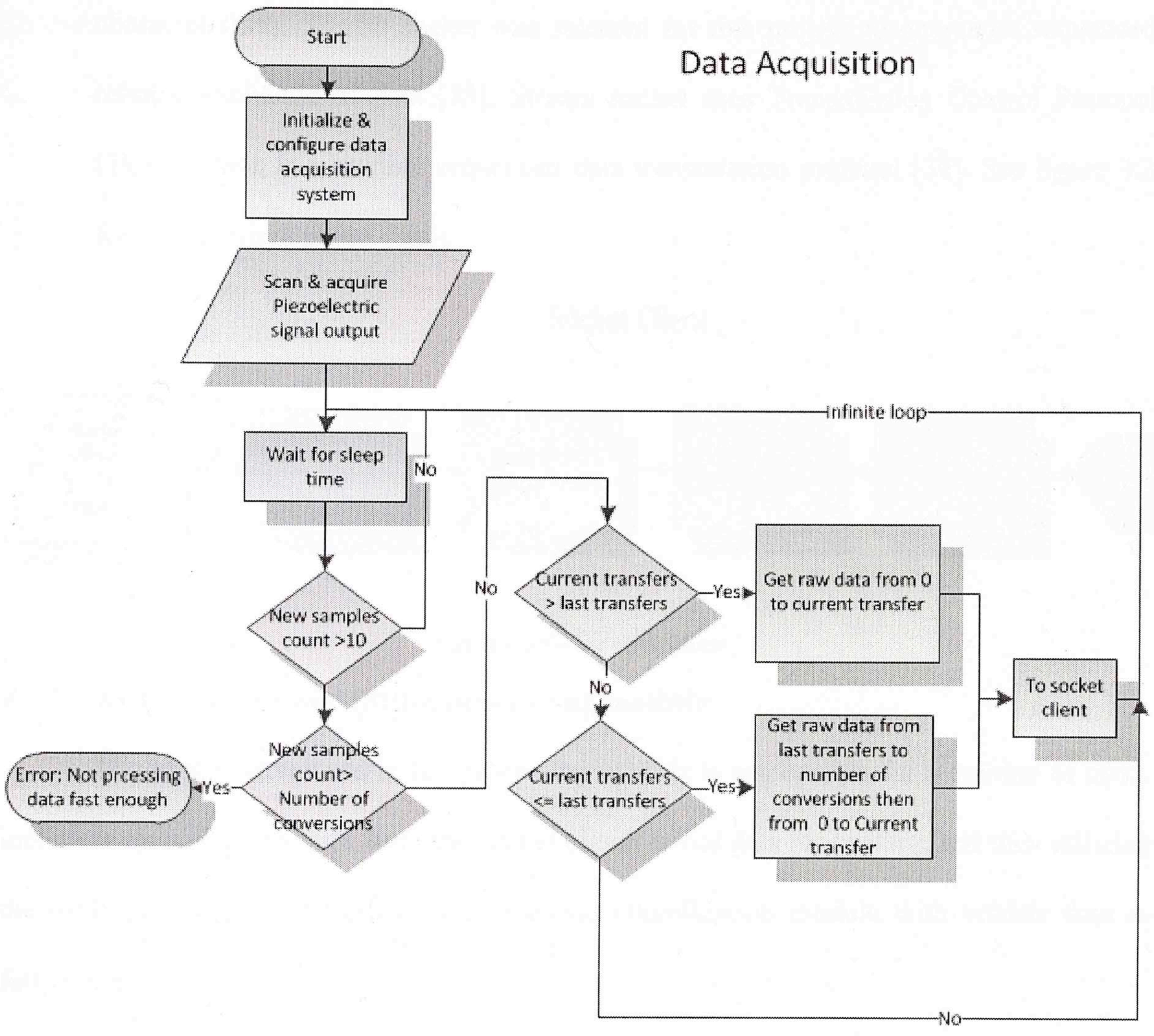


Figure 4-1 Overall software architecture

4.1.1 Data acquisition module

This part of the program includes the actual data acquisition phase and a socket client phase. It configures the DAQ, collects raw data, formats it in character form and sends it to the next module using a Linux socket. The two phases and their responsibilities are highlighted below:

- Data acquisition phase: This phase or sub-module is responsible for the collection of raw data. The REECE DAQ is used for piezoelectric sensor output sampling. The program includes initiating the board, configuring the sampling parameters, and making sure data is not lost due to slow processing. Major sampling parameters are chosen as follows. Sampling rate: 1Ks/s, continuous cyclic sampling; and channels used: one among other configurations related to the DAQ board itself. The phase flow graph and details are depicted in figure 4.2.



Data Acquisition
Start
Initialize & configure data acquisition system
Scan & acquire Piezoelectric signal output
Infinite loop
Wait for sleep time
No
New samples count >10
New samples count> Number of conversions
Yes
Error: Not prcessing data fast enough
No
Current transfers > last transfers
Yes
Get raw data from 0 to current transfer
No
Current transfers <= last transfers
Yes
Get raw data from last transfers to number of conversions then from 0 to Current transfer
To socket client
No

Figure 4-2 Data acquisition phase

- Socket client: Socket sends raw data from the DAQ module to the socket server and initial processing module. Using multiple processes takes advantage of the Linux multi-tasking/threading property so that the data processing load can be distributed among different processes. In this way uninterrupted data acquisition is ensured. During the data acquisition phase, data is fetched from the DAQ buffer on regular basis. If all data processing relied on the same process used by the data acquisition with no multi-threading the result could be latency and data loss.

Sockets are the most popular method for inter-process communication in Linux based operating systems. The client side of the server is responsible for sending data in character form. Stream socket was selected for this project to guarantee sequenced reliable exchange of data [35]. Stream socket uses Transmission Control Protocol (TCP), which is a reliable sequenced data transmission protocol [36]. See figure 4.3 for socket client phase detail.

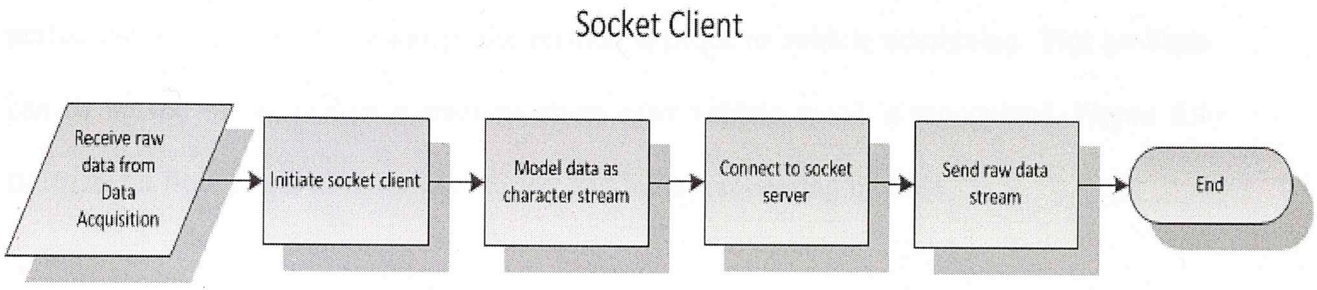


Figure 4-3 Socket client phase

4.1.2 Socket server and initial processing module

The socket server and initial processing module is responsible for a number of tasks, including receiving raw data from the socket client; initial data processing; and then utilizing the multi-threading, call feature extraction and classification module with vehicle data as function input.

The socket server first listens for a connection with the client. When client connects and begins a data streaming, the socket server continuously receives data and converts it from character form to floating point for processing feasibility. The purpose for this initial processing is to isolate single or multiple vehicle data so that it can be sent to the feature extraction and classification algorithm for further processing. Zeros are then suppressed from occupying more resources in the system.

While raw data is being received from the socket client the socket server tests the samples values. If they pass certain negative or positive threshold then detection flag is raised and samples are stored as vehicle data. Furthermore, if a certain number of zeros equal to two seconds is surpassed, vehicle detection ceases and stored vehicle data is called from the feature extraction and classification module as input.

This maximum axle spacing value works well without vehicle splitting for vehicle speeds as low as 15mph and vehicle axle spacing of 40ft [6]. This is the same as the 2 second period mentioned earlier. However, the method is prone to vehicle combining. This problem can be solved in the feature extraction phase after vehicle speed is recognized. Figure 4.4 illustrates a flow graph of the socket server and initial processing module.

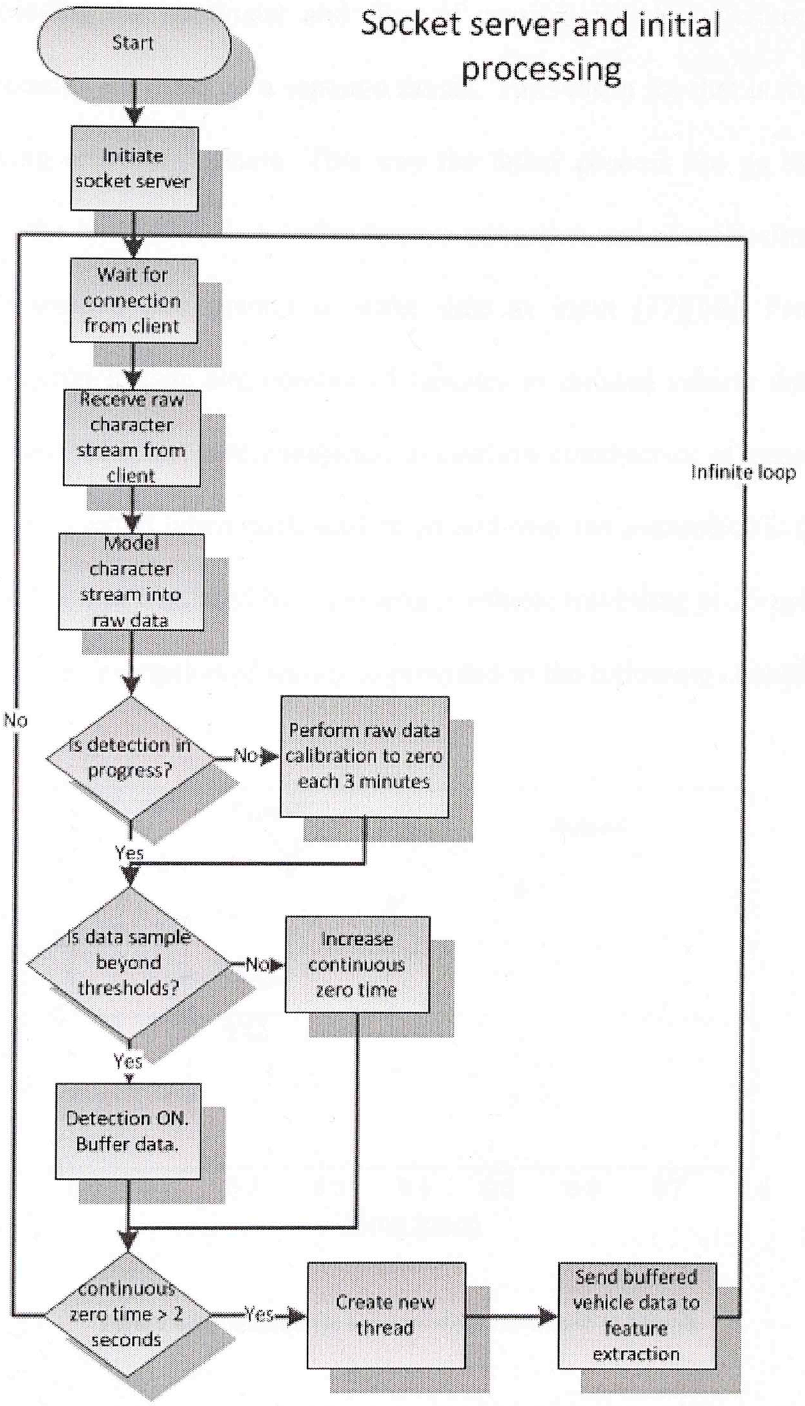

Socket server and initial processing
Start
Initiate socket server
Wait for connection from client
Receive raw character stream from client
Model character stream into raw data
Is detection in progress?
No
Perform raw data calibration to zero each 3 minutes
Yes
Is data sample beyond thresholds?
No
Increase continuous zero time
Yes
Detection ON. Buffer data.
continuous zero time > 2 seconds
Yes
Create new thread
Send buffered vehicle data to feature extraction
No
Infinite loop

Figure 4-4 Socket server and initial processing module

4.1.3 Feature extraction and classification module

The feature extraction and classification module represents the vehicle classification algorithm. This algorithm constitutes the logic used to process raw data into useful information, including the parameter and class of passing vehicles. Feature extraction and classification modules are done on a separate thread. The reason for that is to take advantage of the multitasking operating system. This way the father process can go back to raw data processing while the child process handles feature extraction and classification. Thread takes function to be executed and pointer to some data as input [37][38]. Feature extraction function, isolated vehicle data and number of samples in isolated vehicle data are passed to the new thread. Several tests were conducted to confirm consistency of expected and actual voltage output that resulted when each vehicle passed over the piezoelectric sensor. A visual example of sensor output triggered by a passenger vehicle travelling at 25mph is depicted in figure 4.5. A detailed description of testing is provided in the following chapter.

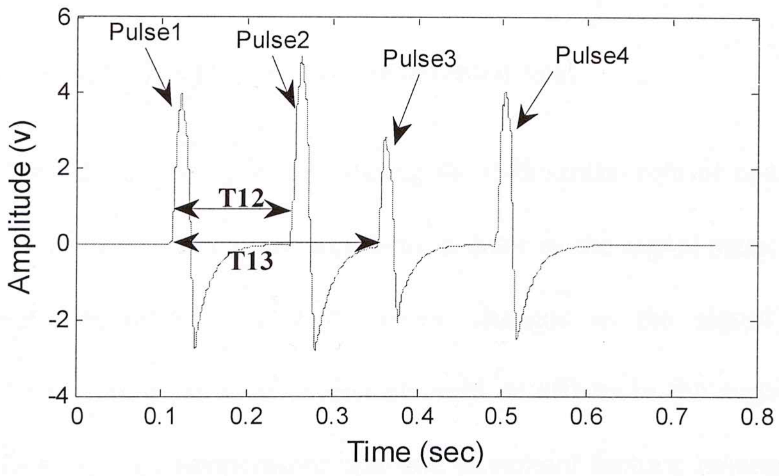


Figure 4-5 Piezoelectric sensor output for Class2 at 25mph

Each tire of a vehicle produces a unique pulse. Time difference between tires detection is equivalent to the distance between tires and axles. The algorithm must detect the pulses and the time between pulses, and then calculate vehicle velocity and tire /axle spacing. Then algorithm employs this information to make a decision regarding vehicle class. The vehicle classification algorithm is divided into two phases: feature extraction and classification.

4.1.3.1 Feature Extraction Phase.

The feature extraction phase is responsible for passing vehicle detection and subsequent extraction of characterizing information and specifications for classification purposes. After acquiring and preprocessing a piezoelectric response signal associated with one passing vehicle, pulses associated with different vehicle tires are identified. In this way the number of vehicle tires becomes known. Also, the time at which each pulse commences is detected, which facilitates recognition of time difference between pulses. By using former extracted vehicle information, vehicle velocity, tire/axle spacing, and total number of tires will be calculated/recognized. These results will then be advanced to the classification phase.

Two methods for pulse detection and extraction during the feature extraction phase were tested.

1) Pulse detection and extraction using differential rate:

   This method depended upon calculating the differential rate of consecutive acquired samples. The rate serves as an adaptable pointer to the signal state, meaning that the rate should be immune to both minor changes in the signal's amplitude not representing a pulse, as well as disturbances or offsets in the acquired value due to various factors, e.g., temperature bias and pavement factors, among others, affecting the sensor.

   Knowing piezoelectric signal behavior, we can pinpoint various patterns that mark pulses triggered by different types of vehicles at the operational range of velocities. Each differential rate value is represented by the amplitude difference between two consecutive piezoelectric sensor samples divided by the time difference:

$$r_i = \frac{S_i - S_{i-1}}{t_i - t_{i-1}} \qquad \text{eq. (4.1)}$$

The procedure contains two chief parts:

1- Marking the beginning of a pulse: A pulse is triggered when accumulated rate exceeds a value of 1000 within a 20-sample period. Using an accumulated rate of 1000 is relative to piezoelectric signal behavior when triggered by vehicles. The 20-sample tolerance period prevents system triggering by values acquired from long-term changes.

2- Marking the end of a pulse: Various conditions will pinpoint the end of a detected pulse.

   - When the rate is positive, the signal begins to rise. Thus, if more than six samples are negative, the signal is assured to be falling below pulse peak.
   - When the accumulated negative rate is higher than the accumulated positive rate, the signal has fallen back to its original statues.
   - When the rate is zero, or very close to zero, for a period of more than eight samples, the signal is considered stable at zero.

2) Pulse detection and extraction using signal thresholds:

The pulse detection and extraction technique uses predetermined thresholds marking the beginning and ending of a pulse. These thresholds are related to the behavior of piezoelectric signal in idle mode, i.e., no vehicle is present, and in active mode, i.e., vehicle passes over the sensor. Thresholds are empirically found through the testing of several passing vehicles. Similar to the differential rate method, this technique has two chief parts:

1- Marking the beginning of a pulse: If signal goes above threshold of 0.1v and remains there for over 0.01s, a new pulse is called and its beginning is recorded. Of note is that pulse is not detected immediately after signal passes the threshold so as to minimize detection of pulses due to transient changes in signal not related to passing vehicles.

2- Marking the end of a pulse: If signal is in idle state below detection threshold for over 0.1s and a pulse is detected, an end of pulse is recorded at the beginning of the idle state.

After a data set of 16 passenger vehicle runs were applied to both techniques, the signal thresholds technique was found to provide superior pulse detection, rendering it the technique used for the project detailed in this thesis. This is probably due to the fact that change of velocity cause change in pulse rise rate, pulse rise becomes sharper, leading to missing pulses at lower velocities. Regular sensor output calibration is added to the code to solve signal bias resulting from temperature and pavement effects, among others. During calibration the mean of sensor output signal is calculated on a regular basis and removed from received raw data. Calibration was employed because pulse detection and extraction using a signal thresholds technique is not inherently immune to signal bias as in pulse detection and extraction using differential rate. Please consult figure 4.6 for a detailed description of the feature extraction phase.

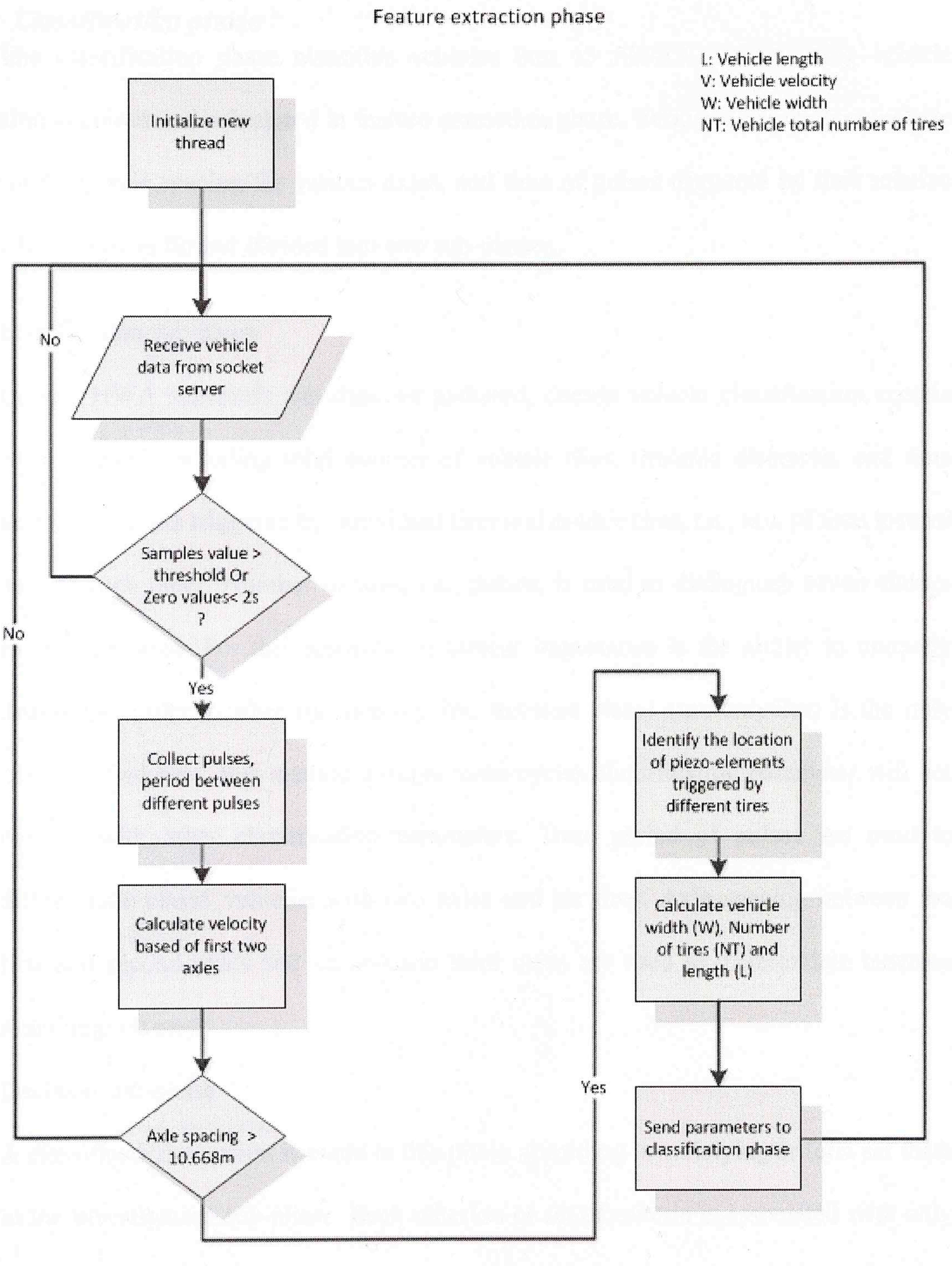

Feature extraction phase
L: Vehicle length
V: Vehicle velocity
W: Vehicle width
NT: Vehicle total number of tires
Initialize new thread
Receive vehicle data from socket server
No
Samples value > threshold Or Zero values< 2s ?
No
Yes
Collect pulses, period between different pulses
Calculate velocity based of first two axles
Axle spacing > 10.668m
Yes
Identify the location of piezo-elements triggered by different tires
Calculate vehicle width (W), Number of tires (NT) and length (L)
Send parameters to classification phase

Figure 4-6 Feature extraction phase

4.1.3.2 Classification phase

The classification phase classifies vehicles into 13 FHWA classes using vehicle information acquired and calculated in feature extraction phase. Required parameters include number of tires, axle spacing for various axles, and time of pulses triggered by tires relative to class. This phase is further divided into two sub-phases.

- Investigation sub-phase

  Using FHWA standards and data we gathered, certain vehicle classification criteria were defined, including total number of vehicle tires, tire/axle distances, and time interval of peaks triggered by individual tires and double tires, i.e., sets of tires located next to each other. Number of tires, i.e., pulses, is used to distinguish seven classes from each other. For this research, of utmost importance is the ability to uniquely distinguish pulse number for motorcycles. Because class1 (motorcycles) is the only one with two tires, this method ensures motorcycles classification parameter will not overlap with other classification parameters. Time period of pulses are used to differentiate class5 vehicles with two axles and six tires. Axle spacing between the first and second axles and second and third axles are used to differentiate between remaining classes.

- Decision sub-phase

  A classification decision is made in this phase according to satisfying criteria set forth in the investigation sub-phase. Each criterion or set of criteria is associated with only one vehicle class. This phase reports detected vehicle information and classification. See figures 4.7 and 4.8 for a detailed flow chart of the complete classification phase.

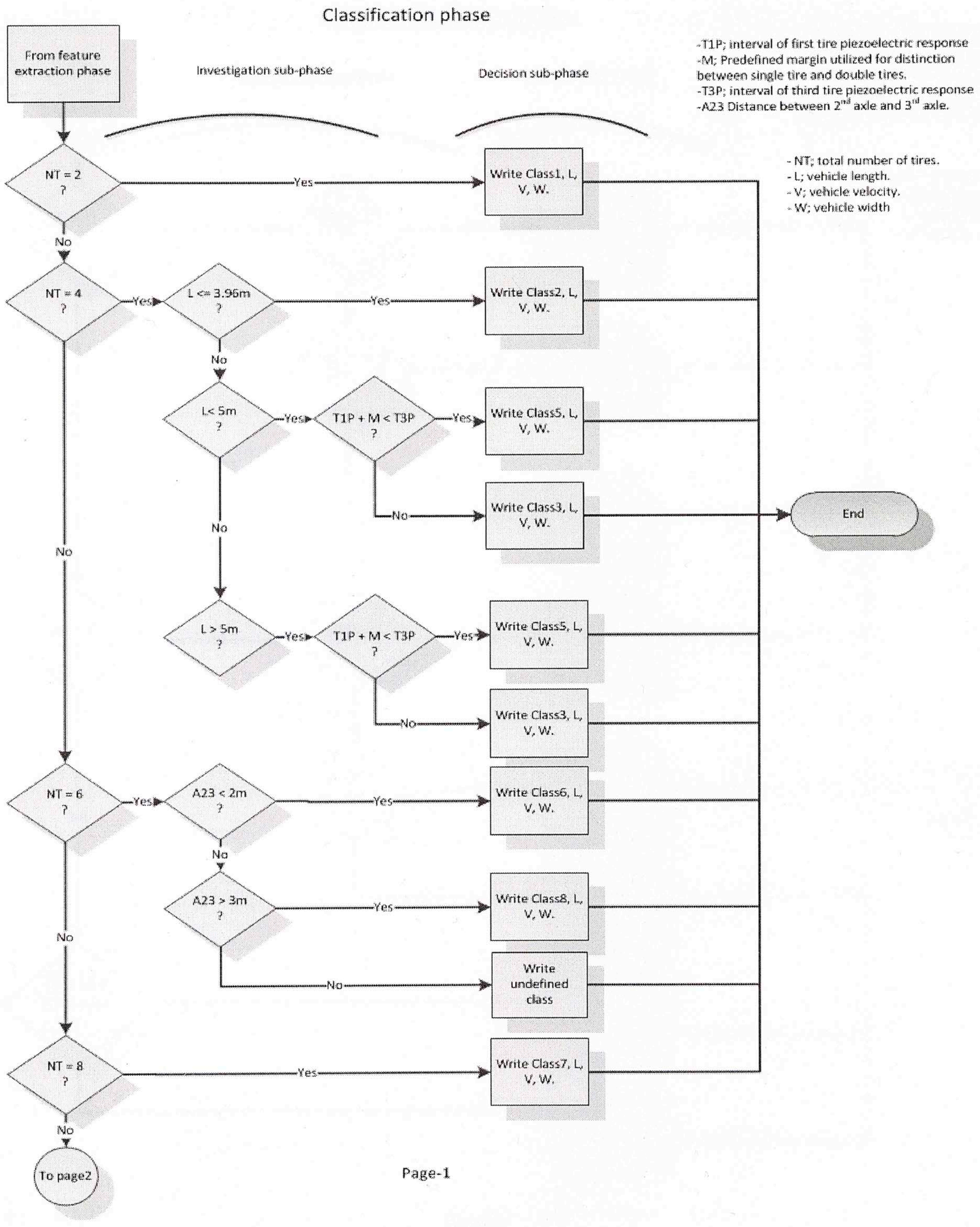


Figure 4-7 Classification phase

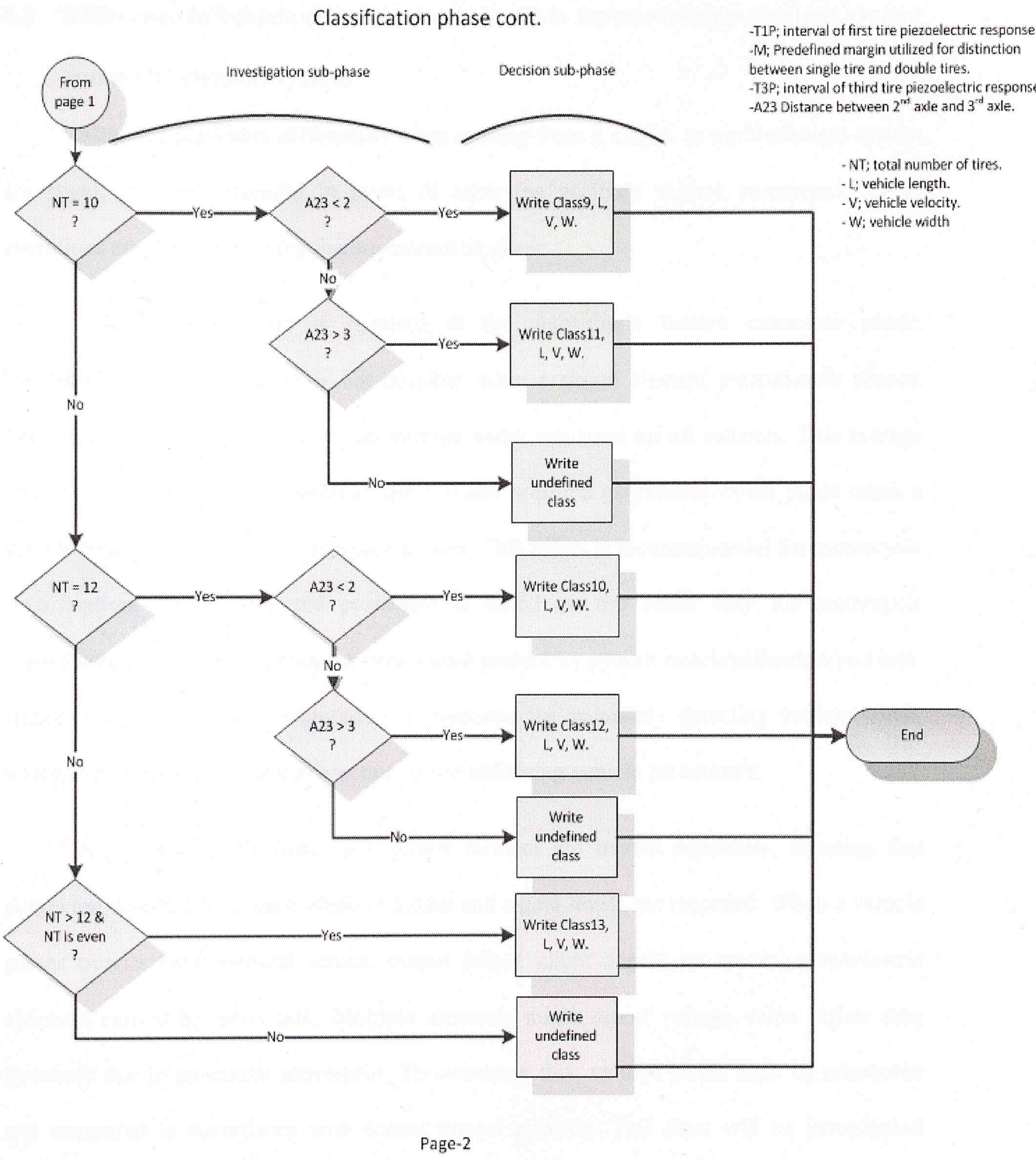


Figure 4-8 Classification phase cont.

4.2 Difference in vehicle classification algorithm between single element system and multi-element system

Although algorithm differences when moving from a single- to multi-element system are slight, they are essential in terms of achieving accurate vehicle parameters. These alterations are done only in the feature extraction phase.

Vehicle track width is required in the algorithm's feature extraction phase. Determining this information is not possible using a single element piezoelectric sensor. Thus, for a single-element system, an average width was used for all vehicles. This average width may cause an error in vehicle detected and acquired parameters which might cause a vehicle to be misclassified as an adjacent class. This effect is inconsequential for motorcycle classification, as the proposed procedure is based on tire count only for motorcycle classification. Nevertheless, the outcome could present an overall misclassification problem. Hence, a multi-element configuration is proposed for accurately detecting vehicle width, which in turn ensures accurate detection of the remaining vehicle parameters.

Algorithm signals from each sensor element are treated separately, meaning that pulses are detected from each element output and signal levels are recorded. When a vehicle passes over a multi-element sensor, output pulses might appear on several piezoelectric elements caused by cross talk. Multiple elements might report voltage value higher than threshold due to pavement movement. To overcome this, voltage levels must be monitored and compared in accordance with sensor impact position. This facet will be investigated further during multi-element sensor road testing.

Vehicles other than class1 have more than one tire in a single axle and will trigger more than one element at a certain separated distance. This distance represents vehicle track width times sin(θ). Given that one vehicle tire hits several sensors at the same time, a point closer to the higher amplitude element is chosen to represent the tire impact point. The position of this point is proportional to difference in voltage values between elements triggered.

Finally tire pulse width will be detected by single element impact or multiple elements impacts, and vehicle width will be calculated based on the distance between triggered elements. With the exception of this distinction between single- and multi-element sensing, feature extraction phase remains the same. No alterations are required during the classification phase.

CHAPTER V

5 Testing and results

Several tests of the single- and multi-element sensors were conducted. Initially, straight and diagonal piezoelectric single element sensor output was tested when triggered by different vehicles at different velocities—the purpose of which was to monitor and study the output of single element piezoelectric sensor. Results aided in confirming expected output and designing pick thresholds used for feature vehicle extraction. Secondly, tests were made to verify the newly developed feature extraction and classification algorithm. In this testing the stability of a single-element vehicle classification system was assessed for motorcycle and passenger car classification accuracy. Finally, a multi-element piezoelectric sensor was tested. The initial test of the fabricated sensor validated operationability and suggested design modification.

All road testing was performed at the University of Oklahoma-Tulsa campus with vehicles limited to a traveling speed of 30mph.

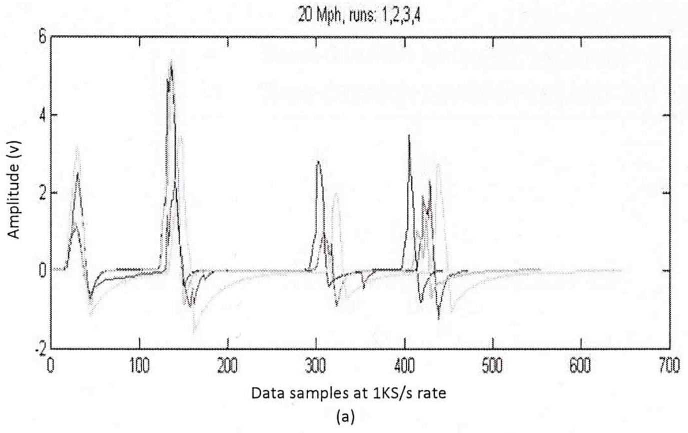
5.1 Single-element piezoelectric sensor testing

The objective of the single—element piezoelectric tests was to study single-element sensor output under various conditions. Results were useful in building the feature extraction and vehicle classification algorithm. Tests monitor changes in output signal resulting from number of reasons, including alterations in vehicle velocity and axle spacing; correlation between signals resulting from different vehicles driving at different velocities; and effects of changing vehicle width on both detected axle spacing and detected velocity. This section highlights a novel method for using track width-to-axle spacing ratio to classify vehicles as well. The new method eliminates the need for velocity detection for classification, which in turn is expected improve vehicle classification capability when utilizing single-element piezoelectric sensor.

5.1.1 Changes in signal due changes in vehicle velocity and axle spacing

This subsection offers test results that highlight the effects of changing velocity and vehicle type over output signal.

Figure 5.1 shows examples of aligned test runs for a passenger car. In graph (a) four aligned runs are plotted to represent results of a test in which the driver was asked to maintain a speed of 20mph. Graph (b) shows four aligned runs plotted, representing when the driver was asked to maintain a speed of 30mph. A comparison is beneficial for highlighting uniform output at the same velocity for the same vehicle. Of note, however, is that the uniformity will change once either velocity or axle spacing changes. Both parameters affect the time between pulses and the shape of detected signal, consequently affecting vehicle classification. As we can see in figure 5.1, as velocity increases both amplitude level and time duration of the signal decreases. Also in (a) we can see that slight changes in vehicle velocity have a clear impact on output signal. Variations in velocity are effects of the driver failing to maintain a constant velocity of 20mph for all test runs.



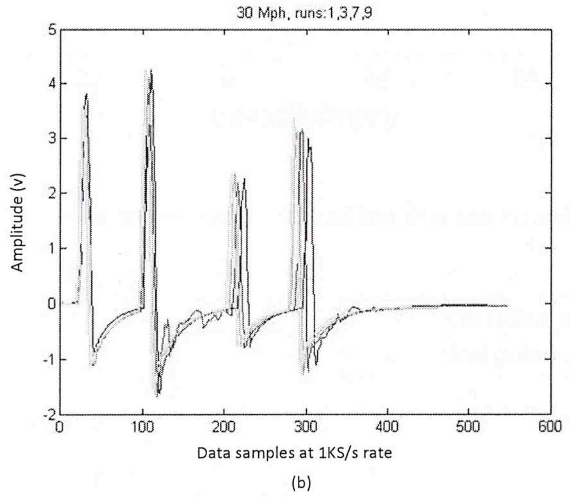


Figure 5-1 Aligned vehicle signals acquired by diagonal single element piezoelectric sensor a) car at 20mph b) car at 30mph

A test set of 11 runs for a passenger car vehicle was used to monitor changes in inter-pulse periods at different velocities. Testing vehicle velocities ranged between 15 and 30 mph. Figure 5.2 demonstrates time durations between the first and second pulse and between the first and third pulse. Duration between the first and second pulse corresponds to vehicle track width, while duration between the first and third pulse corresponds to vehicle axle spacing.

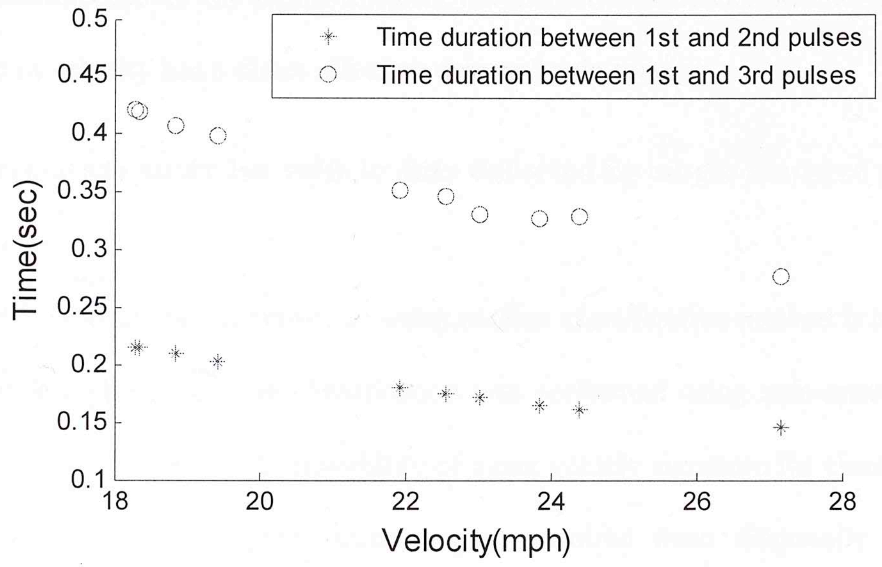


Figure 5-2 Time durations between 1st and 2nd tires and 1st and 3rd tires

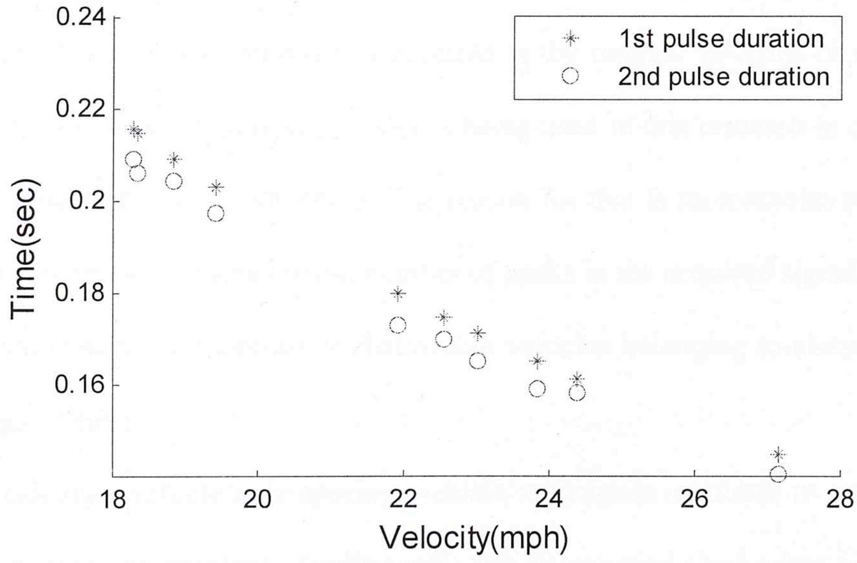


Figure 5-3 Time durations of 1st and 2nd tires pulses.

In figure 5.3, time duration of the first pulse and second pulse are plotted against vehicle velocity for the test vehicle. Vehicle velocity ranges between 15 and 30mph, just as in the previous figure.

Results in figures 5.2 and 5.3 aid in choosing an appropriate sampling rate for the DAQ unit on the embedded system. As vehicle velocity increases, the time interval between pulses and of each distinct pulse decreases. This requires an increase in sampling rate, as well, to accommodate for the higher frequency signal. Also, variation in inter-pulse time due to a change in velocity has a direct effect on detected axle spacing.

5.1.2 Correlation study for vehicle data detected by single element piezoelectric sensor

In this section, the possibility of using another classification method is explored. Until this point in the project vehicle classification was performed using axle-spacing algorithm. This specific study looked at the possibility of using vehicle signature for classification. This signature is obtained from piezoelectric signal acquired from diagonally placed single element sensor when a vehicle passes over it.

Several parameters are required to perform vehicle classification using axle spacing methodology. The primary information required is the number of axles or number of tires. Number of tires instead of number of axles is being used in this research in order to improve motorcycle detection and classification. The reason for that is motorcycles (Class1) are now differentiated from other classes using number of peaks in the acquired signal rather than axle spacing. Axle spacing is necessary to distinguish vehicles belonging to classes that share the same number of tires.

To calculate vehicle axle spacing, vehicle velocity is required; to calculate velocity, distance and time are required. Traditionally this is accomplished using two piezoelectric sensors or two inductive loops separated by a fixed distance. This can also be achieved using a diagonal sensor configuration wherein vehicle width serves as the reference distance.

Classification can also occur with vehicle signature recognition, often with a neural network after training its nodes to classify vehicles. Research in this area has previously focused on inductive loop output signal. The following subsection depicts results using limited data from a preliminary correlation study.

The following graph depicts examples of the correlation factors of data for two car runs at the same speed.

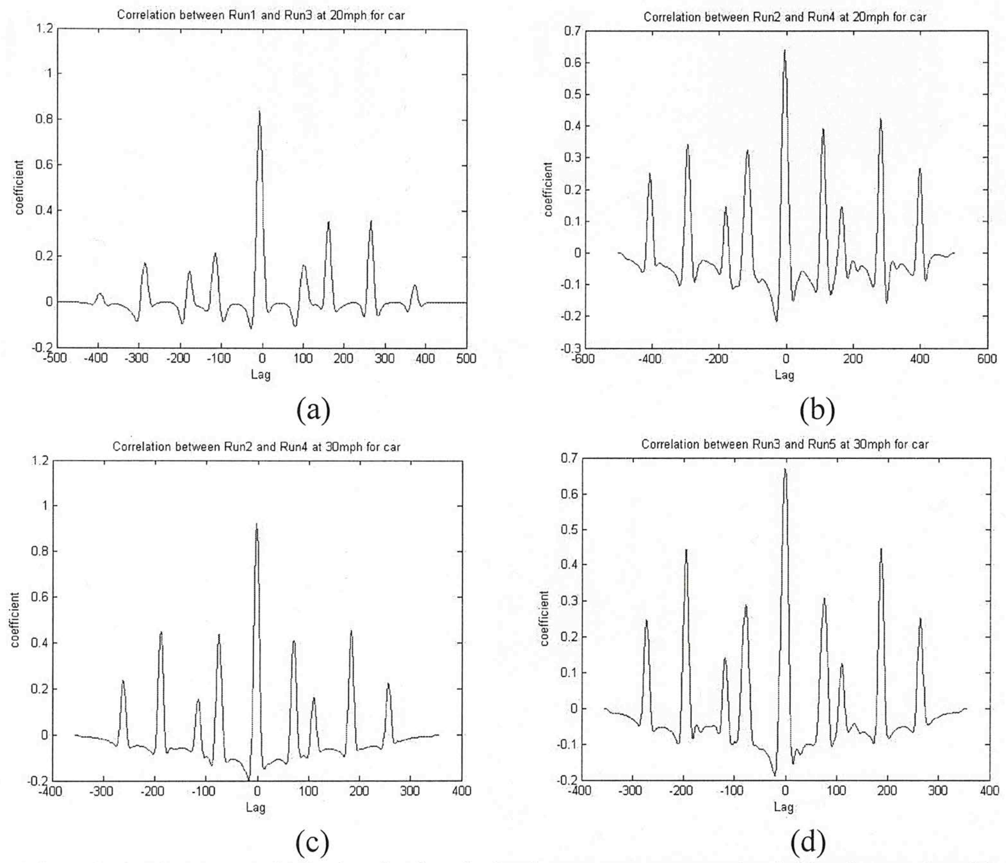
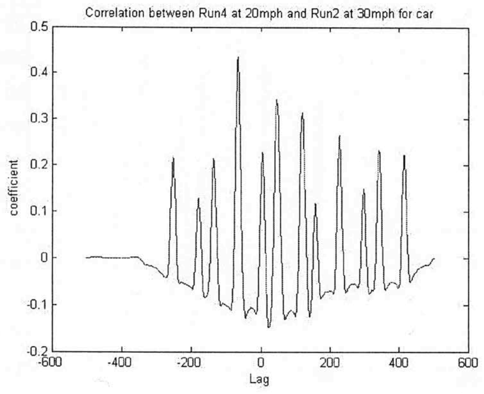


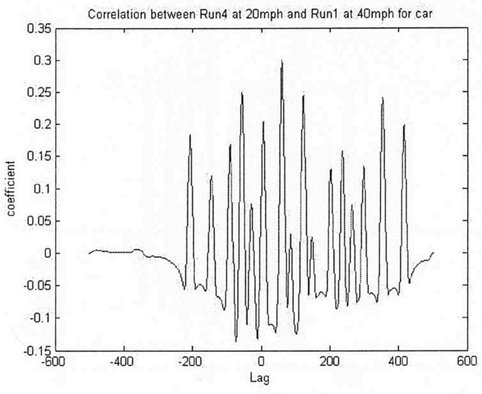
Figure 5-4 Correlation between vehicle signal using single element at same speeds a.) 20mph runs 1 and 3; b.) speed 20mph runs 2 and 4; c.) speed 30mph runs 2 and 4; and d.) speed 30mph runs 3 and 5.

Figure 5.4 shows that the maximum correlation coefficient is relatively high, ranging between 0.6412 and 0.921. A pattern can be seen in the correlation graphs where nine peaks, representing higher correlation, are repeated in all graphs. Peaks in correlation graphs are the result of four pulses in each vehicle output signal subsequent to falling sequentially above one another. The four pulses in vehicle output signal are representative of the test vehicle's four tires.

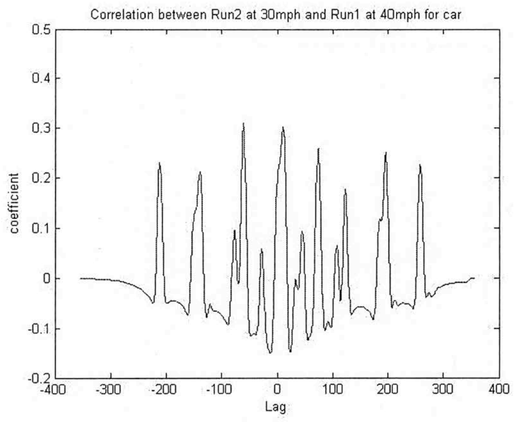
Figure 5.5 shows the correlation between signals of the same vehicle (car) at different speeds.



(a)



(b)

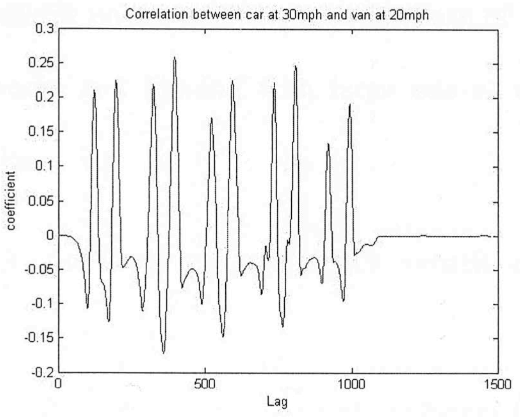


(c)

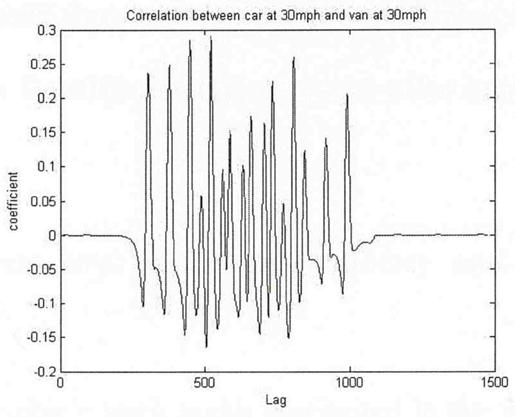
Figure 5-5 Correlation between vehicle signal using single element at different speeds a.) 20mph and 30mph; b) 20mph and 40mph; and c) 30mph and 40mph.

The maximum correlation factors for graphs in fig. 5.5 range between 0.3 and 0.4346—nearly half the correlation factors belonging to signals with the same vehicle speed. Of note is that the pattern in the correlation graphs was lost, as well.

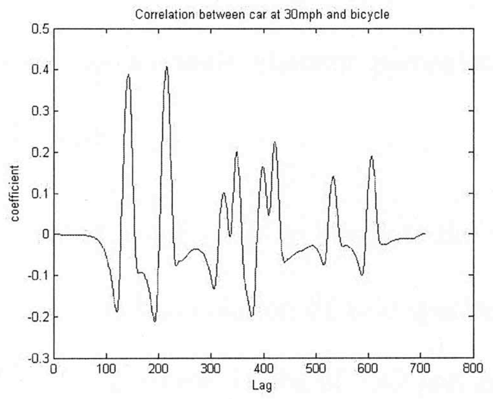
Correlation coefficients for different types of vehicles were investigated in the following. Figure 5.6 demonstrates the correlation between a car at 30mph, a van at 20 and 30mph, and a bicycle. Car data was sampled using NI DAQ at 1kS/s; van and bicycle data was sampled using REECE DAQ at 10kS/s. To perform correlation of data from the van and bicycle vs. the car, van and bicycle data were down-sampled by factor of 10.



(a)



(b)



(c)

Figure 5-6 Correlation between different vehicles signals using single element at different speed a.) car at 30mph and van at 20mph; b.) car at 30mph and van at 30mph; and c.) car at 30mph and bicycle.

Maximum correlation coefficients are 0.2584 for car at 30mph and van at 20mph; 0.2902 for car at 30mph and van at 30mph; and 0.4081 for car at 30mph and bicycle. Correlation coefficients actually dropped from car signal correlations at different speeds, although the drop was minimal. Maximum correlation coefficient of car/bicycle signals is higher than some car/car signals at different speeds, even though car/bicycle signals have entirely different signatures.

No clear, consistent pattern is seen in correlation graphs once either speed or vehicle class was changed. In summary, data from this study implies that because there are two varying parameters—axle spacing and vehicle speed—classification cannot be directly performed using only the correlation between signal from passing vehicles and stored signatures unless an enormous database of vehicle signatures is available. Utilizing neural networks and training with large sets of data for different classes could offer improved results.

5.1.3 Average vehicle track width effects over calculated velocity and axle spacing

As previously discussed in chapter IV, vehicle track width is required in the feature extraction phase of the vehicle classification algorithm. An average track width must be chosen for all vehicle classes, as a single element piezoelectric sensor is incapable of detecting actual vehicle track width.

A road test was performed using a van to simulate the way in which average track width would affect the algorithm and calculation of axle spacing and velocity. Actual track width for the van is 5.78 feet. Two extreme widths of 7.42 feet and 4.14 feet were also tested to compare the effect with acquired vehicle parameters. The driver was asked to travel at speeds of 20-, 25-. and 30-mph. The average of the detected and calculated velocity and axle spacing for four runs at 20 mph, five runs at 25 mph, and a single run at 30mph is depicted in Table 5.1. Of note is that the van's actual wheel base (axle spacing) is 11.5ft, 3.5m.

Table 5-1 Average track width effect over velocity and axle spacing

Requested Speed (mph)	**Track width= 5.78'**		**Track width =7.42'**		**Track width=4.14'**	
	Speed (mph)	Axle spacing (ft)	Speed (mph)	Axle spacing (ft)	Speed (mph)	Axle spacing (ft)
20	18.71	11.27	24.03	14.49	13.40	8.07
25	23.14	11.42	29.70	14.67	16.57	8.17
30	27.16	11.05	34.86	14.17	19.45	7.90

When extreme average track widths are chosen there is a significant change in axle spacing. The shift is 22% for upper limit track width and 39% for lower track width. This phenomenon could lead to classification shift to adjacent classes for vehicles that have a similar number of tires. The deficiency is tolerable, however, since for our purposes, motorcycle classification is not affected and average track width will be used. However, future work should include a suggested solution for detecting actual vehicle width when the piezoelectric sensor is segmented to multi-elements and diagonally placed on the road.

5.1.4 Vehicle classification method using axle spacing to track width ratio

As suggested earlier in this thesis, vehicle velocity is required for classification. Vehicle track width will be needed to calculate velocity when using single-element piezoelectric sensor for vehicle classification. This could be problematic as track width is needed for velocity calculation which unfeasible using single-element piezoelectric sensor. This subsection presents novel approach for vehicle classification using a ratio of vehicle axle spacing-to-track width (L/w), which eliminates the need for velocity and, thus, track width to perform vehicle classification. This method can be applied to classify vehicles with a similar number of tires. As previously discussed, time duration between the first pulse and second pulse (T12) is proportional to vehicle track width w, and time duration between the first and third pulse (T13) is proportional to vehicle axle spacing L. Vehicle velocity V relates T12 to w and T13 to L. By taking the ratio of L to w, we find the following:

$$\frac{L}{w} = \frac{T12.V}{T13.V} = \frac{T12}{T13} \qquad \text{eq. (5.1)}$$

As we can see from equation 5.1, L/w ratio is obtained by detecting the time duration between the first and second pulse and the first and third pulse. Capability of employing this ratio to set thresholds that enables us to accomplish vehicle classification will be assessed using available test data. Potentially, the need for vehicle velocity and track width to achieve vehicle classification could be eliminated. A data set of 30 test runs—10 for a car; 10 for a four-tire, two-axle truck; and 10 for a van—was used to validate this approach. The driver was asked to drive five runs at 20mph and five runs at 25mph. Four runs in total—two runs for the van at 20mph and two for the truck at 25mph—failed. For all others, length-to-width ratios (T13/T12) were plotted with respect to calculated velocity. See figure 5.7. Please note the actual width of each vehicle was used to calculate velocity.

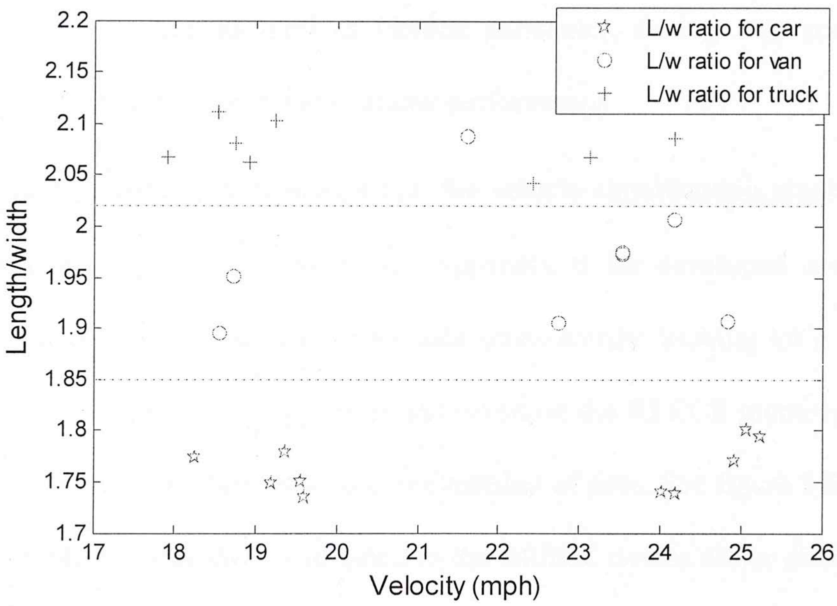


Figure 5-7 Length to width ratio with respect to vehicle velocity.

Figure 5.7 shows that ratios for each vehicle type can be separated by clear thresholds. Only a minor overlap at one point exists between van and truck L/w ratios. Taking this initial test into consideration, this method for vehicle classification looks promising when using single-element piezoelectric sensor without the need to explicitly acquire vehicle velocity or width. However, data from additional testing of vehicles sharing the same number of tires must be studied to fully assess the performance of this approach. L/w mean for acquired testing data are 1.7626, 1.9622, and 2.0774 for car, van, and truck respectively. Standard deviations for the same data are 0.0238, 0.0643, and 0.0222 for car, van, and truck respectively. Please note that actual L/w ratios for test vehicles are 1.8644, 2, and 2.1111 for car, van, and truck respectively.

Over 51 test runs for the passenger vehicle, van, and a four-tire truck were undertaken to verify voltage outputs and to design feature extraction thresholds.

5.2 Developed feature extraction and classification algorithm verification test

Developed feature extraction and classification algorithm verification, including on-road, classification testing, as well as yielded parameter, testing was accomplished post algorithm development and coding to evaluate performance.

Previously, Chapter IV indicated that the vehicle classification algorithm was coded and developed on the REECE device. See Appendix B for developed code detail. When executed, the program processes the sensor data continuously, looking for a passing vehicle. per-vehicle records are reported on screen and saved on the REECE memory. These include passing time, date, vehicle class, velocity, and number of tires. See figure 5.8 for an example of reported output. A connection is initiated to the REECE device either directly to its public IP address or through a VPN connection from a server. During the session, the program can be initiated and stopped, and vehicles can be detected as they pass.

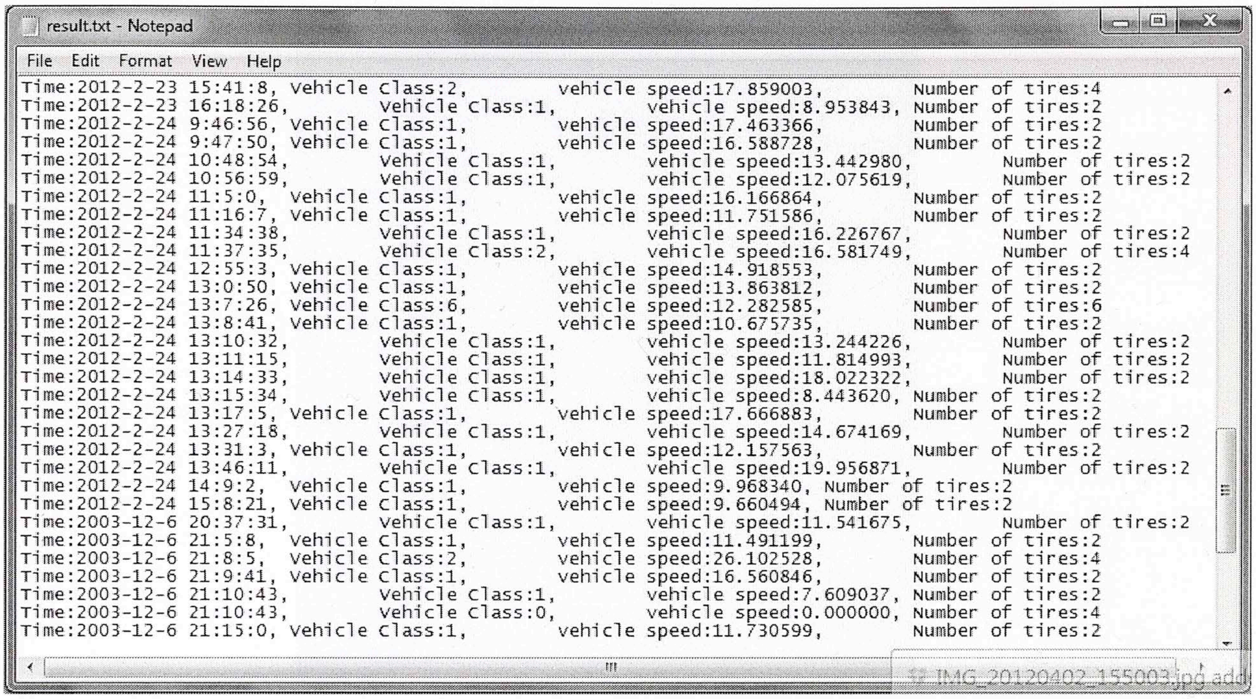
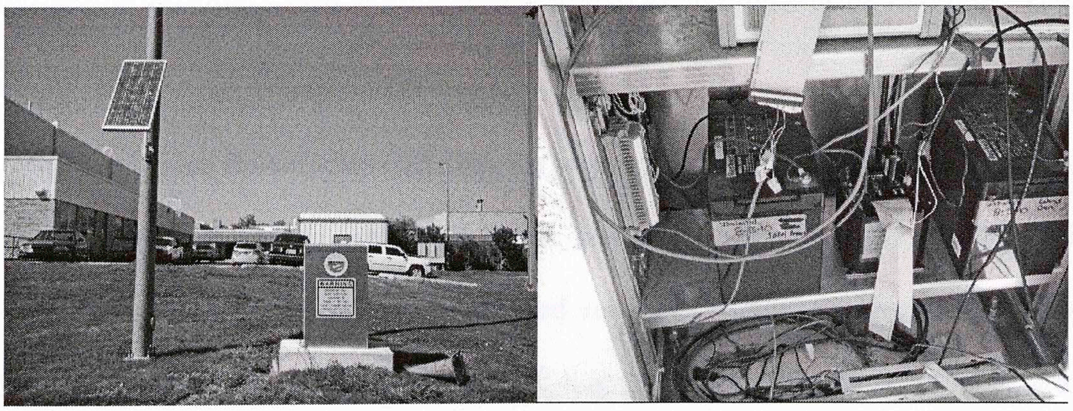

```
Time:2012-2-23 15:41:8, Vehicle Class:2,        vehicle speed:17.859003,        Number of tires:4
Time:2012-2-23 16:18:26,        Vehicle Class:1,        vehicle speed:8.953843, Number of tires:2
Time:2012-2-24 9:46:56, Vehicle Class:1,        vehicle speed:17.463366,        Number of tires:2
Time:2012-2-24 9:47:50, Vehicle Class:1,        vehicle speed:16.588728,        Number of tires:2
Time:2012-2-24 10:48:54,        Vehicle Class:1,        vehicle speed:13.442980,        Number of tires:2
Time:2012-2-24 10:56:59,        Vehicle Class:1,        vehicle speed:12.075619,        Number of tires:2
Time:2012-2-24 11:5:0,  Vehicle Class:1,        vehicle speed:16.166864,        Number of tires:2
Time:2012-2-24 11:16:7, Vehicle Class:1,        vehicle speed:11.751586,        Number of tires:2
Time:2012-2-24 11:34:38,        Vehicle Class:1,        vehicle speed:16.226767,        Number of tires:2
Time:2012-2-24 11:37:35,        Vehicle Class:2,        vehicle speed:16.581749,        Number of tires:4
Time:2012-2-24 12:55:3, Vehicle Class:1,        vehicle speed:14.918553,        Number of tires:2
Time:2012-2-24 13:0:50, Vehicle Class:1,        vehicle speed:13.863812,        Number of tires:2
Time:2012-2-24 13:7:26, Vehicle Class:6,        vehicle speed:12.282585,        Number of tires:6
Time:2012-2-24 13:8:41, Vehicle Class:1,        vehicle speed:10.675735,        Number of tires:2
Time:2012-2-24 13:10:32,        Vehicle Class:1,        vehicle speed:13.244226,        Number of tires:2
Time:2012-2-24 13:11:15,        Vehicle Class:1,        vehicle speed:11.814993,        Number of tires:2
Time:2012-2-24 13:14:33,        Vehicle Class:1,        vehicle speed:18.022322,        Number of tires:2
Time:2012-2-24 13:15:34,        Vehicle Class:1,        vehicle speed:8.443620, Number of tires:2
Time:2012-2-24 13:17:5, Vehicle Class:1,        vehicle speed:17.666883,        Number of tires:2
Time:2012-2-24 13:27:18,        Vehicle Class:1,        vehicle speed:14.674169,        Number of tires:2
Time:2012-2-24 13:31:3, Vehicle Class:1,        vehicle speed:12.157563,        Number of tires:2
Time:2012-2-24 13:46:11,        Vehicle Class:1,        vehicle speed:19.956871,        Number of tires:2
Time:2012-2-24 14:9:2,  Vehicle Class:1,        vehicle speed:9.968340, Number of tires:2
Time:2012-2-24 15:8:21, Vehicle Class:1,        vehicle speed:9.660494, Number of tires:2
Time:2003-12-6 20:37:31,        Vehicle Class:1,        vehicle speed:11.541675,        Number of tires:2
Time:2003-12-6 21:5:8,  Vehicle Class:1,        vehicle speed:11.491199,        Number of tires:2
Time:2003-12-6 21:8:5,  Vehicle Class:2,        vehicle speed:26.102528,        Number of tires:4
Time:2003-12-6 21:9:41, Vehicle Class:1,        vehicle speed:16.560846,        Number of tires:2
Time:2003-12-6 21:10:43,        Vehicle Class:1,        vehicle speed:7.609037, Number of tires:2
Time:2003-12-6 21:10:43,        Vehicle Class:0,        vehicle speed:0.000000, Number of tires:4
Time:2003-12-6 21:15:0, Vehicle Class:1,        vehicle speed:11.730599,        Number of tires:2
```

Figure 5-8 Vehicle per vehicle reported output.

Testing equipment was deployed on campus, and three testing vehicles were used: a passenger vehicle, a van, and a four-tire, two-axle, truck. A 12' piezoelectric sensor was deployed diagonally at an angle of 45^{o} across a traffic lane. The sensor was connected to a REECE device located in a cabinet located 75 feet from the road on which the sensor is installed. See figure 5.9.



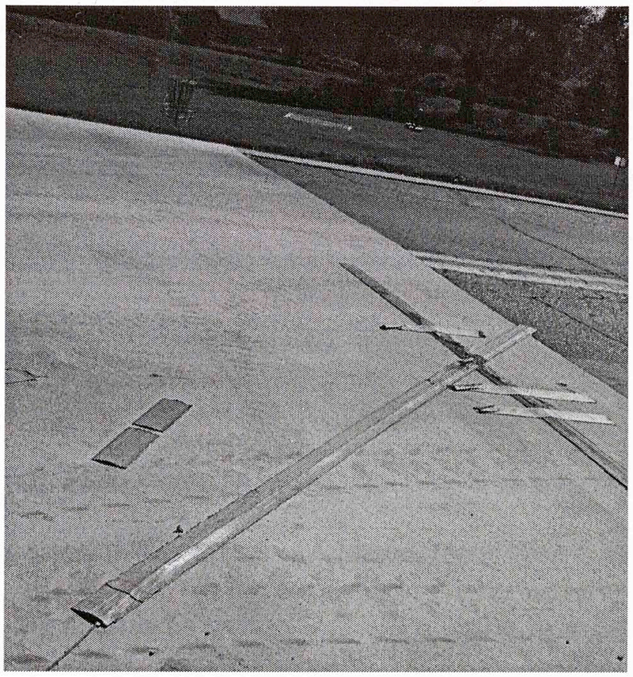


Figure 5-9 On-campus test site for single element vehicle classification system

Axle spacing values were 9.16 feet, 11.66 feet, and 11.08 feet for the car, van, and truck, respectively while track widths were 4.91 feet, 5.83 feet, and 5.25 feet.

A driver was asked to test drive each vehicle, being sure to impact a sensor with all four tires for eight runs—four at 20mph and four at 30mph. An emulation of a motorcycle using two tires was also tested. Again, the driver was asked to test drive the emulated motorcycle for four runs—two at 20 mph and to two at 30 mph—using two tires. Test runs total 36 in number.

All test drives for vehicles with four tires were classified as class2. This was expected, as the axle spacing threshold separating class2 and class3 vehicles is 12.99 feet, which is higher than the axle spacing values for all used vehicles. To ensure algorithm stability, the threshold was changed to 11.15 feet, and an off-line test was accomplished using data from a van. Results are presented in Table I, yielding 90% classification rate as class3 vehicles while the remaining 10% was classified as class2 vehicles.

Table 5.2 shows velocity average and standard deviation for test runs. Results show consistency among velocity calculated. It is important to note that velocity is prone to human error, as a diver might not be able to travel exactly at the pre-specified speed.

Accuracy in velocity calculated using a vehicle running with two tires on the sensor is much lower than the case of four tires, primarily because the algorithm considers vehicle axle spacing instead of track width as parameter to calculate velocity since now only two tires are impacting the sensor. This issue, however, gives us a way to validate algorithm consistency at different vehicles track widths. Velocity is calculated using time duration between first and second pulses which corresponds to track width when using four tires and to axle spacing when using two tire. Taking ratio between velocity using second configuration to velocity using first configuration corresponds to ratio of vehicle axle spacing to track width. Comparing those two ratios together helps us validate algorithm consistency when different vehicles having different track width trigger the sensor. Ratios of calculated velocities for four tires compared to calculated velocities for two tires for car, van, and truck were evaluated. Averages of those ratios were taken for each type of vehicle to combine 20mph results together and 30mph results together. Final ratios are 1.7792, 1.8719 and 2.0326 for car, van and truck respectively. Ratios for each vehicle axle spacing L to track width w were then taken for comparison with velocity ratios. These L/w ratios were 1.8644, 2, and 2.1111, respectively. Comparing L/w ratio to velocity ratios proves them compatible, recognizing that an average track width of 5.78' was used.

Table 5-2 Calculated velocity averages and standard deviations

Requested Speed in mph	Car velocity average (mph)/ velocity standard deviation		Van velocity average (mph)/ velocity standard deviation		Truck velocity average (mph)/ velocity standard deviation	
	Four tires	Two tires	Four tires	Two tires	Four tires	Two tires
20	22.96/ 0.7724	12.81/ 0.1474	17.72/ 0.5915	9.97/ 0.1429	21.28/ 1.0959	10.25/ 0.0943
30	34.12/ 0.3815	19.32/ 0.1339	27.55/ 1.2422	14.01/ 0.5279	29.20/ 0.4629	14.68/ 0.4254

5.3 Multi-element piezoelectric sensor preliminary testing

This section includes testing of multi-element sensor design, as well as preliminary testing of the fabricated multi-element sensor. Connection type was initially assessed. This test is important, since many channels are connected to the DAQ and these might be prone to coupling between different channels and outside noise. Connection cable is tested next for mechanical pressure. Part of the cable, which will be impacted by passing vehicle tires, is laid on the road surface. Because these might create pseudo vehicle pulses, they were lab tested to account for noise coupling . Also the fabricated multi-element sensor was tested on road.

5.3.1 Signal coupling test

In the signal coupling test both single ended (SE) connection and differential connection modes were tested and reported. In SE mode a single input (core) was connected to input channel while the braid is grounded. In differential mode, core is connected to the positive differential input, while braid is connected to negative differential input. Both inputs were then referenced to common ground through resistors.

In the multi-element piezoelectric sensor, multiple channels shared the same multi-conductor cable in which electric signals travel to the DAQ system. Also, in many applications, noise from electronics and the surrounding environment may couple to the designed/studied system, negatively affecting them. Several coupling types of traveling signal occur between conductors at close proximity, namely, radiative, capacitive, and conductive [39].

Radiative coupling is somewhat irrelevant to our study, since the system is designed for highway deployment in areas distant from sources with possibly significant radio wave effect.

For capacitive coupling, a result could be signal corss talk, especially at higher frequencies. When two conductors share the same cable at close proximity, both act as a capacitance. With higher signal amplitude and higher frequency more power will couple to the adjacent wire. Usually by increasing separation between conductors we'll have higher capacitance, thus less cross talk. However this is not an option when using the same cable.

Inductive coupling is caused by the magnetic fields resulting from current passing through the conductor. These fields are vary due to the time varying nature of the passing current. The magnetic field will induce a voltage in nearby conductors. These two conductors can be seen as a transformer with a mutual conductance M. Thus the induced voltage in the interfere (Vn) is given in eq.5.2:

$$V_n = 2\pi f M I_n \qquad \text{eq. (5.2)}$$

where I_n is the induced current, and f is its frequency. For more details refer to [39].

Eq. 5.2 clearly demonstrates that increasing frequency also increases inductive coupling. Inductive coupling can be mitigated by increasing space between conductors. However, as previously stated, this is not feasible when sharing the same cable. Shielding can be used, as well, to channel unwanted signal to the ground.

For a coupling problem, it is advised to use a twisted pair cabling with differential mode. In contrast to SE mode, differential mode is inconvenient to use because it employs twice the amount of DAQ resources since each input requires two channels instead of one. This can create a problem, since every DAQ has a limited number of input channels and the proposed method requires a large number of piezo-elements.

Lab testing was performed for preliminary assessment of coupling and its effect on signal integrity when using piezoelectric sensors. To investigate referenced single ended (RSE) and non-referenced single ended (NRSE), two inputs were attached to channels 1 and 2 in the DAQ. In RSE mode, channel 1 was attached to a piezo-element. See figure 5.10. Channel 2 was attached to a 500Hz sine wave generated by a waveform generator. See figure 5.11. Crosstalk from channel 2 to channel 1 is evident; however, the opposite was not true due to high noise at the channel 2 input.

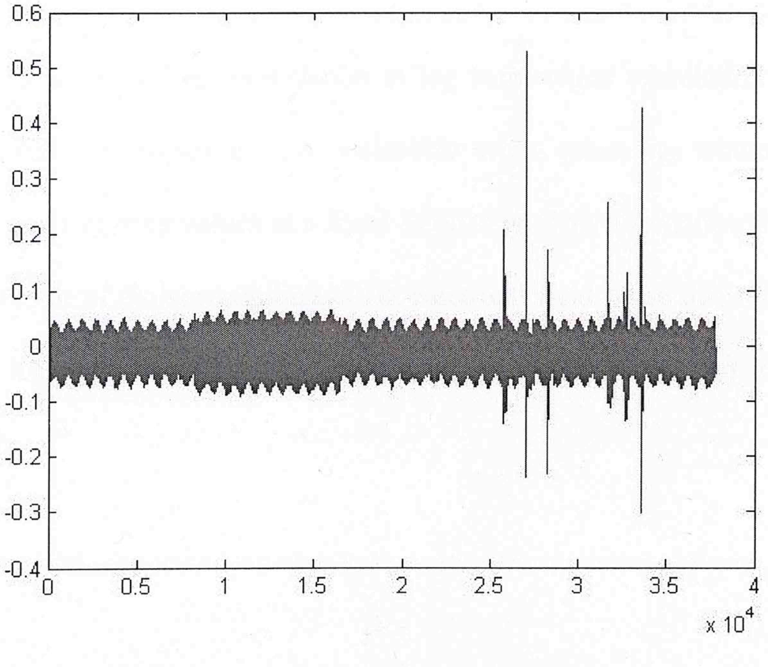


Figure 5-10 RSE mode piezo-element input channel

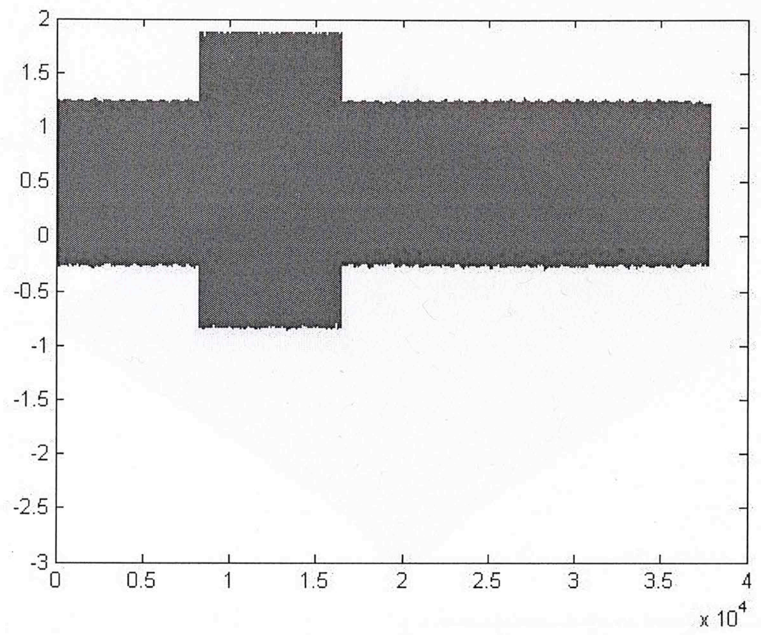


Figure 5-11 RSE mode 500Hz sine wave input channel

The following graph shows the correlation factor between two channels at different lags. Since noise in channel2 is too high, and channel1 to channel2 crosstalk isn't evident, the correlation function could offer a better idea of coupling level.

Figure 5.12 shows a high correlation at lag zero where correlation factor ρ ~= 0.7. Fluctuation in correlation factor is also noticeable when zooming, wherein the plot alter between positive and negative values at a fixed frequency. This alternation frequency is equal to 60Hz and the result of the surrounding 60Hz electrical field noise due to lab testing. Thus when moving in the correlation function this noise get out of phase and then in phase at a fixed rate of 60Hz.

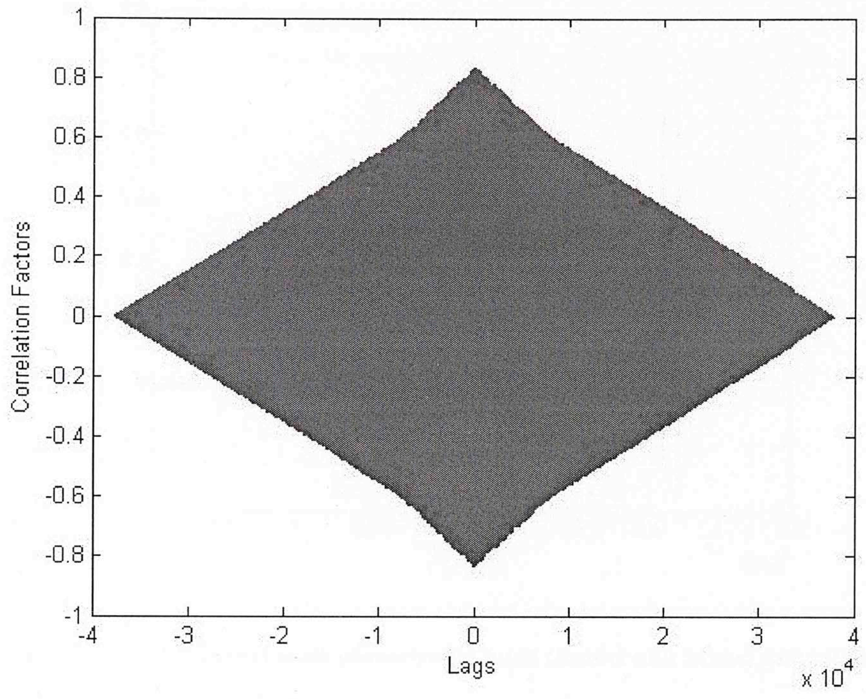


Figure 5-12 Correlation between inputs of channels 1&2 in RSE mode

NRSE mode was tested, and results were poor as a result of shifted output and high noise. As such, it will not be considered for future experiments.

The following test was conducted with twisted pair cable and differential DAQ mode. As previously mentioned, the purpose for using twisted pair cabling is mitigating coupling effect. Channel1 input resulted in a pulse generated by a piezo-element, as shown in figure 5.13. Channel2 input was a sine wave at 500Hz and 600mv, as shown in figure 5.14. Figure 5.14 clearly demonstrates an extremely high noise component when the input is off. Of note is that the piezoelectric signal was generated in the lab merely by tapping on the piezo-element.

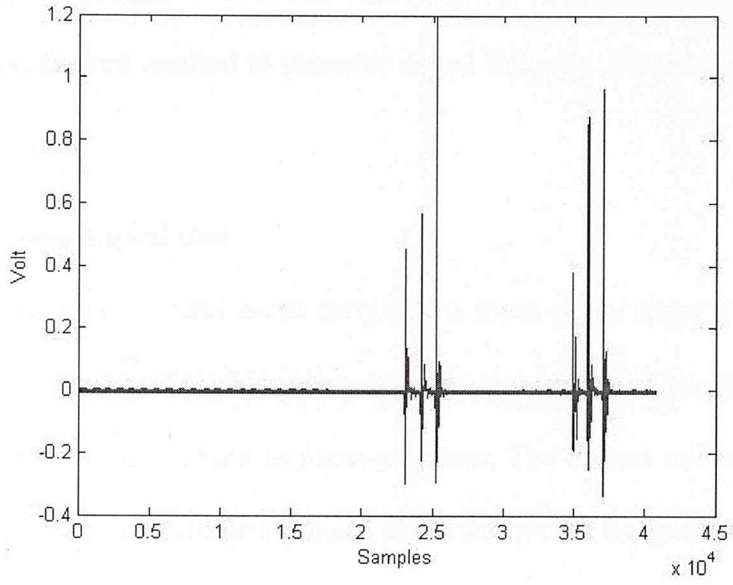


Figure 5-13 Differential mode piezo-element input channel with twisted pair cabling

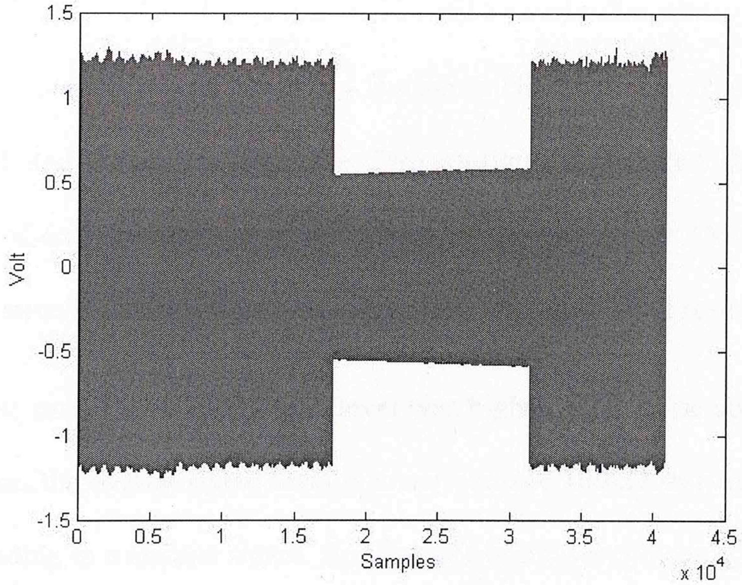


Figure 5-14 Differential mode 500Hz sine wave input channel with twisted pair cabling

Correlation factor between both inputs was calculated, and a correlation factor at lag zero is ρ ~= 0.7 was assigned.

In terms of coupling, it was concluded that using differential mode with twisted pair cabling is the most desired method to preserve signal integrity. Please note that no shielding was used.

5.3.2 Cable pseudo signal test

As previous noted, several wires carrying an input signal share a cable. The cable is located on the pavement next to the piezo-elements themselves. This cable is impacted by passing vehicles in the same fashion as piezo-elements. The impact will alter the capacitance between wires and might also create a pseudo pulse that would be detected at the input of the DAQ. Testing was needed to assess different types of connection and observe resulting signals from the cable impact—whether or not it might affect the final method selection. In this test, differential mode with twisted pair was used because of its attractive coupling resistive properties.

Lab tests were conducted wherein a student in charge of the experiment impact both the piezo-element and the conducting cable. Two configurations were tested: 1.) Positive and negative inputs of each channel were referenced to the ground by 1MΩ resistors; and 2.) Positive and negative inputs are referenced to ground through 100kΩ resistors.

In the first case, the overall signal level was higher, thus, noise appears more clearly. In the second case, the overall signal level is lower because 100kΩ resistors pass more power to the ground leading to a cleaner signal. An example of the second configuration is shown in figure 5.15 wherein six piezo-element impacts and 13 cable impact s are depicted.

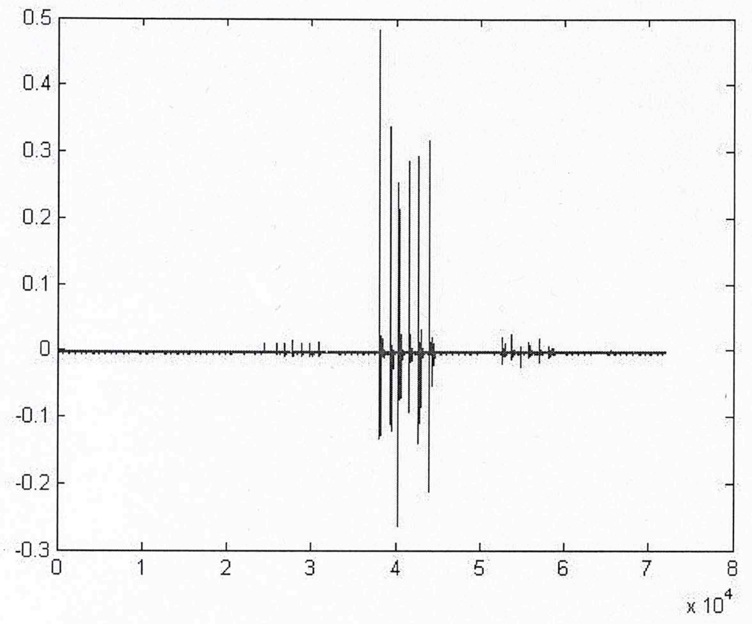


Figure 5-15 Cable hit compared to piezo-element hit in-lab test using 100kΩ resistor for ground referencing

The average of the six piezo-element peaks amplitude impact s is 0.327v, whereas the average of 13 peaks generated by a cable impact is 0.0121v. Thus, in this scenario, the cable hit level is approximately 28db lower than that of the piezo-element impact.

5.3.3 Multi-element sensor road test

Three vehicles (car, van, and truck) were used to test a multi-element sensor at University of Oklahoma-Tulsa campus. The sensor was deployed on the campus' south road at a 45 degree angle to traffic flow.

A driver was asked to perform 10 test runs—five at 20 mph and five at 30 mph—for each of the three vehicle types, making sure that all four tires crossed a sensor. Figure 5.16 depicts the road test setup.

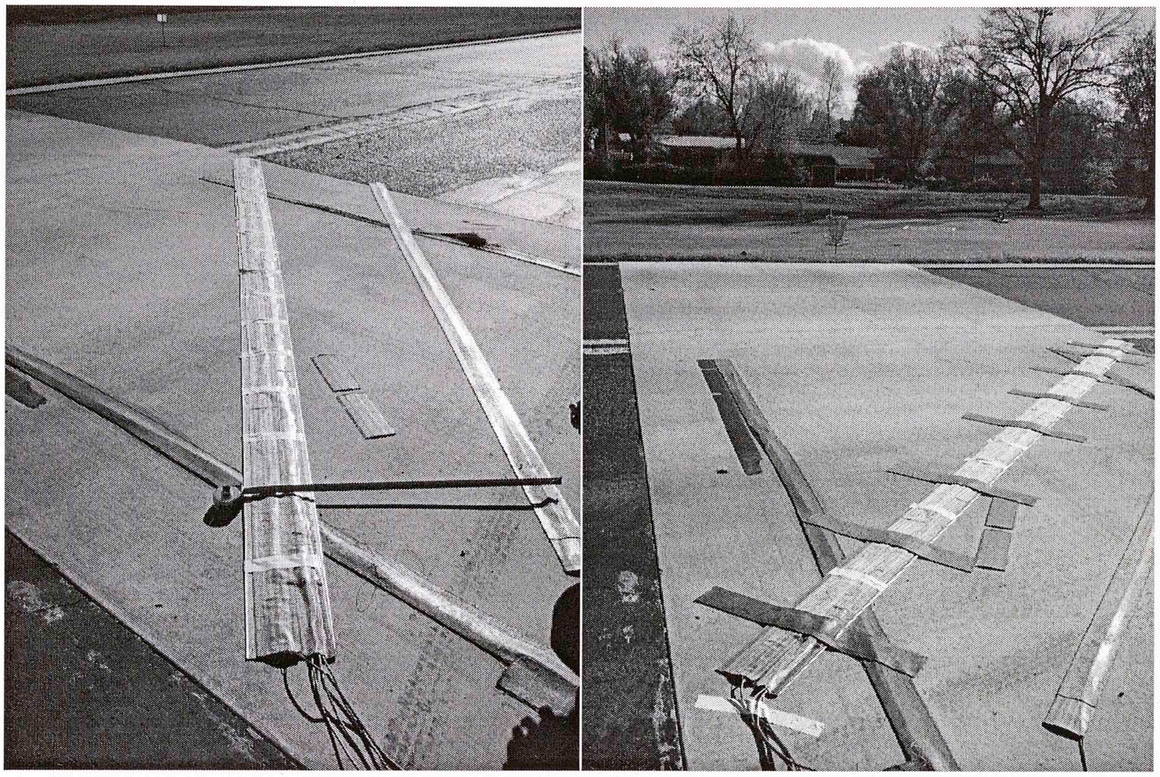


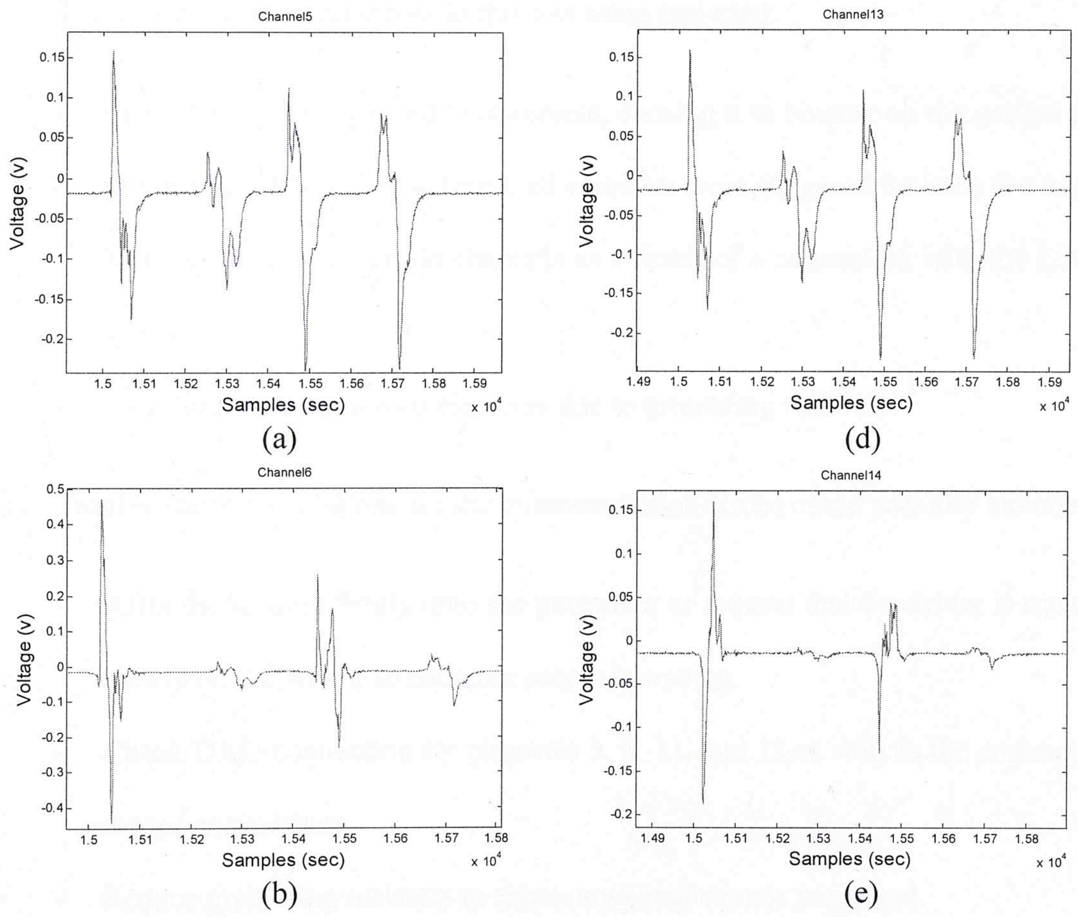
Figure 5-16 On road test site for multi-element piezoelectric sensor.

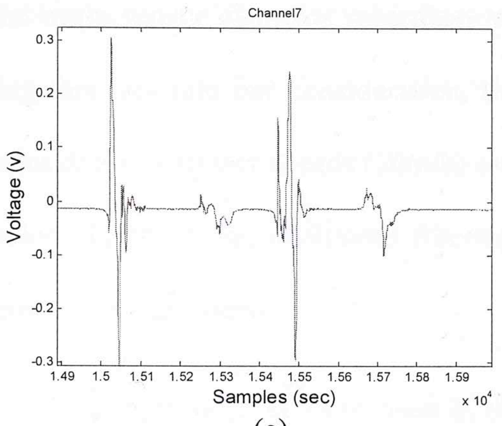
Yielded results demonstrated a repeating pattern in all tests. Elements 3, 4, 11, and 12 registered voltage amplitude that is an order of magnitude higher than other sensor elements. Also, sensor elements 5 and 13 showed a higher amplitude value than other sensor elements, implying voltage build up on input channels. During this test the driver attempted to impact different elements during different test runs.

Other sensor elements produced the same voltage level with varying values between different elements. Sensor elements registered four pulses in most test runs. Although pulses were characterized with different values in different runs, signal shapes from all elements were similar.

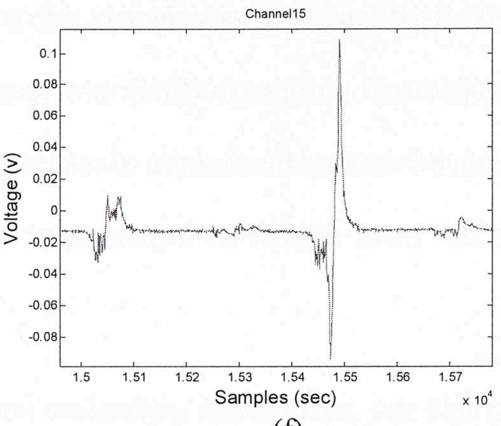
These issues made it difficult to distinguish which element was triggered by which tire. Elements unaffected by high level voltage were studied, as voltage values are similar. For some test results some elements measured higher amplitude than adjacent ones; these were separated by width comparable to the vehicle width. See figure 5.17 for example of van signal output at 20mph for an element characterized with higher output and then the adjacent element. Of note is that this was not typical for all test runs. Signals from elements with higher amplitude also have similar pulse shapes.

Figure 5.17 demonstrates that channels 6 and 14 have higher output than the outputs of adjacent channels, when taking into consideration that channel 4 is affected by voltage buildup. However, pulses 1 and 3 have higher amplitude than 2 and 4 in all runs, most likely due to sensor bouncing. Sensor elements 6 and 14 are 8-feet apart, which, when multiplied by cos(45^{o}), is 5.6569 feet. This is comparable to the van track width of 5.8333 feet.





(c)



(f)

Figure 5-17 Multi-element sensor output for van at 20mph a) channel 5 b) channel 6 c) channel 7 d) channel 13 e) channel 14 f) channel 15

The expected cause for errors in this test setup included:

- Sensor is not firmly fixed to pavement, causing it to bounce on the ground at each tire impact. When this occurred, all elements were triggered for each tire impact.
- Voltage buildup at certain channels as a result of a connection with the DAQ and sensor.
- Coupling between sensor elements due to grounding resistor.

Trouble shooting solutions for the aforementioned issues could possibly include:

- Affix the sensors firmly onto the pavement or request that the driver is travel very slowly on the sensor to decrease sensor bouncing.
- Check DAQ connection for channels 3, 4, 11, and 12,as well as the corresponding sensor connections.
- Reduce grounding resistors to drive unwanted signals to ground.

It was discovered that persistent sensor signal errors during our previous sensor road installation and field-testing dated in 03/23/2012 was caused by the loos installation of the sensor on the surface. Mechanical vibration caused by a passing vehicle produced signals on all the multi sensor elements regardless whether the elements were impact by the vehicle. Taking this fact into our consideration, the sensor was firmly fixated on the road and test vehicles drove at slower speeds (10m/h) over the sensor to minimize the sensor bouncing and vibration. Furthermore, additional filtering was implemented to remove cross talk among different DAQ channels.

Three test vehicles were used in this new evaluation, namely; car, van and a truck. Four runs were performed for the car while three runs were performed for each the truck and the van.

Better results and signal quality were obtained during this installation. Impacted sensors accurately generated two pulses indicating vehicle detection while in previous tests impacted elements generated a total of four pulses. Figure 5.18 shows the signals from impacted sensor elements. This is a very interesting result because it proves that once all channel inputs are calibrated to uniform output multi-element sensor will be ready to be used for vehicle classification.

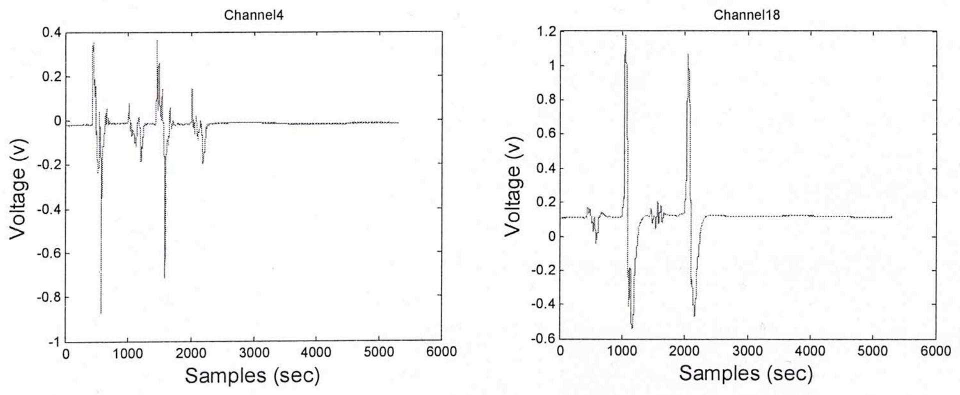


Figure 5-18 Multi-element car signal output. a) Element 4 b) Element 11

The firm installation of the sensor was able to remove the bouncing effect on the sensor. However, signal errors due to coupling and cross talk persisted in this installation. Elements that were not impacted by the overpassing vehicle generated signals as strong as those generated by the impacted elements, resulting in element detection error. problem and energy build up still exists where channel 10-14 have close output that is much higher in amplitude than other channels. Also channel 4 and 5 had close output in most of the test runs. This problem prevents us from detecting exact element that was triggered by the vehicle tire. This can be in part due to connection to the DAQ which is very crowded, see figure 5.19.

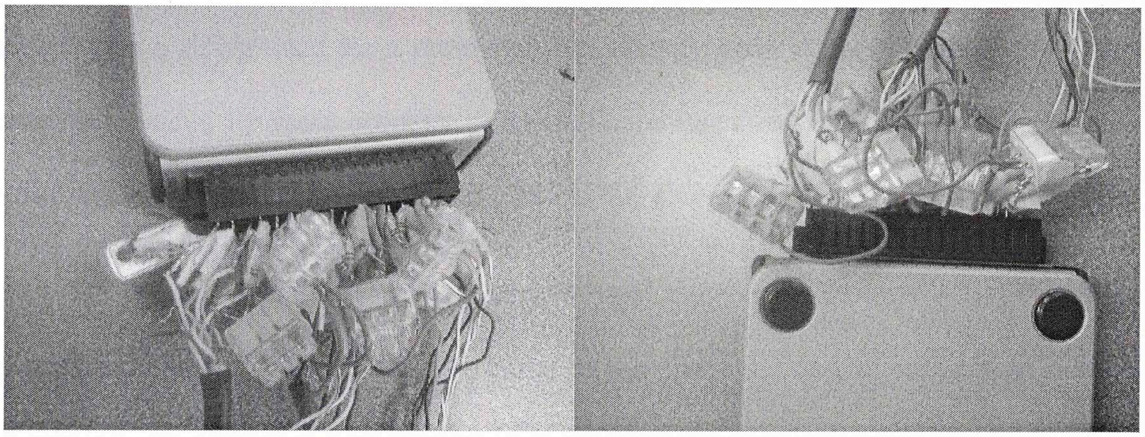


Figure 5-19 Multi-element connection to DAQ

New connection scheme will be done for future testing to eliminate coupling and further testing will be carried out to validate multi-element sensor operation.

Chapter VI

6 Conclusion and future work

The number of motorcycles traveling on highways has increased in recent years. This trend has not been reflected in FHWA statistics for vehicle miles travelled (VMT), indicating a critical deficiency in motorcycle classification using current systems. Such a discrepancy demonstrates a rising need for a more accurate classification system. Though classification system is intended for more accurate motorcycle classification, system should be capable of performing vehicle classification of other 12 FHWA classes.

The proposed vehicle classification system used a novel approach of tilting the piezoelectric sensor to classify vehicles based on tire count and spacing rather than axle count and spacing. An algorithm was developed in which tire count, tire spacing, vehicle velocity, average track width, and pulse duration were used to process feature extraction and vehicle classification. Using tire count makes motorcycles clearly distinguishable from other vehicle classes.

Algorithm was developed on an embedded system using C code. For current development, a single element piezoelectric sensor was used. The sensor was deployed diagonally over the roadway surface. The newly designed, single-element vehicle classification system was tested using multiple vehicles and an emulation of motorcycles with two tires. The system successfully classified motorcycles. Results were presented in this thesis.

A new vehicle classification method using axle spacing to track width ratio was also presented. This method was applied to data from on-road test of various vehicles. The system showed promising results. The method will advance technology so that velocity and track width are no longer be required to classify vehicles given the axle spacing to track width ratio. This method eliminates the need to use the same average track width for all vehicles, thus improving classification accuracy for vehicle classes sharing the same number of tires. A multi-element sensor was fabricated and underwent initial testing. Results are presented in this thesis.

6.1 Future testing

Further on-road testing of the single-element classification system for vehicles representing a variety of classes is needed. Most important, testing will aid in assessing axle spacing to track width ratio usability on all vehicle classes sharing same number of tires. Also, more extensive testing is required to fully develop the multi-element vehicle classification system. Additional focus on the amplitude of signal pulses and their position with regard to each other is warranted to assist in differentiating elements impacted by a particular vehicle tire from other vehicles in close proximity.

6.2 Algorithm development for vehicle classification system using L/w ratio

Using results from variety of vehicle classes tested, the proposed algorithm can be adjusted to use axle spacing to track width ratio for vehicle classification. Testing is required to set thresholds distinguishing vehicle classes with the same number of tires. Changes to single element vehicle classification system are only in software.

6.3 Multi-element classification system development

Multi-element vehicle classification system should also be developed on an embedded system following sufficient testing is performed taking into consideration previously mentioned algorithm differences with single element system. A computing system with superior performance is required since the system is characterized by a large number of DAQ input channels.

References:

[1] "Traffic Safety Facts," NHTSA's National Center for Statistics and Analysis, 2005.

[2] Xuemin Chen et al., "Evaluating Innovative Sensors and Techniques for Measuring Traffic Loads", International Conference on Networking, Sensing and Control, 2008. ICNSC 2008. IEEE , 2008.

[3] B.G. Stoneman and A.C. Moore, "DYNAMIC AXLE AND VEHICLE WEIGHT MEASUREMENTS", Second International Conference on Road Traffic Monitoring, 1989

[4] Saowaluck Keawkamnerd et al., "Vehicle Classification with low computation magnetic sensor", ITS Telecommunications, ITST 2008. 8th International Conference on, 2008.

[5] Sing Yiu Cheung et al., "Traffic measurement and vehicle classification with a single magnetic sensor", 84th Annual Meeting, Transportation Research Board, 2004

[6] Research Division, Nevada Department of Transportation, "Development of a Low-Cost Automatic Vehicle Classification (AVC) System", 2001

[7] Saowaluck Kaewkamnerd et al., "Vehicle Classification Based on Magnetic Sensor Signal," Proceedings of the 2010 IEEE International Conference on information and Automation, 2010

[8] Pursula, M. and Kosonen, I., "Microprocessor and PC-Based Vehicle Classification Equipment Using Induction Loops", Second International Conference on Road Traffic Monitoring, 1989

[9] Janusz Gajda et al., "A Vehicle Classification Based on Inductive Loop Detectors", IEEE Instrumentation and Measurement Technology Conference, 2001

[10] Janusz Gajda et al., "Measurement of Road Traffic Parameters Using an Inductive Single-Loop Detector", 1997

[11] Sroka, R., "Data fusion methods based on fuzzy measures in vehicle classification process," Proceedings of 21st IEEE Instrumentation and measurement Technology Conference, pp.2234-2239, May 2004.

[12] Carlos Sun et al., "Inductive Classifying Artificial Network for Vehicle Type Categorization ," Computer-Aided Civil and Infrastructure Engineering, pp. 161-172, 2003.

[13] Soner Meta and Muhammed G. Cinsdikici, "Vehicle-Classification Algorithm Based on Component Analysis for Single-Loop Inductive Detector," IEEE Transactions on Vehicular Technology, VOL. 59, NO. 6, pp. 2795-2805, 2010.

[14] Surendra Gupte et al., "Detection and Classification of Vehicles",IEEE TRANSACTIONS ON INTELLIGENT TRANSPORTATION SYSTEMS, 2002

[15] Mo Shaoqing et al., "Real-time Vehicle Classification Method for Multi-lanes Roads",4th IEEE Conference on Industrial Electronics and Applications, 2009

[16] Wenbin Zhang et al., "A Novel Vehicle Classification Using Embedded Strain Gauge Sensors," Sensors – Open Access Journal, 8, pp. 6952-6971, 2008.

[17] Ravneet Bajwa et al., "In-Pavement Wireless Sensor Network for Vehicle Classification," Information Processing in Sensor Networks (IPSN), pp. 85-96, 2011.

[18] "Counting Motorcycles final report," American Association of State Highway and Transportation Officials (AASHTO), 2010.

[19] Website: http://support.diamondtraffic.com/knowledgemanager/questions/40/Road+Tube+Known+Problems+%26+Solutions, 10/17/2011

[20] "Traffic Monitoring Handbook", Florida Department of transportation Transportation Statistics Office, 2007

[21] P. Bonsall, "Information Technology Applications in Transport (Topics in Transportation), V.S.P. Intl Science, 1987

[22] "BL Roadtrax® Traffic Sensor Installation Instructions", Measurement Specialties, Inc, 2005.

[23] "Highway Monitoring System Traffic Data For High Volume Routes: Best Practices and Guidelines Final Report", FHWA Office of Highway Policy information, 2004

[24] A Summary of Vehicle Detection and Surveillance Technologies used in Intelligent Transportation Systems, The Vehicle Detector Clearinghouse, 2007

[25] Donald L. Halvorsen, "Piezoelectric Polymer Sensors", National Traffic Data Acquisition Conference, 1996

[26] "Roadtrax® BL Piezoelectric Axle Sensor", Measurements Specialties, 2008

[27] "Piezo Polymer Coax Cable Rev 1", Measurements Specialties, 2009

[28] "Model 108 Temperature Probe Revision: 11/11", Campbell Scientific, Inc. 2011

[29] NI 9215, National Instruments Corporation, 2012

[30] NI 9205, National Instruments Corporation, 2012

[31] "Helios Single Board Computer PC/104 SBC with Vortex Processor and Integrated Data Acquisition", Diamond Systems Corporation, 2012

[32] "Helios Panel I/O Board User Manual Revision A", Diamond Systems Corporation , 2011

[33] "Oracle VM VirtualBox User Manual Version 4.1.8", Oracle Corporation, 2011

[34] Website: http://www.cprogramming.com/tutorial/c-tutorial.html , 04/13/2002

[35] Website: http://gnosis.cx/publish/programming/sockets.html, 04/13/2002

[36] "RFC:793 Transmission Control Protocol, Darpa Internet Program, Protocol Specification", Information Sciences Institute, University of Southern California, 1981

[37] Website: http://users.actcom.co.il/~choo/lupg/tutorials/multi-thread/multi-thread.html, 04/13/2002

[38] Website: https://computing.llnl.gov/tutorials/pthreads/, 02/14/201

[39] "Field Wiring and Noise Considerations for Analog Signals", National Instruments, 2011

Appendix A

This appendix shows the procedure by which a new REECE device is added to the Linux server vpn network. It is assumed that the REECE device is working and have internet connection.

- REECE will keep trying to connect to server.
- On server check for REECE connection and get its public IP address: tail –f /var/log/secure
- Connect to REECE using putty of any other terminal software
- Generate new RSA keys
- Secure copy (scp) public key to the server
- On server add REECE name to the list of users
- Give this user (REECE) permission to create vpn connection

  Echo '<name> ALL=NOPASSWD: /usr/sbin/pppd call <name>' >> /etc/sudoers
- Make new file in "/etc/ppp/peers/" named as the REECE user. This file the vpn connection options and parameters for this user and the specific vpn network address that REECE will have.
- Install public keys on server and make REECE user as their owner:

  cp /PATHofPublicKey/id_rsa.pub /home/<name>/.ssh/authorized_keys2

  chown <name>:<name> -R /home/<name>

Appendix B

Major code used in Matlab:

Feature_extraction.m:

```
function [pulse_numberD,velocityM,vehicle_length, t12,t13, T1D,T3D ] =
Feature_extraction(FileToBeRead)

% assuming a fixed motorcycle length of 5 ft/1.5 meters
vehicle_count = 0;
%while(true)
    signal_state1 = 0; signal_state2 = 0;
    sampling_rate = 10240; sample_period = 1/sampling_rate;
    detection = false; data_pointer = 1;
    pulse_number = 0; zeros_cons = 2/sample_period; % number of samples equvilant to 15 metesr
at 80 mph
    count1 = 0; count2 = 0; count3 = 0; count4 = 0; count5 = 0; count6 = 0; count7 = 0; %
initiating indexes for spikes arrays
    count8 = 0; count9 = 0; count10 = 0; count11 = 0; count12 = 0; count13 = 0; count14 = 0;
    theta = pi/4;
    width = 1.262;
    en_class = false;
    once_cond = true ;
    number_of_starts = 0;
    number_of_ends = 1;
    previous_pulse = -1000000;
    % in C/C++ this will be do{}while
    %while(once_cond)    %commented to prevent continuity (remove condition from if @ 107
too!!
        data = csvread(FileToBeRead);
      % dlmwrite('new_file.txt',eval(var_name),'delimiter','','-append','roffset',5);;
        if data_pointer == 1
            length_data = length(data);
        end
        for counter = data_pointer:length_data          % counter represents value's index
within all of the detected values
            if data_pointer == 1 && counter == data_pointer                         % i
represents value's index within the data file currently being read
                i = 1;
            elseif data_pointer > 1 && counter == data_pointer
                i = counter - data_pointer;
            else i = i + 1;
            end


            if data(i)> -0.25 && data(i) < 0.0025
                data1(i) = 0;
            else data1(i) = data(i);
            end

            if data1(i) > 0
               signal_state2 = signal_state1;
               signal_state1 = 1;
               zeros_cons = 0;
               detection = true;
            elseif data1(i) < 0
               signal_state2 = signal_state1;
               signal_state1 = -1;
               zeros_cons = 0;
               detection = true;
            else
                zeros_cons = zeros_cons+1;
            end

            if signal_state1 == 1 && signal_state2 ~= signal_state1
                if counter > previous_pulse + (0.01/sample_period)
                pulse_number = pulse_number + 1;
                var_name1 = genvarname(['st_spike',num2str(pulse_number)]);          % Can be
implemented usig switch case (1 to 14)
```

```
                eval([var_name1 '= counter;' ])
                previous_pulse = counter;
                number_of_starts = number_of_starts+1;
                end
            end
            if zeros_cons*sample_period > 0.10 && number_of_starts == number_of_ends
%pay attention, might be wrong at high speed
                % pulse_number = pulse_number + 1;
                var_name2 = genvarname(['en_spike',num2str(pulse_number)]);          % Can be
implemented usig switch case (1 to 14)
                eval([var_name2 '= counter-zeros_cons;' ])
                number_of_ends=number_of_ends+1;
            end

            if detection == true %&& sw == true
                if pulse_number ==1                             % extracting pulses
                    count1 = count1 + 1;
                    spike1(count1) = data1(i);
                elseif pulse_number == 2
                    count2 = count2 + 1;
                    spike2(count2) = data1(i);
                elseif pulse_number == 3
                    count3 = count3 + 1;
                    spike3(count3) = data1(i);
                elseif pulse_number == 4
                    count4 = count4 + 1;
                    spike4(count4) = data1(i);
                elseif pulse_number == 5
                    count5 = count5 + 1;
                    spike5(count5) = data1(i);
                elseif pulse_number == 6
                    count6 = count6 + 1;
                    spike6(count6) = data1(i);
                elseif pulse_number == 7
                    count7 = count7 + 1;
                    spike7(count7) = data1(i);
                elseif pulse_number == 8
                    count8 = count8 + 1;
                    spike8(count8) = data1(i);
                elseif pulse_number == 9
                    count9 = count9 + 1;
                    spike9(count9) = data1(i);
                elseif pulse_number == 10
                    count10 = count10 + 1;
                    spike10(count10) = data1(i);
                elseif pulse_number == 11
                    count11 = count11 + 1;
                    spike11(count11) = data1(i);
                elseif pulse_number == 12
                    count12 = count12 + 1;
                    spike12(count12) = data1(i);
                elseif pulse_number == 13
                    count13 = count13 + 1;
                    spike13(count13) = data1(i);
                elseif pulse_number == 14
                    count14 = count14 + 1;
                    spike14(count13) = data1(i);
                end
            end
            %.........................................................................
        if (zeros_cons*sample_period > 2 || counter == length_data) && pulse_number > 0
% Calculating parameters (length, velocity,...etc) after detection is over
                detection = false;
                if pulse_number >= 4
                t12 = (st_spike2-st_spike1)*sample_period;
                D12 = width*cot(theta);
                velocity = D12/t12;
                velocityM = velocity/0.44704;
                pulse_numberD = pulse_number;
                end
                T1P = 0; T3P = 0;                           % period of 1st and 3rd pulses (tires)
                A23 = 0;
                switch pulse_number
                    case 1
                        en_class = false;
                    case 2
                        en_class = true;
                        vehicle_length = 1.5;
                        width = 0;
                    case 4
                        en_class = true;
                        t13 = (st_spike3-st_spike1)*sample_period;    % added only for data
analysis!
                        vehicle_length = ((st_spike3 - st_spike1)* sample_period)* velocity;
                        T1P = count1 * sample_period;
                        T1D = T1P;
                        T3P = count3 * sample_period;
                        T3D = T3P;
                    case 6
                        en_class = true;
                        vehicle_length = ((st_spike5 - st_spike1)* sample_period)* velocity;
                        A23 = ((st_spike5 - st_spike3)* sample_period)* velocity
                    case 8
                        en_class = true;
                        vehicle_length = ((st_spike7 - st_spike1)* sample_period)* velocity;
                    case 10
                        en_class = true;
                        vehicle_length = ((st_spike9 - st_spike1)* sample_period)* velocity;
                        A23 = ((st_spike5 - st_spike3)* sample_period)* velocity;
                    case 12
                        en_class = true;
                        vehicle_length = ((st_spike11 - st_spike1)* sample_period)* velocity;
                        A23 = ((st_spike5 - st_spike3)* sample_period)* velocity;
                    otherwise
                        if pulse_number >= 14 && mod(pulse_number,2)== 0                  %
for even number of pulses that is more than 14
                            en_class = true;
                             before_last = genvarname(['st_spike',num2str(pulse_number-1)]);
                             vehicle_length = ((eval([before_last]) - st_spike1)*
sample_period)* velocity;
                        else
                             en_class = false;
                        end
                end

                if en_class
                vehicle_count = vehicle_count + 1;
                class = Classification(pulse_number,vehicle_length,T1P,T3P,A23)
                else false_trigger = true;
                    disp('false_trigger')
                end

                signal_state1 = 0; signal_state2 = 0;
                pulse_number = 0; zeros_cons = 2/sample_period;
                clear st_pulse1 st_pulse2 st_pulse3 st_pulse4 st_pulse5 st_pulse6 st_pulse7
st_pulse8 st_pulse9;
                clear st_pulse10 st_pulse11 st_pulse12 st_pulse13 st_pulse14;
                clear en_pulse1 en_pulse2 en_pulse3 en_pulse4 en_pulse5 en_pulse6en_pulse7
en_pulse8 en_pulse9;
                clear en_pulse10 en_pulse11 en_pulse12 en_pulse13 en_pulse14;
                count1 = 0; count2 = 0; count3 = 0; count4 = 0; count5 = 0; count6 = 0; count7
= 0; % initiating indexes for spikes arrays
                count8 = 0; count9 = 0; count10 = 0; count11 = 0; count12 = 0; count13 = 0;
count14 = 0;
                en_class = false;
           end
           end

             once_cond = detection;
    %end
end
```

Classification.m:

```
function class = Classification(pulse_number,vehicle_length,T1P,T3P,A23)

switch pulse_number
    case 2
        class = 1;
    case 4
        if vehicle_length <= 3.96 && vehicle_length > 0
            class = 2;
        elseif vehicle_length <= 5 && vehicle_length > 0
            if T1P+0.02 == T3P;
                class = 4;
            else class = 3;
            end
        elseif vehicle_length > 5
            if T1P+0.02 == T3P;
                class = 5;
            else class = 3;
            end
        else class = 0;
        end
    case 6
        if A23 < 2
            class = 6;
        elseif A23> 3
            class = 8;
        else class = 0;
        end
    case 8
        class = 7;
    case 10
        if A23 < 2
            class = 9;
        elseif A23> 3
            class = 11;
        else class = 0;
        end
    case 12
        if A23 < 2
            class = 10;
        elseif A23> 3
            class = 12;
        else class = 0;
        end
    otherwise
        class = 14;
end
```

Code developed on REECE device

DAQ_1CHcont.c:

```
/***************************DAQ_1CHcont.c***************************/
/******************************Author******************************/
/****************************Samer Rajab***************************/
/*************Diamond systems universal driver example*************/
/*******************codes were used in this code*******************/

// settings: One channel, sampling rate; 1KS/s, number of converstions; 5*10K=50KS,
fifo_depth;1024, dump_threshold; 2048
#include <stdio.h>
#include <stdlib.h>
#include <math.h>
#include <time.h>
#include <string.h>
// diamond driver includes
#include "dscud.h"


#include <sys/time.h>


// macros defined
#define SLEEP_TIME 1
#define HELIOS_MAX_AD_CHANNEL_NUMBER 15

// var declarations
BYTE result; // returned error code
DSCB dscb;   // handle used to refer to the board
DSCCB dsccb; // structure containing board settings
DSCADSETTINGS dscadsettings; // structure containing A/D conversion settings
DSCAIOINT dscaioint; // structure containing auto-calibration settings
```

```
DSCS dscs;           // used for interrupts.
DFLOAT voltage;
char volts[8];
ERRPARAMS errorParams;  // structure for returning error code and error string
int intBuff;            // temp variables of size int
long longBuff;          // temp variables of size long
float floatBuff;        // temp variables of size float
int num_conversions = 4096000;
long stop_after_transfers = 6000000;
long current_transfers;
long last_total_transfers;
long last_transfers;
int new_sample_count;
long i = 0;          // miscellaneous counter
int k = 0;
int num;

#define HELIOS_DEFAULT_BASE_ADDRESS 0x280

//Main function

int main()
{
        //=============================================================================
        // I. DRIVER INITIALIZATION
        //
        //    Initializes the DSCUD library.
        //
        //=============================================================================

        if ( dscInit ( DSC_VERSION ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                fprintf ( stderr, "dscInit error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        //=============================================================================
        // II. BOARD INITIALIZATION
        //
        //         Initialize the HELIOS board. This function passes the various
        //         hardware parameters to the driver and resets the hardware.
        //
        //=============================================================================

        printf ( "\nHELIOS BOARD INITIALIZATION:\n" );

        dsccb.base_address = HELIOS_DEFAULT_BASE_ADDRESS;

        dsccb.int_level = 5;

        if ( dscInitBoard ( DSC_HELIOS, &dsccb, &dscb ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                fprintf ( stderr, "dscInitBoard error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        //=============================================================================
        // III. AD SETTINGS INITIALIZATION
        //
        //          Initialize the structure containing the AD conversion settings and
        //              then pass it to the driver.
        //
        //=============================================================================

        printf ( "\nAD SETTINGS INITIALIZATION\n" );

        memset(&dscadsettings, 0, sizeof(DSCADSETTINGS));

        dscadsettings.range = RANGE_10;

        dscadsettings.polarity = 0;
        // The Helios SBC only has a 10V physical range. This
        //   struct member is for backward compatibility.
        dscadsettings.range = RANGE_10;

        dscadsettings.gain = 0;
        dscadsettings.load_cal = 0;

        dscadsettings.current_channel = 0;

        if ( ( result = dscADSetSettings ( dscb, &dscadsettings ) ) != DE_NONE )
        {
                dscGetLastError(&errorParams);
                fprintf ( stderr, "dscADSetSettings error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                return 0;
        }

        // Interrupt settings

        printf ( "\nI/O INTERRUPT SETTINGS INITIALIZATION\n" );

        memset(&dscaioint, 0, sizeof(DSCAIOINT));

        dscaioint.num_conversions = num_conversions;

        dscaioint.conversion_rate = 1000; // If all 16 channel are operational the
rate=rate/16

        dscaioint.cycle = 1;

        dscaioint.internal_clock  = 1;

        dscaioint.low_channel = 0;

        dscaioint.high_channel = 0;

        dscaioint.external_gate_enable = 0; // can enable it if need be

        dscaioint.internal_clock_gate = 0;   // can enable it if need be

        dscaioint.fifo_enab = 1;


        dscaioint.fifo_depth = 1024;


        // Dump Threshold here is the threshold at which the data from
        //   kernel space is copied to the structure submitted in user space.


        dscaioint.dump_threshold = 1024;

        // allocate space for buffer
        // our samples buffer(Team)
        dscaioint.sample_values = (DSCSAMPLE*)malloc( sizeof(DSCSAMPLE) *
dscaioint.num_conversions );




                if ( ( result = dscADSampleInt ( dscb, &dscaioint ) ) != DE_NONE ) // Do the
sampling process and if the result is zero that means no error happend
                {
                // this function is going to sample the input and put it in buffer passed in
dscaioint.(team)
                //an error happend so log it.
                        dscGetLastError(&errorParams);
                        fprintf ( stderr, "dscADSampleInt error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                        free( dscaioint.sample_values ); // remember to deallocate malloc()
memory
                        return 0;
                }


                ///////////////////////////
                // Check Interrupt status
```

```
            //////////////////////////
            dscs.transfers = 0;
            dscs.overflows = 0;
            dscs.op_type = OP_TYPE_INT;

            current_transfers = 0;
            last_total_transfers = 0;
            last_transfers = 0;
            do
            {

            do
            { dscGetStatus(dscb, &dscs);
            dscSleep(SLEEP_TIME);
        }while(dscs.total_transfers - last_total_transfers < 10);

            printf("point4\n");

        if ( dscs.overflows )
            {
            printf("Operation failed: FIFO overflowed\n");
            break;
        }
        if ( dscs.total_transfers == last_total_transfers )
            {
            printf("Operation failed: no new samples taken in %d ms\n", SLEEP_TIME);
            break;
        }

        new_sample_count = dscs.total_transfers - last_total_transfers;

        /* Number of new samples should never exceed the size of the circular buffer.  If
         * it does it means that either "sleep_ms" should be smaller so you check status
         * more often, or "num_conversions" should be bigger so the circular buffer is bigger
         */
        if ( new_sample_count > num_conversions )
            {
            printf("Operation failed: not processing data fast enough.  %d samples lost\n",
                new_sample_count - num_conversions);
            break;
        }

             current_transfers = dscs.transfers;
             last_total_transfers = dscs.total_transfers;
        if ( current_transfers > last_transfers )
            {                        printf("point3\n");
            for ( i = last_transfers; i < current_transfers; i++ )
                {
                                if ( dscADCodeToVoltage ( dscb, dscadsettings,
dscaioint.sample_values[i], &voltage ) != DE_NONE)
                                {
                                        dscGetLastError(&errorParams);
                                        fprintf( stderr, "dscADCodeToVoltage error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                                        free ( dscaioint.sample_values );
                                        return 0;
                                }

                                sprintf(volts,"%6.3lf\n", voltage );
                                socket_client(volts);
                                }

        /* Case two: more data has been placed in the circular buffer both after
         * the "last_transfers" and before it at the start of the buffer.
         */
        } else if ( current_transfers <= last_transfers )
            {                        printf("point2\n");

            for ( i = last_transfers; i < num_conversions; i++ )
                                {
                                if ( dscADCodeToVoltage ( dscb, dscadsettings,
dscaioint.sample_values[i], &voltage ) != DE_NONE)
                                {
                                        dscGetLastError(&errorParams);
                                        fprintf( stderr, "dscADCodeToVoltage error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                                        free ( dscaioint.sample_values );
                                        return 0;
                                    }

                                    sprintf(volts,"%6.31f\n", voltage );
                                    socket_client(volts);
                                    }

            for ( i = 0; i < current_transfers; i++ )
                        {                       printf("point3\n");
                                    if ( dscADCodeToVoltage ( dscb, dscadsettings,
dscaioint.sample_values[i], &voltage ) != DE_NONE)
                                    {
                                        dscGetLastError(&errorParams);
                                        fprintf( stderr, "dscADCodeToVoltage error: %s %s\n",
dscGetErrorString(errorParams.ErrCode), errorParams.errstring );
                                        free ( dscaioint.sample_values );
                                        return 0;
                                    }
                                    sprintf(volts,"%6.31f\n", voltage );
                                    socket_client(volts);
                                    }
        }
        last_transfers = current_transfers;
        // condition to break when total number of samples reaches desired value. (This is
currently being kept for expirement only)
                if ( last_total_transfers >= stop_after_transfers )
                {

                break;
                }
      }while ( 1 );



        free( dscaioint.sample_values );

        dscFree();

        printf ( "\nDSCADSampleInt completed.\n" );

        return 0;
}
```

socket_client.c:

```
/**************************socket_client.c**************************/
/*****************************Author*****************************/
/***************************Samer Rajab***************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>

#define ADDRESS     "/root/con"  /* addr to connect */


void socket_client(char voltage[])
{
    char c;
    FILE *fp;
    register int i, s, len;
    struct sockaddr_un saun;

    /*
     * Get a socket to work with.  This socket will
     * be in the UNIX domain, and will be a
     * stream socket.
     */
    if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) < 0) {
        perror("client: socket");
        exit(1);
    }

    // To which address this connection is going
    saun.sun_family = AF_UNIX;
    strcpy(saun.sun_path, ADDRESS);

    /*
     * Try to connect to the address.  For this to
     * succeed, the server must already have bound
     * this address, and must have issued a listen()
     * request.
     *
     * The third argument indicates the "length" of
     * the structure, not just the length of the
     * socket name.
     */
    len = sizeof(saun.sun_family) + strlen(saun.sun_path);

    if (connect(s, &saun, len) < 0) {
        perror("client: connect");
        exit(1);
    }


    // Sending the string containing voltage value to server

       send(s, voltage, strlen(voltage), 0);

    // close socket
    close(s);

}
```

socket_server.c:

```
/*************************socket_server.c************************/
/****************************Author****************************/
/***************************Samer Rajab***************************/
#include <sys/types.h>
#include <sys/socket.h>
#include <sys/un.h>
#include <stdio.h>
#include <stdlib.h>
#include <string.h>
#include <pthread.h>
#include <math.h>
#include <time.h>

#define NUM_THREADS     3
#define ADDRESS     "/root/con"  /* addr to connect */
pthread_mutex_t a_mutex = PTHREAD_MUTEX_INITIALIZER;


FILE* handle;
void Classification(int pulse_number,float vehicle_length,float T1P,float T3P,float A23,float
velocity)
{
FILE* handleres;
float Mvelocity = velocity/0.44704;
time_t t = time(NULL);

struct tm tm = *localtime(&t);                              //defining time stamp to add to
file's name

int class;
switch (pulse_number)
        {
    case 2:
        class = 1;
              break;
    case 4:
        if (vehicle_length <= 3.96 && vehicle_length > 0)          // Some general bounderies
are taken from FHWA traffic monitoring guide sec4 table Table 4-A-1
```

```
            class = 2;
                // Other thresholds and bounderies are assuemd and maybe adjusted to have
better perfomance
        else if (vehicle_length <= 5 && vehicle_length > 0)
            {
                    if (T1P+0.02 == T3P)
                class = 4;
            else class = 3;
            }
        else if (vehicle_length > 5)
            {
                    if (T1P+0.02 == T3P)
                class = 5;
            else class = 3;
            }
        else class = 0;
       break;
    case 6:

              if (A23 < 2)
            class = 6;
        else if (A23> 3)
            class = 8;
        else class = 0;

        break;
    case 8:
        class = 7;
              break;
    case 10:
        if (A23 < 2)
            class = 9;
        else if (A23> 3)
            class = 11;
        else class = 0;
       break;
    case 12:
        if (A23 < 2)
            class = 10;
        else if (A23> 3)
            class = 12;
        else class = 0;
        break;
    default:
        class = 14;
              break;
        }

handleres = fopen("/root/result.txt", "a+");
fprintf(handleres,"Time:%d-%d-%d %d:%d:%d,   Vehicle Class:%d,        vehicle speed:%f,
    Number of tires:%d    \n", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour,
tm.tm_min, tm.tm_sec,class,Mvelocity,pulse_number);
printf("Time:%d-%d-%d %d:%d:%d,        Vehicle Class:%d,       vehicle speed:%f,     Number of
tires:%d    \n", tm.tm_year + 1900, tm.tm_mon + 1, tm.tm_mday, tm.tm_hour, tm.tm_min,
tm.tm_sec,class,Mvelocity,pulse_number);
fclose(handleres);

}

void *Feature_Extraction(int test)    // function used for vehicle feature extraction and
classification
{       printf("spoint1\n");
        float *vehicle_data;
        vehicle_data=test;
        int data_pointer = 1,counter,i, signal_state1 = 0, signal_state2 = 0,
zeros_cons=100000,en_class=0;
        int number_of_starts = 0, number_of_ends = 1,previous_pulse = -1000000,pulse_number =
0,length_data;
        int
st_spike1,st_spike2,st_spike3,st_spike4,st_spike5,st_spike6,st_spike7,st_spike8,st_spike9,st_s
pike10,st_spike11,st_spike12,st_spike13,st_spike14,st_spike;
        int
en_spike1,en_spike2,en_spike3,en_spike4,en_spike5,en_spike6,en_spike7,en_spike8,en_spike9,en_s
pike10,en_spike11,en_spike12,en_spike13,en_spike14;
        float sampling_rate = 1000, sample_period = 1/sampling_rate, velocity=0, Awidth =
1.762,vehicle_length,t13,T1D,T3D;
        double pi = 4.0*atan(1.0), theta = M_PI/4 ;
        float T1P = 0, T3P = 0,A23 = 0;                          // period of 1st and 3rd pulses
(tires)
     int detection, ones_cond = 1;
        //printf("spoint4\n");
        length_data = (int)floorf (vehicle_data[0]);    // The length of data is added to
vehicle data array (first field)

while(ones_cond)
{
        for(counter=data_pointer;counter<length_data+1;counter++)       //for loop to examine
all data points and extract vehicles
                {
                if (data_pointer == 1 && counter == data_pointer)              // pointer to lock
on data point index (if multiple vehicles are present in the data)
                i = 1;
                else i = i+1;

        if (vehicle_data[i] > 0.1)                                 //positive threshold for pulse
detection (adjustable)
        {
                signal_state2 = signal_state1;
        signal_state1 = 1;
        zeros_cons = 0;
        }
        else if (vehicle_data[i] < -0.25)                                    //negative
threshold for pulse detection
        {
                signal_state2 = signal_state1;
        signal_state1 = -1;
        zeros_cons = 0;
        }
        else zeros_cons = zeros_cons+1;                    // counter for consequitive zeros
to know when to a pulse ends or a vehicle ends
        if (signal_state1 == 1 && signal_state2 != signal_state1)      // Pulse detection
(changing sign of data samples)
        {
                if (counter > (previous_pulse + (0.01/sample_period)))         //duration since
last pulse (to avoid multiple detections of the same pulse)
                {
                pulse_number = pulse_number + 1;
                switch (pulse_number)                                                  //detection
of pulses
                        {                               case 1:
                                        st_spike1 = counter;
                                break;
                                case 2:
                                        st_spike2 = counter;
// When the 2nd pulse arrives then assume that it's the first axle of an arriving vehicle and
start classification
                                        velocity = (Awidth*tan(theta))/((st_spike2-
st_spike1)*sample_period);
                                break;
                                case 3:
                                        st_spike3 = counter;
                                break;
                                case 4:
                                        st_spike4 = counter;
                                break;
                                case 5:
                                        st_spike5 = counter;
                                break;
                                case 6:
                                        st_spike6 = counter;
                                break;
                                case 7:
                                        st_spike7 = counter;
                                break;
                                case 8:
                                        st_spike8 = counter;
                                break;
                                case 9:
                                        st_spike9 = counter;
                                break;
                                case 10:
                                        st_spike10 = counter;
                                break;
                                case 11:
                                        st_spike11 = counter;
                                break;
                                case 12:
                                        st_spike12 = counter;
                                break;
                                case 13:
                                        st_spike13 = counter;
                                break;
                                case 14:
                                        st_spike14 = counter;
                                break;
                                default:
                                if (pulse_number >14)
                                st_spike = counter;
                                else{
                                perror("Error: Number of pulses higher than 14");
                                exit(1);
                                }
                                break;
                        }
                previous_pulse = counter;
        number_of_starts = number_of_starts+1;
                }
        }
                if (zeros_cons*sample_period > 0.1 && number_of_starts == number_of_ends)
        //When to call an end of a pulse (based on axle spacing) pay attention, might be wrong
at high speed
            {
                switch (pulse_number)
                                {
                                        case 1:
                                                en_spike1 = counter-zeros_cons;
                                        break;
                                        case 2:
                                                en_spike2 = counter-zeros_cons;
                                        break;
                                        case 3:
                                                en_spike3 = counter-zeros_cons;
                                        break;
                                        case 4:
                                                en_spike4 = counter-zeros_cons;
                                        break;
                                        case 5:
                                                en_spike5 = counter-zeros_cons;
                                        break;
                                        case 6:
                                                en_spike6 = counter-zeros_cons;
                                        break;
                                        case 7:
                                                en_spike7 = counter-zeros_cons;
                                        break;
                                        case 8:
                                                en_spike8 = counter-zeros_cons;
                                        break;
                                        case 9:
                                                en_spike9 = counter-zeros_cons;
                                        break;
                                        case 10:
                                                en_spike10 = counter-zeros_cons;
                                        break;
                                        case 11:
                                                en_spike11 = counter-zeros_cons;
                                        break;
                                        case 12:
                                                en_spike12 = counter-zeros_cons;
                                        break;
                                        case 13:
                                                en_spike13 = counter-zeros_cons;
                                        break;
                                        case 14:
                                                en_spike14 = counter-zeros_cons;
                                        break;
                                        default:
                                        perror("Error: Number of pulses higher than 14");
                                        exit(1);
                                        break;
                                }
                        number_of_ends=number_of_ends+1;
                        }
                    if( (velocity>0 && velocity*zeros_cons*sample_period>10.668) ||(counter >=
length_data-1))            //start feature extraction assuming a maximum axle spacing of
10.668m or if exceeding two seconds or when reaching end of data
                            {
                    //When to call an end of a pulse (based on axle spacing)
                            switch (pulse_number)
                                                                                   // This will
include calculation of parameters necessary for classification
                    {
                                        case 1:
                          en_class = 0;
                                        break;
                          case 2:
                              en_class = 1;
                              vehicle_length = 1.5;
                              Awidth = 0;
                                        break;

                                        case 4:


                                                  en_class = 1;
                              t13 = (st_spike3-st_spike1)*sample_period;    // added only for data
analysis!
                              vehicle_length = ((st_spike3 - st_spike1)* sample_period)* velocity;
                              T1P = (en_spike1-st_spike1) * sample_period;
                              T1D = T1P;
                              T3P = (en_spike3-st_spike3) * sample_period;
                              T3D = T3P;
                                                  break;

                                        case 6:

                                                  en_class = 1;
                              vehicle_length = ((st_spike5 - st_spike1)* sample_period)* velocity;
                              A23 = ((st_spike5 - st_spike3)* sample_period)* velocity;
                                                  break;

                                        case 8:

                                                  en_class = 1;
                              vehicle_length = ((st_spike7 - st_spike1)* sample_period)* velocity;
                                                  break;

                                        case 10:

                                                  en_class = 1;
                              vehicle_length = ((st_spike9 - st_spike1)* sample_period)* velocity;
                              A23 = ((st_spike5 - st_spike3)* sample_period)* velocity;
                                                  break;

                                        case 12:

                                        en_class = 1;
                              vehicle_length = ((st_spike11 - st_spike1)* sample_period)* velocity;
                              A23 = ((st_spike5 - st_spike3)* sample_period)* velocity;
                                                  break;

                                        default:
                         if (pulse_number > 13 && (pulse_number%2) == 0)                   // for
even number of pulses that is more than 14
                                                      {
                                                      en_class = 1;
                                   vehicle_length = (st_spike - st_spike2)* sample_period* velocity;
                                                          }
                                                  else
                                   en_class = 0;
                                             break;

                                   }

                           if(en_class==1)                       // if en_class = 1 then pulses had
been detected and classification is possible
                           {
                           detection = 1;
                        Classification(pulse_number,vehicle_length,T1P,T3P,A23, velocity);
//Function used for vehicle classification
                        data_pointer = counter;
                        break;
                        }
                        else if (detection == 0 || en_class == 0)      // condition to see if
detection has been performed for this data before or not if it was then this might be the
zeros reminder
                        {
                        printf("False trigger, Number of Pulses: %d\n", pulse_number);
                        data_pointer = counter;
                        break;
                        }
                        }
}
if (counter >= length_data - 1)     //condition to exit the while loop after examining all of
the data
{
 ones_cond = 0;
}
else
{       signal_state1 = 0; signal_state2 = 0; zeros_cons=100000;en_class=0;
        number_of_starts = 0, number_of_ends = 1,previous_pulse = -1000000;pulse_number = 0;
        st_spike1 = 0;st_spike2= 0;st_spike3= 0;st_spike4= 0;st_spike5= 0;st_spike6=
0;st_spike7= 0;st_spike8= 0;st_spike9= 0;st_spike10= 0;st_spike11= 0;st_spike12= 0;st_spike13=
0;st_spike14= 0;st_spike= 0;
        en_spike1 = 0;en_spike2= 0;en_spike3= 0;en_spike4= 0;en_spike5= 0;en_spike6=
0;en_spike7= 0;en_spike8= 0;en_spike9= 0;en_spike10= 0;en_spike11= 0;en_spike12= 0;en_spike13=
0;en_spike14= 0;
        velocity=0, vehicle_length= 0;t13= 0;T1D= 0;T3D= 0;
        T1P = 0; T3P = 0;A23 = 0;
}
}


        memset(vehicle_data, NULL, (length_data)+4);          //Free memory occupied by vehcile
data

        pthread_exit(NULL);
}
int get_min()
{
time_t ti = time(NULL);
struct tm tim = *localtime(&ti);
return tim.tm_min;
}

main()
{
    char c;
        float vehicle_data[200000];               // array containing data samples for each
passing vehicle
        float calibration[10000];                                      // array used for
calibration (removing the mean)
        float cal_val = 0,sum;
        int minutes = 0, cal_counter = 0, current_min = 3;
        int avg_counter;
        int verification_switch = 1;
        int count_stop = 0;
        int zeros_cons,ij,  detection=0;
        float *ip;
        int sample_count = 0;
        float volts;
        int si;
        int count;
        char carray[7];
        int add_topass[2];
        pthread_t threads[NUM_THREADS];

        FILE *fp;
    int fromlen;
    register int i, s, ns, len;
    struct sockaddr_un saun, fsaun;


// We are using stream socket to guarantee reliable connection
```

```
    if ((s = socket(AF_UNIX, SOCK_STREAM, 0)) < 0)            // socket server to receive data
from client (Data acquisition unit)
        {
        perror("server: socket");
        exit(1);
    }

    // Creating address that we'll be accepting connections at
    saun.sun_family = AF_UNIX;
    strcpy(saun.sun_path, ADDRESS);

    // Binding address to socket
    unlink(ADDRESS);
    len = sizeof(saun.sun_family) + strlen(saun.sun_path);

    if (bind(s, &saun, len) < 0) {
        perror("server: bind");
        exit(1);
    }

        while(1)                                   // an infinite loop to keep listening to get
more data from client
        {
        count_stop++;           //temp
        volts = 0;
        count = 0;
        si = 1;
    if (listen(s, 5) < 0)                  //stop and listen until you get a connection with
data point from client
        {
         perror("server: listen");
         exit(1);
    }

    // Accepting connections from client
    if ((ns = accept(s, &fsaun, &fromlen)) < 0) {
        perror("server: accept");
        exit(1);
    }

    // Read socket


        fp = fdopen(ns, "r");


                // defenitions here
                while((c = fgetc(fp)) != EOF)
                {

                        if(c == '\n')
                        break;

                        if(c == '-')
                        si = -1;            // used to compensate for "-" sign in received data
                        else
                        {carray[count] = c;
                        count++;
                        }

                 }


                 volts = strtod(&carray[0],&carray[5]);      // convert from char to float
                 volts = volts * si;
                 current_min = get_min();

                         //procedure done for calibration each 20 minutes.
                 if (detection ==0 && minutes<current_min-2)
                     {       calibration[cal_counter] = volts;
                             //printf("in calibration %d\n",cal_counter);
                             cal_counter++;
                             if (cal_counter>= 10000)
                             {
                             for (avg_counter = 0; avg_counter < 10000; ++avg_counter)
                     {sum+=calibration[avg_counter];}
```

```
                                    cal_val = sum/10000;
                                    sum = 0;
                                    cal_counter = 0;
                                    minutes = current_min;
                                    printf("calibration value = %f minutes %d\n",cal_val,minutes);
                                    }


                            }

                            volts = volts - cal_val;
                      if (volts > 0.1)                  // if data sample is higher that a threshold
raise detection flag and start recording data
                              {
                              //printf("%6.3lf\n",volts);
                               zeros_cons = 0;
                     detection = 1;
                               vehicle_data[sample_count+1] = volts;
                               //printf("%6.3lf\n",vehicle_data[sample_count]);
                               sample_count++;
                               }
                  else if (volts < -0.25)         // if data sample is lower that a threshold raise
detection flag and start recording data
                              {
                              //printf("%6.3lf\n",volts);
                               zeros_cons = 0;
                     detection = 1;
                               vehicle_data[sample_count+1] = volts;
                               //printf("%6.3lf\n",vehicle_data[sample_count]);
                               sample_count++;

                               }
                  else if ((volts < 1) && (volts > -0.25) && detection == 1) // if data sample is
close to 0 and detection flag is raised start accumulating zeros
                            {
                            // printf("%6.3lf\n",volts);
                            zeros_cons = zeros_cons+1;
                            vehicle_data[sample_count+1] = volts;
                            //printf("%6.3lf\n",vehicle_data[sample_count]);
                            sample_count++;
                            }
                     if (detection == 1 && verification_switch ==1)
                     {
                     printf("detection in progress\n");
                     verification_switch = 0;
                     }

                     if (zeros_cons/10000 > 2 && detection == 1)  // zero count is higher than
maximum axle spacing: trigger detection with recorded data
                     {
                     vehicle_data[0] = (float)sample_count;
                     detection = 0;
                     verification_switch = 1;
                     zeros_cons = 0;
                     // send vehicle_data to feature extraction

                     printf("point4\n");
                     pthread_create(&threads[0], NULL, Feature_Extraction, vehicle_data); // Create
new thread and send data to it for feature extraction and vehicle classification
                     sample_count = 0;
                     volts = 0;
                     }



    //Close socket

fclose(fp);

}
close(s);

pthread_exit(NULL);
exit(0);
}
```